SeeBeyond™ eBusiness Integration Suite

e*Xchange Partner Manager Implementation Guide

Release 4.5.2

Contents

Chapter 4

Chapter 5

Chapter 1

Introduction

This guide provides comprehensive information on implementing eBusiness solutions using the e*Xchange portion of the SeeBeyond eBusiness Integration Suite. It discusses the essentials of implementing e*Xchange, Business-to-Business Integration, and the components used in a complete e*Xchange implementation. This guide also provides detailed information on the e*Xchange architecture and its core components, as well as the e*Gate schema components that make up an e*Xchange implementation. Finally, it discusses how e*Xchange and e*Gate work together to provide a comprehensive toolset for designing, creating, and maintaining a fully functional eApplication.

1.1 Document Purpose and Scope

This guide explains how to use the SeeBeyond Technology Corporation™ (SeeBeyond™) e*Xchange Partner Manager, including:

- Understanding the e*Xchange schema components.
- Functions and methods available to the user

This guide gives the e*Xchange implementor the necessary background and methodology for getting an e*Xchange system up and running in a real-world situation. To do this, it provides detailed information on the e*Gate schema that e*Xchange uses as its back end and explains the various areas requiring configuration. This guide also contains several detailed case studies showing how to implement various features built into e*Xchange, such as how to send secure transactions.

1.2 Intended Audience

The reader of this guide is presumed to be a developer or system administrator with responsibility for developing or maintaining the e*Xchange system. The implementor should have experience of Windows NT and UNIX operations and administration, and should be thoroughly familiar with Windows-style GUI operations.

Since most of the work in an e*Xchange implementation involves setting up the e*Gate components that send data into and out of the e*Xchange system, the implementor should also have experience implementing e*Gate.

1.3 Organization of Information

The *e*Xchange Partner Manager Implementation Guide* includes the following information:

- List of Tables - Displays a list of the Tables in the document.
- List of Figures - Displays a list of Figures in the document.
- 1 - Introduction to the various applications included in the SeeBeyond eBusiness Integration Suite and the components of each. Intended audience, writing conventions, Overview of the e*Xchange Suite, purpose of guide
- 2 - Introduction to Business-to-Business Integration. Overview of the e*Xchange components.
- 3 - General overview of the e*Gate components and e*Gate architecture.
- 4 - Explanation of the design and purpose of each of the e*Gate components used in the e*Xchange schema.
- 5 - Explanation of the structure, design, and purpose of the ETD used to move data between the e*Xchange components.
- 6 - A generalized method for approaching an e*Xchange implementation, with explanations of how to accomplish some common implementation tasks.
- 7 - Case study of a simplified order processing e*Xchange implementation.
- 8 - Case study showing how to use e*Xchange to send an X12 850 purchase order to a trading partner.
- 9 - Case study showing how to use e*Xchange to send out a purchase order created by an e*Insight activity component.
- 10 - Descriptions of the specialized e*Xchange Monk functions.
- Appendix A - Example of the XML version of the ETD used by e*Xchange to exchange data.
- Appendix B - Tables
- Glossary - Definitions of technical terms specific to the e*Insight Business Process Manager, as well as some industry terms.
- Index - Index of key terms.

1.4 Writing Conventions

The writing conventions listed in this section are observed throughout this document.

Hypertext Links

When you are using this guide online, cross-references are also hypertext links and appear in **blue text** as shown below. Click the **blue text** to jump to the section.

For information on these and related topics, see **"Supporting Documents" on page 18**.

Command Line

Text to be typed at the command line is displayed in a special font as shown below.

```
java -jar ValidationBuilder.jar
```

Variables within a command line are set in the same font and bold italic as shown below.

```
stcregutil -rh host-name -un user-name -up password -sf
```

Code and Samples

Computer code and samples (including printouts) on a separate line or lines are set in the command-line font as shown below.

```
Configuration for BOB_Promotion
```

However, when these elements (or portions of them) or variables representing several possible elements appear within ordinary text, they are set in *italics* as shown below.

path and *file-name* are the path and file name specified as arguments to **-fr** in the **stcregutil** command line.

Notes and Cautions

Points of particular interest or significance to the reader are introduced with ***Note***, ***Caution***, or ***Important***, and the text is displayed in *italics*, for example:

Note: *The Actions menu is only available when a Properties window is displayed.*

User Input

The names of items in the user interface such as icons or buttons that you click or select appear in **bold** as shown below.

Click **Apply** to save, or **OK** to save and close.

File Names and Paths

When names of files are given in the text, they appear in **bold** as shown below.

Use a text editor to open the **ValidationBuilder.properties** file.

When file paths and drive designations are used, with or without the file name, they appear in **bold** as shown below.

In the **Open** field, type **D:\setup\setup.exe** where **D:** is your CD-ROM drive.

Parameter, Function, and Command Names

When names of parameters, functions, and commands are given in the body of the text, they appear in **bold** as follows:

The default parameter **localhost** is normally only used for testing.

The Monk function **iq-put** places an Event into an IQ.

After you extract the schema files from the CD-ROM, you must import them to an e*Gate schema using the **stcregutil** utility.

1.5 Supporting Documents

The following SeeBeyond documents provide additional information about e*Xchange and e*Gate:

- *SeeBeyond eBusiness Integration Suite Deployment Guide*
- *SeeBeyond eBusiness Integration Suite Primer*
- *e*Xchange Partner Manager User's Guide*
- *e*Xchange Partner Manager Installation Guide*
- *e*Gate Integrator Alert Agent User's Guide*
- *e*Gate Integrator Alert and Log File Reference Guide*
- *e*Gate Integrator Collaboration Services Reference Guide*
- *e*Gate Integrator Intelligent Queue Services Reference Guide*
- *e*Gate Integrator SNMP Agent User's Guide*
- *e*Gate Integrator System Administration and Operations Guide*
- *e*Gate Integrator User's Guide*
- *Monk Developer's Reference*
- *Standard e*Way Intelligent Adapters User's Guide*

1.6 SeeBeyond Web Site

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

http://www.SeeBeyond.com

Chapter 2

Introduction to the SeeBeyond eBI Suite

This chapter provides an overview of the SeeBeyond eBusiness Integration Suite, and explains how the e*Xchange Partner Manager fits into the Suite.

2.1 SeeBeyond eBusiness Integration Suite

This section provides an overview of the SeeBeyond eBusiness Integration Suite and its parts. It also provides a detailed overview of the e*Xchange Partner Manager and eSecurity Manager components.

Complex and dynamic partner relationships, and the management of various processes, present a tremendous challenge in eBusiness. Organizations and their trading partners are both faced with the problem of managing disparate component applications and aligning proprietary software requirements. In addition, organizations and their trading partners must agree on data exchange and security standards.

The SeeBeyond eBusiness Integration Suite merges traditional Enterprise Application Integration (EAI) and Business-to-Business (B2B) interactions into a multi-enterprise eBusiness Integration (eBI) product suite. This suite allows you to:

- Leverage your existing technology and applications.
- Create an eApplication consisting of component applications that are managed by your organization or your trading partners.
- Rapidly execute eBusiness strategies.
- Create and manage virtual organizations across the entire value chain.
- Rapidly implement industry standard business protocols.
- Quickly and easily establish new business partners, or update existing ones.
- Automatically secure transmissions sent over the public domain.

This suite also provides:

- Extensive and flexible back-office connectivity.
- Powerful data transformation and mapping facilities.
- Content-based routing.
- Unparalleled scalability based on a fully distributed architecture.

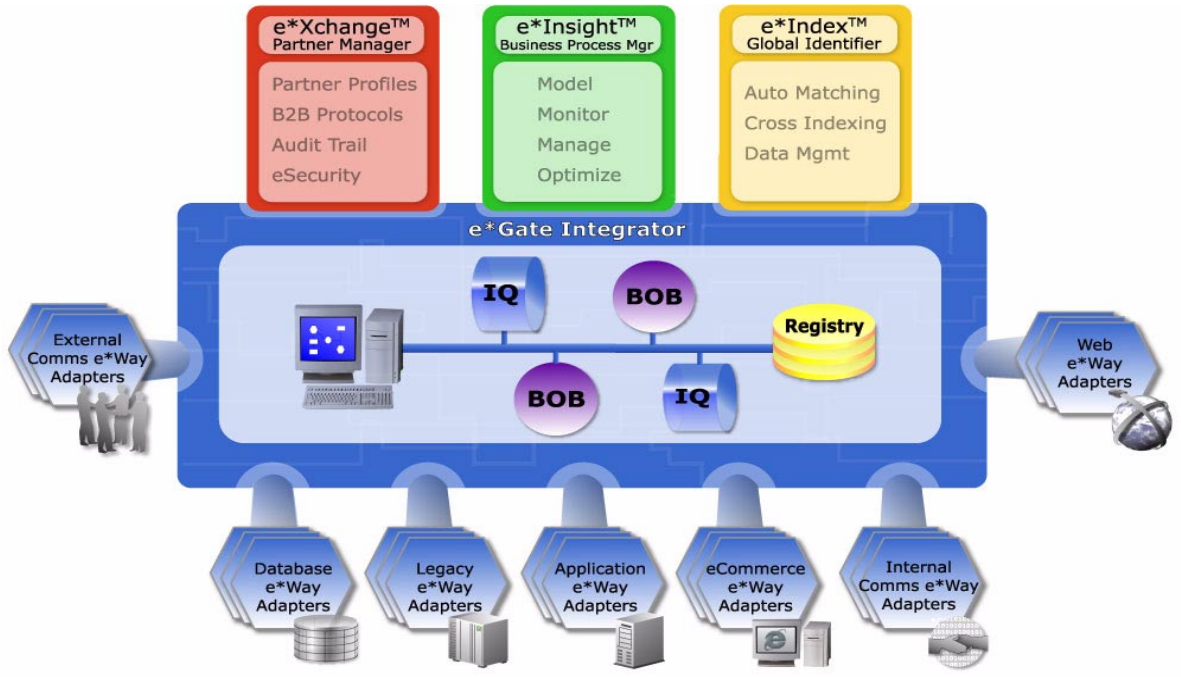
2.1.1 SeeBeyond eBusiness Integration Suite Components

The SeeBeyond eBusiness Integration Suite includes the following components and sub-components:

- eBusiness integration applications:
  - e*Insight™ Business Process Manager
  - e*Xchange™ Partner Manager
  - e*Index Global Identifier
- e*Gate™ Integrator:
  - e*Way™ Intelligent Adapters
  - Intelligent Queues (IQ™)
  - Business Object Brokers (BOBs)

See Figure 1 for a graphical representation of the SeeBeyond eBusiness Integration Suite and its components.

Figure 1 SeeBeyond eBusiness Integration Suite



e*Insight Business Process Manager

The e*Insight Business Process Manager facilitates the automation and administration of business process flow across eBusiness activities. Through graphical modeling and monitoring, business analysts can instantly assess the detailed state of a business process instance and identify bottlenecks in the process.

e*Xchange Partner Manager

The e*Xchange Partner Manager manages trading partner profiles and supports standard eBusiness message format and enveloping protocols, including RosettaNet, UN/EDIFACT, ASC X12, and BizTalk. The e*Xchange Partner Manager includes a Validation Rules Builder to aid in the creation of X12 and UN/EDIFACT message validation based on industry implementation guides.

The eSecurity Manager authenticates and ensures full integrity of message data sent to and from trading partners, which is imperative when conducting eBusiness over the public domain. The eSecurity Manager uses public key infrastructure (PKI) to ensure origin authentication of the sender and encryption ensures business messages remain secure and private.

e*Index Global Identifier

e*Index Global Identifier (e*Index) is a global cross-indexing application that provides a complete solution for automated person-matching across disparate source systems, simplifying the process of sharing member data between systems.

e*Index centralizes information about the people who participate throughout your business enterprise. The application provides accurate identification and cross-referencing of member information in order to maintain the most current information about each member. e*Index creates a single, consistent view of all member data by providing an automatic, common identification process regardless of the location or system from which the data originates.

e*Gate Integrator Components

e*Gate Integrator enables the flow of information across an extended enterprise by providing comprehensive connectivity to applications and datastores across a network. e*Gate is based on a distributed architecture with an open design that deploys flexible load balancing options. e*Gate processes Events according to user-defined business logic and integrates business processes between applications, ensuring end-to-end data flow into back-office systems.

e*Way Intelligent Adapters

e*Way Intelligent Adapters provide specialized application connectivity and also provide support for robust data processing such as business Collaborations, transformation logic, and publish/subscribe relationships. e*Way adapters are multi-threaded to enable high-performance distributed processing capabilities. This multi-threaded processing allows for ultimate deployment flexibility and load balancing.

Intelligent Queues

Intelligent Queues (IQs) are open-queue services for SeeBeyond or third-party queuing technology that provide robust data transport.

In conjunction with Java-enabled Collaborations, SeeBeyond JMS IQs can provide guaranteed exactly once delivery of messages.

Business Object Brokers

A BOB component is similar to an e*Way in the sense that it establishes connectivity and is capable of data transformation. BOBs use Collaborations to route and transform data within the e*Gate system. They have the following properties:

- They only communicate with IQs within e*Gate. They do not communicate with external applications as e*Ways do.
- They are optional by design. You can add them to an environment to remove some load from your e*Ways, either to set up easily maintainable data processing or to enable multiple internal processes.

2.2 Building an eApplication

An eApplication is an integrated collection of software that enables you to model and manage an eBusiness. The SeeBeyond eBusiness Integration Suite provides the glue and essential building blocks that allow you to create a composite eApplication for running your eBusiness.

Implementing e*Xchange Partner Manager involves the following steps:

1. Install and learn the basics of e*Xchange.

   Use the *e*Xchange Partner Manager Installation Guide* to help you install the e*Xchange software. See the *e*Xchange Partner Manager User 's Guide* for overview information and details on using the e*Xchange GUIs.

2. Obtain a working knowledge of e*Xchange.

   Read chapters 1 through 3 of this Guide to comprehend the technical architecture of e*Xchange, its components, and how they work together with e*Gate back-end components. This provides the foundation for implementing a working end-to-end eBusiness scenario.

3. Create an implementation plan.

   Use this manual as a guide for preparing a step-by-step roadmap of your implementation. This book describes several different types of e*Xchange implementations. Use these as the basis for planning the e*Xchange implementation best suited to your business needs.

Chapter 3

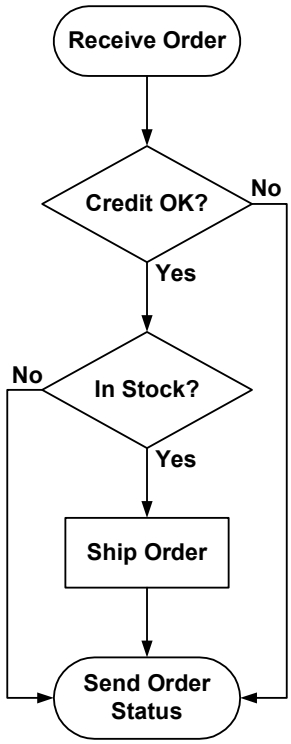
Business-to-Business Integration

Electronic Business-to-Business Integration, or eBusiness Integration (eBI), does more than allow one business to send electronic documents to another. eBI automates and integrates the entire business supply chain so that a business process that uses external trading partners can be managed as a single process. In moving from intra-business to inter-business, the integrator must overcome several challenges, most of which stem from the need to use infrastructure that is outside one's control. Once these challenges are overcome, the enterprise can manage the entire end-to-end business process and extend the proven planning and cost savings abilities of Enterprise Application Integration (EAI) to the larger world of eBI.

3.1 An eBI Example

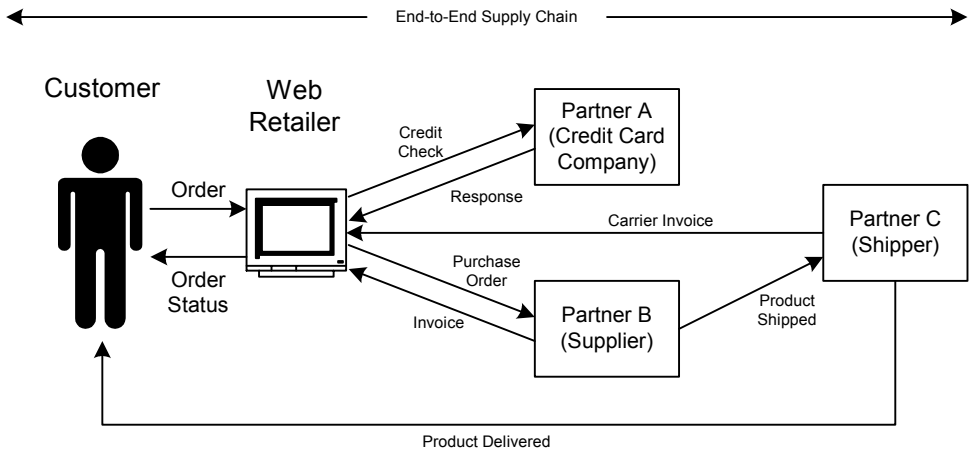
The need to integrate a number of trading partners is an essential requirement in the realm of internet retailing. For example, consider a Web retailer that sells sports equipment online. This retailer sets up an electronic storefront that allows a customer to browse an online catalog of items and place orders for them. After securing payment via credit card, the items are shipped to the customer, along with the status of the order. Figure 2, on the next page, shows a flow chart of the Web retailer's business process outlining the steps involved in a typical transaction.

Figure 2 Web Retailer Business Process



Three out of the five steps in this business process (checking credit, stock availability, and shipping to the customer) are outside the Web retailer's enterprise. However, from the customer's point of view, the entire transaction is handled by the online retailer. The Web retailer's business model depends on the efficient use of trading partners to fulfill parts of the business transaction that he does not handle directly. Figure 3 shows the interrelationships between the retailer and the trading partners.

Figure 3 Trading Partner Relationships



The goal of eBI is to successfully integrate the trading partner relationships into the overall business process in order to create a composite eApplication.

3.2 How Is eBI different from EAI?

The necessity to coordinate the information systems of multiple trading partners outside one's own control is the main difference between eBusiness Integration and traditional EAI.

3.2.1 Traditional EAI

Traditional Enterprise Application Integration focused on getting a company's in-house business management software applications to work together, and on improving business process efficiency by sharing data. Data sharing also made possible timely planning and analysis, which made businesses more efficient.

EAI became necessary because the specialized nature of the various tasks involved in running a business gave rise to a compartmentalized approach to handling them. Consequently, businesses often divided up the work load into departments, with each department in charge of accomplishing a specific business task. For example, the sales department took orders, the finance department received payments, the warehouse stored goods and prepared the orders, and the shipping department delivered the goods to the customer.

Each department in turn had its own computer system for keeping track of the data for which it was responsible, and periodically prepared reports to be used by the people entrusted with planning for the business as a whole. These stand-alone departmental systems usually could not communicate well with each other, because each had unique requirements for how they handled data. This inability to share data limited inter-departmental planning or business level planning, and any suggestions for business improvement had to wait for each department's reports to be produced, combined together, and reconciled.

EAI solutions improved business integration dramatically. By allowing the departmental applications to share data, EAI solutions made it possible to model the entire process of a business from order taking to order fulfillment, and provided the glue to hold all the pieces of the process together. Moreover, business planners could now do real-time analysis of how a business was doing across all its departments and divisions, in whatever detail was required.

3.2.2 The Emerging eBI Model

eBI essentially performs the same kind of integration as EAI, but at a higher level. Instead of integrating departments, it integrates trading partners. Because these trading partners are autonomous businesses, the integration itself must be more flexible and based on cooperation. Moreover, this integration needs to use the public electronic infrastructure such as the Internet and Value Added Networks (VANs), and established business protocols such as X12, UN/EDIFACT, RosettaNet, and CIDX, that

any business can utilize. The challenge for businesses implementing an eBI model of integration is to find ways to achieve the same level of business process tracking and planning that are gained with EAI, within this looser structure.

3.3 Meeting the Challenges of eBI

As the logical next step in business integration, eBI faces all the challenges faced by traditional EAI, with two other important additions:

- It must support autonomous trading partners
- It must be able to use the public electronic infrastructure

3.3.1 Meeting the EAI Challenge

Given the vast range of ways to exchange electronic information, so many data formats, transmission protocols, and different types of software, simply making the connection between these disparate systems is a significant technological challenge. Once these disparate components are connected, companies face a further challenge to manage and monitor the entire system. e*Xchange addresses these issues by using e*Gate, the most powerful suite of tools for Enterprise Application Integration.

e*Gate's reliable, flexible, scalable, and distributed architecture, combined with data transformations, enables you to manipulate data whenever, wherever, and however you wish. This solid base, combined with the wide range of e*Way communication adapters that can exchange data between almost any software and hardware, means that much of the work of integration and implementation has already been done for you.

3.3.2 Meeting the Trading-Partner Challenge

In a traditional EAI project, even though you must integrate different computer systems, you always enjoy the security of knowing that ultimately you have control of the entire composite system. Unfortunately, you do not enjoy this same sense of control within an eBI configuration; there are autonomous entities outside your enterprise—and outside your control.

Fortunately, there are standards that provide "rules of engagement" between entities in an eBI chain: the well established eBusiness protocols such as X12, UN/EDIFACT, RosettaNet, and CIDX, which any business can use to exchange electronic business documents. By supporting these standards, e*Xchange gives you a powerful way to integrate beyond your enterprise. e*Xchange includes built-in support for the standard versions of X12, UN/EDIFACT, RosettaNet, and CIDX, and includes a tool for building customized versions of some of these standards for your particular industry.

3.3.3 Meeting the Challenge of Using Public Domains

The implementation of a traditional EAI project occurs behind the safety of a company's firewall. This type of isolated integration environment is unavailable in an eBI implementation. Just as every business must use the existing public transportation infrastructure to move its physical goods to market, so too must an eBusiness use the public Internet and Value Added Networks open to every business.

Public networks provide opportunities for unauthorized users to access your sensitive data. e*Xchange uses safe and secure ways to carry on electronic commerce, and includes support for sending encrypted messages over secure channels. By using a combination of the proven public-key approach for sending secure messages over unsecured channels, and support for the HTTPS protocol for securely connecting two computers, e*Xchange has features that make eBusiness safe and secure.

3.4 The Benefits of eBI

Despite its challenges, eBI has definite benefits:

- Increased efficiency
- The ability to track an individual business transaction through the entire supply chain
- The ability to analyze your business model

3.4.1 Increased Efficiency

Sharing data electronically vastly increases efficiency. Every paper-based transfer of information brings with it the risk of introducing error and inefficiency. Every time business data is re-entered into another system by hand, the cost of doing so is added to the transaction, as is the cost of correcting the errors that this type of transfer inevitably creates.

There is also a latency problem in getting the data to its intended destination; even with "overnight delivery," the time it takes a paper transaction to be delivered physically is significantly longer than the time it takes to deliver it electronically.

3.4.2 Tracking Complete Business Transactions

By tying all the trading partners that handle steps in your business process into a single end-to-end configuration, you can track the entire business transaction from beginning to end.

3.4.3 Business Model Analysis

Because you can track the entire business process from end to end, you can analyze, over time, how your model is performing. You can identify bottlenecks and make intelligent decisions about how to improve your process.

Chapter 4

Understanding e*Gate Integrator

e*Gate Integrator (e*Gate), the application suite that contains the executable components and modules that actually move data from one point to another, is the foundation of an e*Xchange system. Implementing e*Xchange requires a fundamental understanding of e*Gate and the skills required to configure components in e*Gate's graphical interfaces.

This chapter provides a brief overview of e*Gate's architecture and components. More information on using e*Gate can be found in the *e*Gate Integrator System Administrator and Operations Guide* and the *e*Gate Integrator User's Guide*.

4.1 e*Gate Architecture

e*Gate is based on a distributed and open architecture, allowing components to reside on different workstations within a global network. Based on which communication protocols and adapters you choose, e*Gate can communicate with and link multiple applications and databases across a variety of operating systems.

4.1.1 Schemas

e*Gate system components are organized into schemas. A *schema* is a configuration scheme that contains all of the modules and configuration parameters that control, route, and transform data as it travels through the e*Gate system. Schemas also maintain the relationships between the components, including the publish/subscribe information that is at the heart of the data transportation process.

Whenever you define or configure components, establish data routing between components, or exchange files with the e*Gate Registry, you do so within the context of a single schema. You can also import or export schema components, or entire schemas, so that you can move a working configuration from one environment to another; for example, moving from test to production environments.

4.1.2 Components

From a functional standpoint, e*Gate components can be organized into three groups:

- Registry Host components
- Participating Host components

- Graphical User Interfaces (GUIs)

Registry Host Components

The *Registry Host* contains all of the configuration information that makes up the schema, and maintains a repository of all the configuration, executable, and application logic files required for its operation. The Registry's file repository is divided into "Sandbox" and "run time" areas to allow different users to simultaneously configure different components in the schema without conflicting with each other. When a user opens a file for editing, it is automatically downloaded to the user's Sandbox and an advisory "lock" is placed on the run time file. This lock warns other users who try to open the file that it is currently being edited by the person who opened it. In the meantime, the run time schema is unaffected by any modifications to the file until the revised file is promoted to the run time system.

For more information on codeveloping schemas in e*Gate, see *Codeveloping in e*Gate: Using the Team Registry*. For technical information about Registry Host services and directory structure, see the *e*Gate System Administration and Operations Guide*.

Participating Host Components

The executable e*Gate components reside on a *Participating Host*. The primary Participating Host components are:

- *Control Brokers*, which start, stop, and monitor all the components in the schema on a single Participating Host
- *e*Way Intelligent Adapters*, which handle the data exchange between e*Gate and external systems
- *Business Object Brokers* (BOBs), which have the same business-logic execution and data processing as e*Ways, but do not communicate with external systems

These components are discussed in greater detail later in this chapter.

When you add a Participating Host to a schema, you must ensure that the host is registered for that schema and that the Control Broker is active. This must be done directly on the machine on which you installed the Participating Host; it cannot be done remotely.

For more information on installing Participating Hosts and activating Control Brokers, see the *e*Gate System Administration and Operations Guide*.

Graphical User Interfaces (GUIs)

e*Gate incorporates a number of *Graphical User Interfaces* (*GUIs*) to streamline component configuration and simplify the task of implementing the programming logic necessary to process the data. e*Gate includes a monitoring GUI that allows you to view and resolve errors, and to start and stop components in a schema.

e*Gate GUIs include the following:

- Enterprise Manager
- SeeBeyond Collaboration Rules Editor

- SeeBeyond Collaboration-ID Rules Editor
- ETD Editor
- e*Way Editor
- e*Gate Monitor
- Alert Agent configuration tool

For more information on the e*Gate GUIs, see the *SeeBeyond eBusiness Integration Suite Primer* or the online Help system for the individual GUI.

4.2 e*Gate Components

e*Gate components are organized into schemas which contain all of the parameters, relationships, and configuration details necessary to transform and route data through the system. The following components make up an e*Gate schema:

- Users
- Event Types and Event Type Definitions
- IQ Intelligent Queues, IQ Services, and IQ Managers
- e*Way Intelligent Adapters
- BOBs
- Collaborations, Collaboration Rules, and Collaboration Services

The following sections give brief overviews of these components and their relationship to each other in a schema.

Note: *All schema components are created and configured in the Enterprise Manager. For information on how to create or configure a component, see the Enterprise Manager's online Help system.*

4.2.1 Security and e*Gate Users

Before you can configure schema components in the Enterprise Manager, you must log in as a specific *user* for that Registry Host. Requiring user authentication prevents unauthorized modifications to the e*Gate system.

e*Gate sets up an initial user called **Administrator** when you install the Registry Host. You must use the **Administrator** username and password when you log into a Registry Host for the first time. Then you can create additional users by adding them to the **Users** folder in the Enterprise Manager GUI. The user list applies to all schemas on that Registry Host, and all users can define additional users for the host.

4.2.2 Event Types and Event Type Definitions

Every *Event* (packet of data) that passes through the system is identified as a particular Event Type. An *Event Type* is a class of Events with a common data structure. For example, all Events with a specific set of fields with known characteristics and delimeters could belong to the same Event Type.

Event Type Definitions define Event Types. An *Event Type Definition* (ETD) is a programmatic representation of an Event Type used to parse, transform, or route the data through the system. Each node in the ETD represents a specifically-defined portion of an Event. Each node can also contain subnodes, so that an ETD takes on a hierarchical, tree-like structure.

You create Event Type components by adding them to the **Event Types** folder in the Enterprise Manager GUI. You create ETDs in the ETD Editor, then associate them with Event Type components. ETDs carry a file extension of **.ssc** (Monk) or **.xsc** (Java).

Each Event Type is defined by one and only one ETD. A single ETD, however, can define multiple Event Types. Having multiple Event Types utilizing a single ETD enables you to limit the number of ETD files that must be maintained in a schema.

4.2.3 IQ Intelligent Queues, IQ Services, and IQ Managers

IQ Intelligent Queues (IQs) manage the exchange of information between components within the e*Gate system, providing persistent storage for data as it passes from one component to another. When an Event leaves one component it is published to an IQ. Another component then picks up the Event from the IQ based on the Event Type under which the Event was published.

IQ Services provide the mechanism for moving Events between IQ Intelligent Queues, handling the low-level implementation of data exchange such as system calls to initialize or reorganize a database. IQ Intelligent Queues can use different IQ Services depending on the implementation requirements.

IQ activities are overseen by *IQ Managers*, which reorganize queues, archive queue information (upon request, to save disk space), and lock the queues when maintenance is performed. When you create a new schema, e*Gate automatically creates an IQ Manager component and places it under the Participating Host in the Enterprise Manager GUI. When you add additional Participating Hosts to the schema, you must manually add IQ Manager components for that host. Each schema must have at least one IQ Manager. Only one per schema is needed for most installations, but you can define additional IQ Managers if additional queue maintenance is required.

4.2.4 e*Way Intelligent Adapters

*e*Way Intelligent Adapters* transfer data between external systems and e*Gate. e*Ways perform three primary functions:

- Receive unprocessed data from external systems and package it as Events of known Event Types
- Forward Events to other components within the e*Gate system via IQ Intelligent Queues

- Send processed data to external systems

You install an e*Way by first installing the required software on the Participating Host, and then using the Enterprise Manager to add and configure the e*Way component within a schema. The most important part of the installation process is configuring the e*Way's properties and communication parameters to meet the specific application or protocol requirements of the external application to which the e*Way is connecting.

For more information on creating and configuring e*Way components, refer to the *e*Gate Integrator User's Guide*.

4.2.5 BOBs

Business Object Brokers (*BOBs*) perform the same functions as e*Ways except that they only communicate within the e*Gate system; they cannot communicate with applications outside of e*Gate. Adding BOBs to your schema can help redistribute the data processing workload or allow you to modularize multiple processes. BOBs are optional components; they are not required in order for the e*Gate system to operate correctly.

All Participating Hosts contain the software required for BOBs; you do not need to install anything separately to include BOBs within a schema.

4.2.6 Collaborations, Collaboration Rules, and Collaboration Services

Collaborations are e*Gate's data processing "powerhouses." Each Collaboration contains two parts: the publisher half publishes Events of a specific Event Type and the subscriber half "listens" for Events of a specific Event Type. Events are published to a specific IQ or to an external system. Events (Event Types) that are subscribed to must be published by other components (Collaborations or external systems) in the schema.

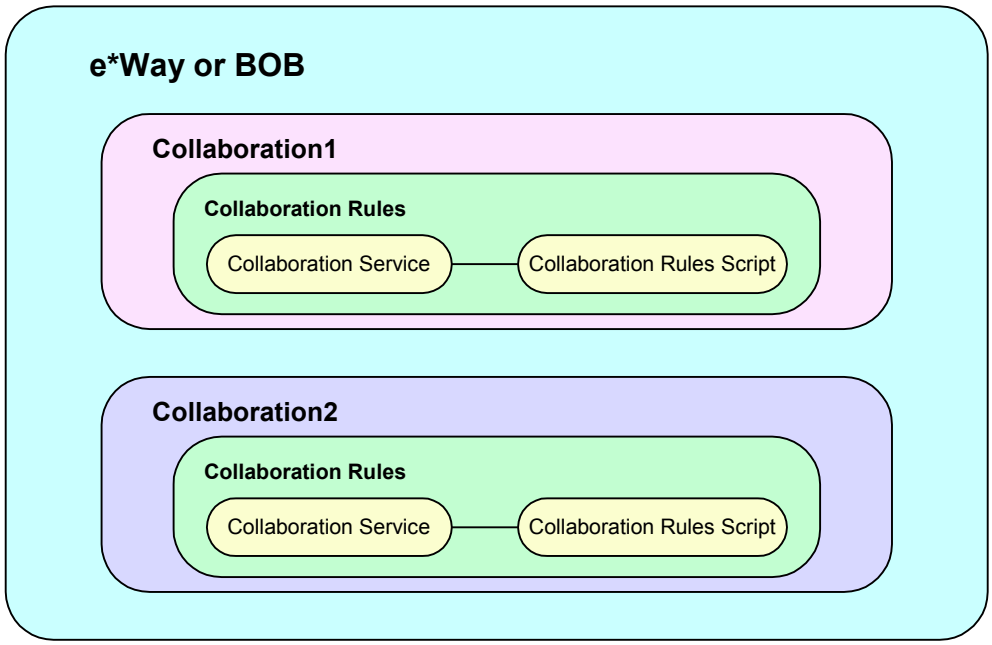
The data processing within a Collaboration is completed using Collaboration Rules. *Collaboration Rules* extract selected information from an incoming Event and process it according to a specific set of instructions in a Collaboration Rules Script (CRS). The type of CRS required depends on the Collaboration Service selected for the Collaboration Rules. *Collaboration Services* are DLLs that provide the mechanism through which e*Gate executes the rules (instructions) in the file. For example, a Collaboration Rules that use the Java Collaboration Service require a Java class file as a CRS. Collaboration Rules that use the Monk Collaboration Service require a Monk-based file as a CRS.

Collaborations are assigned to e*Ways or BOBs. An e*Way or BOB can contain multiple Collaborations, but a particular Collaboration can only be assigned to one component (e*Way or BOB) in a schema.

Each Collaboration has a single Collaboration Rules component associated with it. The Collaboration Rules component has a Collaboration Service and CRS associated with it.

The following diagram illustrates the relationship and dependencies of these components.

Figure 4 Relationships Between Components



The "embedded" design illustrated in Figure 4 allows you to create programming logic once, and reuse it in multiple places in the schema. You can reuse the programming logic in CRSs in two ways:

- By associating a single CRS with multiple Collaboration Rules components, or,
- By associating a single Collaboration Rules component with multiple Collaborations in e*Ways or BOBs.

In either case, the result is the same: Collaborations subscribing to and publishing different Event Types can perform the same functions without having to rewrite the code each time.

See *Creating an End-To-End Scenario in e*Gate* or the Enterprise Manager's Help system for detailed information about how to create components and how to link components together to create a flow of data.

Chapter 5

e*Xchange Schema Components

The e*Gate schema for e*Xchange is the e*Gate schema that implements a particular e*Xchange installation. The starting point is the e*Gate schema called **eXSchema** created when you install the e*Gate schema for e*Xchange from the installation CD. This schema contains a number of pre-configured and partially pre-configured e*Gate components used by e*Xchange. In addition to the components that are provided on the CD, a complete e*Xchange implementation requires several other e*Gate components that are added to the e*Xchange schema during the implementation process. The pre-configured components that are used, as well as the additional e*Gate components that are added to make up the final working e*Xchange schema, depends entirely on the specifics of the implementation.

The purpose of this chapter is to describe the e*Gate components provided with the eXSchema as well as those that are added in the implementation process, and discuss how each fits into and supports a working e*Xchange implementation. For each component there is a detailed drawing showing the other components with which it interacts as well as the publication and subscription information for its Collaborations. In addition, for each component we discuss: the type of component it is, its function in e*Xchange, any configuration the implementor must perform, the Collaborations it uses, and what is contained in the Events it processes.

5.1 The Purpose of the e*Gate Schema for e*Xchange

The purpose of the e*Gate Schema for e*Xchange is to provide the working portion of e*Xchange. Whereas the e*Xchange Web Interface and GUIs are primarily used to configure and monitor the e*Xchange system, the e*Gate Schema components actually move and transform the data handled by e*Xchange.

5.1.1 e*Xchange Components

The e*Xchange components manage the exchange of electronic messages with trading partners.

5.2 e*Gate schema for e*Xchange Components Overview

Table 1 lists all of the component types used by e*Xchange. It lists the components that are provided as part of the e*Gate schema for e*Xchange (eXSchema) installation, and also the components that the user adds in the implementation process. The meaning of the column headings is as follows.

- **Component**—The e*Gate logical name for the component. *Italics* indicates that the name varies by association or is user-defined.
- **Description**—A brief description of what the component does in e*Xchange.
- **In Default eXSchema**—Whether or not this component is provided as part of the back end installation of e*Xchange.
- **Configuration Required**—Most of the components in the default eXSchema require little configuration on the part of the implementor. Table 1 uses the following terms to describe the level of configuration required:
  - **No—**The component does not require any configuration or programming on the part of the implementor.
  - **Minor—**You must add the e*Xchange database connection information to the configuration file.
  - **Some—**You must make additional changes to the configuration file beyond providing the e*Xchange database connection information.
  - **Yes—**The component is mostly or entirely user-defined and must be configured and programmed by the implementor.
- **More Information**—A cross reference to the section that describes this component in detail.

Table 1 e*Xchange Back-End Component Types

Component	Description	In Default eXSchema?	Configuration Required?	More Information
eX_ePM e*Way	Handles the tracking, validating, security, and enveloping of Events sent to and from trading partners.	Yes	Minor	5.3.1 on page 39
eX_ePM_Ack_Monitor e*Way	Handles the process of resending to trading partners Events for which no acknowledgment has been received. For X12 and UN/EDIFACT, this e*Way sends the message to a staging area. For RosettaNet, it sends the message out to the queue.	Yes	Minor	5.3.2 on page 43
eX_ePM_Batch e*Way	Handles the process of bundling together transactions to be sent out as a group to a single trading partner.	Yes	Minor	5.3.3 on page 45
eX_ePM_Trans_Poll	For X12 and UN/EDIFACT, handles the process of sending out interactive Events that require acknowledgments. This is also used for resend messages from the Web Interface.	Yes	Minor	5.3.4 on page 47
eX_Batch_to_Trading_ Partner e*Way	Sends out Events to trading partners in batch (FTP) mode.	Yes	No	5.3.5 on page 48
eX_Https_to_ Trading_ Partner e*Way	Sends out Events to trading partners using a secure HTTPS (encrypted) or unsecure HTTP (not encrypted) communication protocol.	Yes	No	5.3.6 on page 50
eX_Poll_Receive_FTP	Polls the e*Xchange database for information on trading partners in batch (FTP) mode. This information is passed to the eX_Batch_From_Trading_Partner e*Way.	Yes	Minor	5.3.7 on page 51
eX_Batch_From_Tradin g_Partner e*Way	Receives Events from trading partners in batch (FTP) mode.	Yes	Minor	5.3.8 on page 52
eX_Mux_from_ Trading_Partner e*Way	Sends and receives Events from trading partners using a Web server.	Yes	No	5.3.9 on page 54
eX_POP3_from_ Trading_ Partner e*Way	Receives events via email.	Yes	No	5.3.10 on page 57
eX_SMTP_to_ Trading_ Partner e*Way	Sends out Events to trading partners via email.	Yes	No	5.3.11 on page 58
Send_to_ePM e*Way	Prepares Events coming from a business application for processing by e*Xchange.	Yes	Yes	5.3.12 on page 59
Receive_from_ePM e*Way	Prepares Events coming from e*Xchange for use by a business application.	Yes	Yes	5.3.13 on page 61
eX_from_Trading _Partner e*Way	Prepares Events coming from trading partners for processing by e*Xchange.	No	Yes	5.3.14 on page 61

5.2.1 e*Xchange Schema Component Relationships Diagram

Figure 6 on the next page illustrates the relationships among the e*Xchange schema components. Not every e*Xchange implementation uses all of these components. Some of the components shown are not provided as part of the e*Gate schema for e*Xchange installation from the CD. These components are shown in light blue and must be added to the base e*Xchange schema by the implementor as needed.

Figure 5 e*Xchange Overview Legend

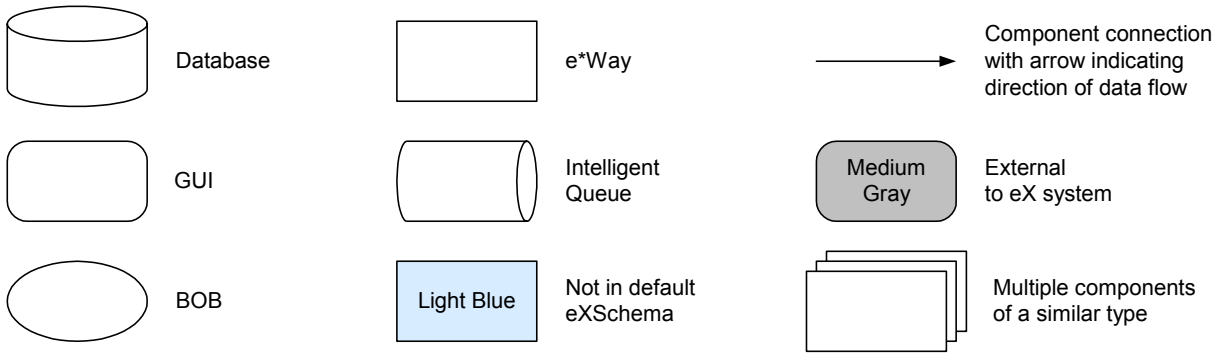
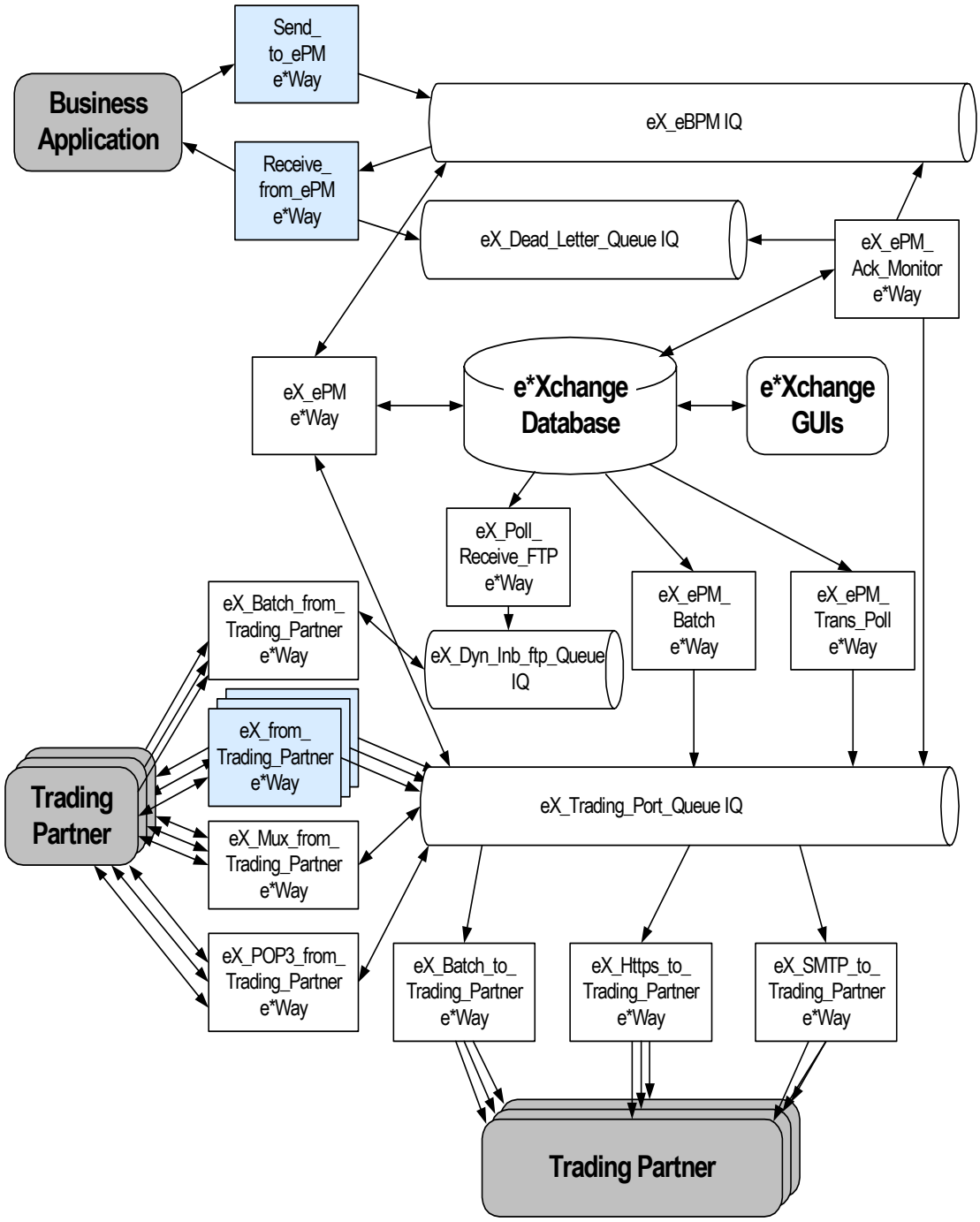


Figure 6 e*Xchange Components



5.3 e*Xchange Partner Manager Components

e*Xchange contains the e*Gate components that handle the sending, receiving, and tracking of messages to and from trading partners. The e*Xchange group is divided into components that interact internally or with the e*Xchange database and those that interact with external systems and trading partners.

e*Xchange Partner Manager—Internal Components

- **eX_ePM e*Way**
- **eX_ePM_Ack_Monitor e*Way**
- **eX_ePM_Batch e*Way**
- **eX_ePM_Trans_Poll e*Way**
- **eX_Poll_Receive_FTP e*Way**

All of these components are provided when the e*Gate schema for e*Xchange is installed. They require only minimal configuration on the part of the user. The components only require that you provide e*Xchange database logon information in their configuration files.

e*Xchange Partner Manager—External Components

The e*Xchange—External component contains e*Ways that send data to and receive data from trading partners and business applications.

- **eX_Batch_from_Trading_Partner e*Way**
- **eX_Batch_to_Trading_Partner e*Way**
- **eX_HTTPS_to_Trading Partner e*Way**
- **eX_Mux_from_Trading_Partner e*Way**
- **eX_POP3_from_Trading_Partner e*Way**
- **eX_SMTP_to_Trading Partner e*Way**
- **Send_to_ePM e*Way**
- **Receive_from_ePM e*Way**
- ***eX_from_Trading_Partner e*Way*** (this is a user-defined component)

5.3.1 eX_ePM e*Way

The e*Xchange e*Way is the main workhorse in the back-end portion of the e*Xchange Partner Manager. The e*Xchange e*Way:

- validates protocol-specific data from trading partners
- writes Event data to the database
- retrieves trading partner profile information from the database

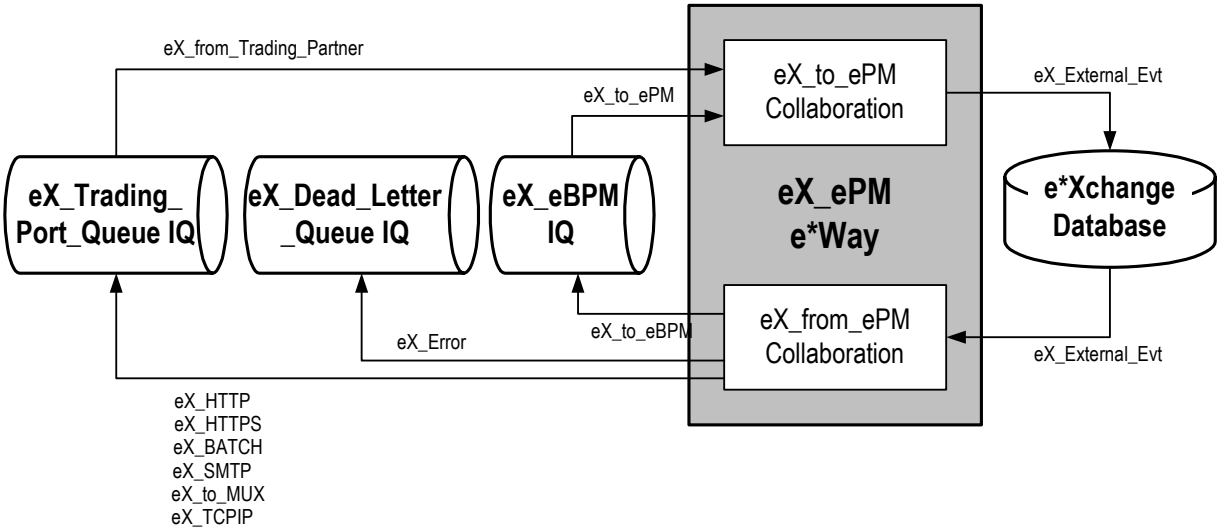
- envelopes the data as required by the destination trading partner

The **eX_ePM** e*Way is a bidirectional e*Way that communicates with both the **eX_eBPM** IQ and the **eX_Trading_Port_Queue** IQ, as well as the e*Xchange database. It forms a bridge between the e*Insight side of the e*Xchange system and the e*Xchange side, receiving Event information both from activity e*Ways and the e*Ways that communicate directly with trading partners.

The e*Xchange engine prepares *outbound* Events coming from e*Insight activity e*Ways to be forwarded to the appropriate trading partner. Conversely, the e*Xchange engine takes *Inbound* Events coming into e*Xchange from trading partners and prepares them to be forwarded to e*Insight.

The following diagram illustrates the **eX_ePM** e*Way.

Figure 7 eX_ePM e*Way Detail



Configuring the e*Xchange Database Connectivity e*Ways

All of the e*Xchange components that communicate with the e*Xchange database are database connectivity e*Ways. You must edit the configuration files for these e*Ways and provide the logon information about the e*Xchange database to which they connect. Table 2 provides information about the required parameters that must be filled in.

Table 2 Parameter Settings for the e*Xchange Database Connectivity e*Ways

Screen	Parameter	Setting
General Settings	(All)	(Default)
Communication Setup	(All)	(Default)
Monk Configuration	(All)	(Default)

Table 2 Parameter Settings for the e*Xchange Database Connectivity e*Ways

Screen	Parameter	Setting
Database Setup	Database Type	The database type. Select one of the following: ▪ ODBC for a SQL Server 7, SQL Server 2000, or UDB ▪ Oracle 8i for either an Oracle 8.1.6 or 8.1.7 database ▪ Sybase for Sybase 11.9 or 12
	Database Name	The **Database Name** is the TNS name the e*Way uses to connect to the e*Xchange database.
	User name	This is the database user name, used by the e*Way to access the e*Xchange database. Any user who is assigned to one of the two Roles defined in the e*Xchange Administrator has access rights to run the e*Gate schema for e*Xchange.
	Encrypted Password	This is the password associated with the database user name the e*Way uses to access the e*Xchange database. The default password used by e*Xchange database creation scripts is **ex_admin**.

eX_to_ePM Collaboration

The **eX_to_ePM** Collaboration is *not* user-configurable.

The **eX_to_ePM** Collaboration retrieves Events to be processed by the e*Xchange engine from either e*Xchange IQ (**eX_eBPM** or **eX_Trading_Port_Queue**), and passes the information to the database script that writes the data from the Events to the e*Xchange database.

Events subscribed to by the **eX_to_ePM** Collaboration must have values populating the e*Xchange-required nodes in the **eX_Standard_Event.ssc** ETD used by these Event Types. These required values include:

- Message ID (a unique identifier for the message), if the direction is outbound and the message does not have a validation check.
- Direction ("I" = inbound, "O" = outbound)
- Partner Name (must correspond exactly to the Logical Name used in the outer envelope of the trading partner profile)

These nodes are explained in more detail in **"Using the ETD in e*Xchange" on page 69** and in the e*Xchange Partner Manager *User's Guide*.

Subscribed Event Types:

- **eX_from_Trading_Partner**—This Event is published by the user-defined e*Ways that handle the inbound Event traffic from trading partners. The Event's data must already be in the XML format required by the e*Xchange side of e*Xchange.
- **eX_to_ePM**—These Events are published either by a **Send_to_ePM** e*Way (or one with similar function) or one of the activity e*Ways associated with the e*Insight business process. The Event's data must already be in the XML format required by the e*Xchange side of e*Xchange.

Published Event Type: eX_External_Evt

This Event carries the data to the database script that writes the information to the e*Xchange database.

eX_from_ePM Collaboration

The **eX_from_ePM** Collaboration is *not* user-configurable.

The **eX_from_ePM** Collaboration retrieves Events prepared by the e*Xchange engine from the database and publishes them to the appropriate IQ. Events forwarded to trading partners are published to the **eX_Trading_Port_Queue** IQ. Events sent to e*Insight are published to the **eX_eBPM** IQ.

Subscribed Event Type: eX_External_Evt

This Event carries information retrieved from the e*Xchange database after the data has been prepared by the e*Xchange engine.

Published Event Types: eX_to_eBPM

- **eX_to_eBPM**—This Event contains information from a trading partner to be sent to the e*Insight side of e*Xchange. This would be the case, for example, if an activity e*Way required an acknowledgment from a trading partner before returning the "Done" Event to the e*Insight engine for that activity.
- **eX_to_Trading_Partner**—This Event contains information that has been prepared by the e*Xchange engine to be sent to a trading partner.
- **eX_to_ePM**—This Event contains information that has been prepared for e*Xchange.
- **eX_HTTPS**—This Event contains the enveloped Event along with destination information.
- **eX_HTTP**—This Event contains the enveloped Event along with destination information.
- **eX_Error**—This Event contains error information.

Important: *You need to create an e*Way or BOB that subscribes to eX_Error, otherwise the eX_ePM e*Way is unable to publish this Event Type.*

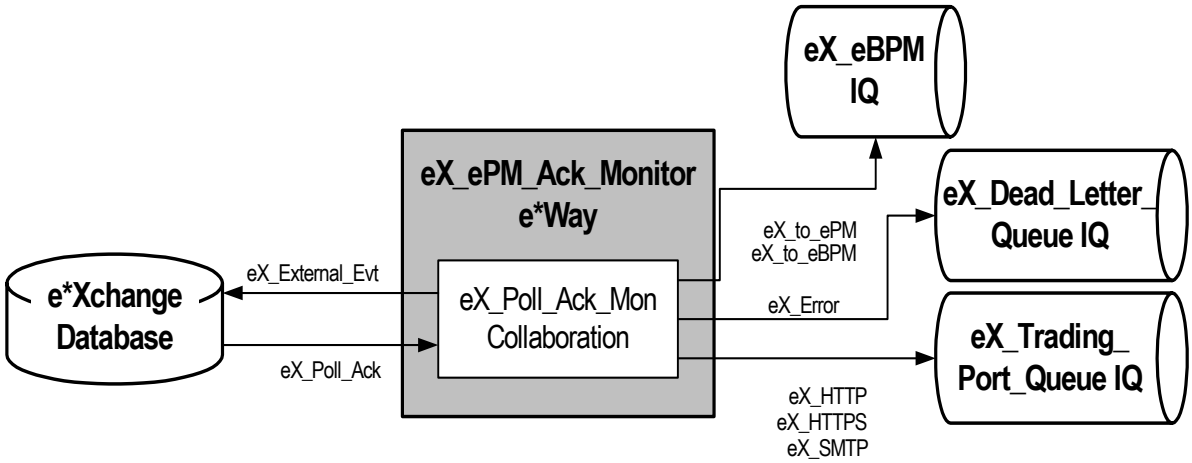
- **eX_to_MUX**—This Event contains the enveloped Event along with destination information.
- **eX_TCPIP**—This Event contains the enveloped Event along with destination information.
- **eX_SMTP**—This Event contains the enveloped Event along with destination information.
- **eX_BATCH**—This Event contains the enveloped Event along with destination information.

5.3.2 eX_ePM_Ack_Monitor e*Way

The **eX_ePM_Ack_Monitor** e*Way is a database connectivity e*Way that monitors the e*Xchange database for Event acknowledgments that are overdue from trading partners.

Figure 8 illustrates the **eX_ePM_Ack_Monitor** e*Way.

Figure 8 eX_ePM_Ack_Monitor e*Way Detail



An acknowledgment is considered overdue if the specified amount of time to wait for an acknowledgment has passed. This "timeout" is configurable and can be set in the e*Xchange GUI. The acknowledgment handling for X12 messages is different than that for RosettaNet messages.

X12 and UN/EDIFACT Acknowledgment Handling

When an acknowledgment is overdue, the **eX_ePM_Ack_Monitor** e*Way determines if the retry limit (the number of times to retry sending the Event) has been reached. If it has not, the e*Way places the Event in a "staging area" within the database to be picked up by the **eX_ePM_Trans_Poll** e*Way and resent to the trading partner. If the retry limit has been reached, the e*Way logs information about the transaction and corresponding error information in the database, and sends an **eX_Error** Event back to the eX_Dead_Letter_Queue IQ with "Hit Re-send Limit" in the eX_Standard_Event.

RosettaNet Acknowledgment Handling

The **eX_ePM_Ack_Monitor** e*Way polls the e*Xchange database for overdue acknowledgments. If an acknowledgment is overdue and the retry limit for that message has not been reached, then the original message is resent to the trading partner. If an acknowledgment is overdue and the retry limit has been exceeded, a failure notification is sent to both the trading partner and the internal application that generated the original RosettaNet message.

Resends

eX_ePM_Ack_Monitor retrieves the original RosettaNet message from the e*Xchange database, increments the retry counter, resigns the message, and then publishes the message to the **eX_Trading_Port_Queue**. The message is then picked up and forwarded to the trading partner.

Failures

eX_ePM_Ack_Monitor e*Way publishes this failure notification using two different Event Types: **eX_to_ePM** and **eX_to_eBPM**. The e*Xchange engine picks up the **eX_to_ePM** failure notification, processes it, and then sends it out to the trading partner via the **eX_Trading_Port_Queue** IQ. The e*Insight engine picks up the **eX_to_eBPM** failure notification and sends it to the internal application.

Configuring the eX_ePM_Ack_Monitor e*Way

The **eX_ePM_Ack_Monitor** e*Way requires only minor changes to the e*Way's configuration file. The implementor must edit this file and provide the information required in the **Database Setup** section as shown in **Table 2 on page 40**.

eX_Poll_Ack_Mon Collaboration

Subscribed Event Type: ex_Poll_Ack

This Event type does not carry any information, since no data is actually extracted from the database by the database script associated with the **eX_ePM_Ack_Monitor** e*Way.

Published Event Types:

- **eX_to_ePM**—This Event Type carries the RosettaNet failure notification sent to the trading partner when the retry message limit has been exceeded.
- **eX_to_eBPM**—This Event Type carries the RosettaNet failure notification sent to the internal application when the retry message limit has been exceeded.
- **eX_to_HTTP**—This Event Type carries the RosettaNet message that is resent when an acknowledgment is overdue.
- **eX_to_HTTPS**—This Event Type carries the RosettaNet message that is resent when an acknowledgment is overdue.
- **eX_to_SMTP**—This Event Type carries the RosettaNet message that is resent when an acknowledgment is overdue.
- **eX_Poll_Ack**—This Event Type is used by the eX_ePM_Ack_Monitor to communicate with the e*Xchange database.
- **eX_Error**—This Event contains error information.

Important: *You must create an e*Way or BOB that subscribes to eX_Error, otherwise the eX_ePM_Ack_Monitor e*Way is unable to publish this Event Type.*

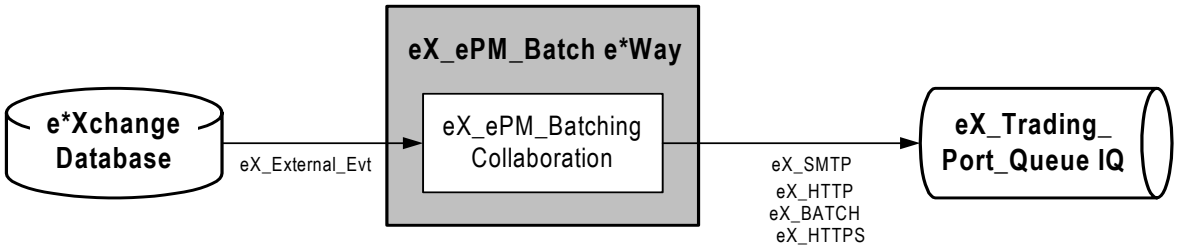
5.3.3 eX_ePM_Batch e*Way

The **eX_ePM_Batch** e*Way polls the e*Xchange database for Events being sent to trading partners using Fast Batch transfer mode or Batch transfer mode (rather than an Interactive) and prepares them to be sent to the appropriate trading partner. When multiple Events need to be sent to the same trading partner, the e*Way bundles these Events together, according to a user-definable bundling scheme, before enveloping them and publishing them to the **eX_Trading_Port_Queue** IQ.

Note: *This e*Way only acts on Events using X12 or UN/EDIFACT enveloping protocols. Events using RosettaNet or BizTalk protocols cannot be transmitted in batch mode.*

The following diagram illustrates the **eX_ePM_Batch** e*Way.

Figure 9 eX_ePM_Batch e*Way Detail



Batch Bundling Schemes

There are 3 types of bundling schemes used by the **eX_ePM_Batch** e*Way:

- **Fast batch**—A set number of Events, all of the same transaction type, are bundled together and sent to a trading partner.
- **Per schedule batch**—At a set time all the Events destined for a single trading partner. The Events can be of differing transaction types.
- **Per interval batch**—The e*Way waits a set interval then bundle all the Events destined for a single trading partner. The Events can be of differing transaction types.

Whether a particular Event uses batch transfer mode and what type of bundling scheme is used for a particular batched Event, is set in the inner envelope definition for that transaction type. See the *e*Xchange Partner Manager User's Guide* for information on setting up the inner envelope to use batch transfer mode.

e*Xchange Event Requirements for Fast Batch

The e*Xchange Event that contains a transaction to be sent to a trading partner using Fast Batch transfer mode, must have the following name and value pairs configured in the standard event:

- FB_UNIQUE_ID — this name and value pair sets the fast batch unique ID. All messages with the same identifier are batched together for processing.
- FB_COUNT — this name and the value pair sets the total number of fast batch records. When e*Xchange receives a fast batch record count equal to or greater than

the value specified in FB_COUNT, or if the fast batch records have exceeded the timeout period, then the **eX_ePM_Batch** e*Way sends the batch records to the trading partner.

Two functions are required for each name and value pair; the first sets the name of the pair (for example, FB_UNIQUE_ID), and the second sets the value.

The example below shows how to configure both name and value pairs for a single event that contains multiple messages in the FB_Feeder.Payload node. The unique ID is the same for every message in the event, and the count is set by counting the number of messages contained in the event (that is, the number of occurrences of the FB_Feeder.Payload node). The method that you use to populate the Value nodes depends on your implementation.

```
// to set the value in the Name node for the unique ID
(copy-strip "FB_UNIQUE_ID"
~output%eX_Event.DS.eX_Event.CT.DSN.DS.TP_EVENT.CT.DSN.DS.TPAttribute
.CT.DSN.DS.NameValuePair[0].CT.DSN.DS.Name.CT.DSN.DS.Data "")

// to set the actual fast batch unique id value in Value node
(uniqueid
~output%eX_Event.DS.eX_Event.CT.DSN.DS.TP_EVENT.CT.DSN.DS.TPAttribute
.CT.DSN.DS.NameValuePair[0].CT.DSN.DS.Value.CT.DSN.DS.Data)

// to set the value in the Name node for the count
(copy-strip "FB_COUNT"
~output%eX_Event.DS.eX_Event.CT.DSN.DS.TP_EVENT.CT.DSN.DS.TPAttribute
.CT.DSN.DS.NameValuePair[1].CT.DSN.DS.Name.CT.DSN.DS.Data "")

// to set the actual total fast batch record count in the Value Node
(copy-strip (count-rep ~input%FB_Feeder.Payload)
~output%eX_Event.DS.eX_Event.CT.DSN.DS.TP_EVENT.CT.DSN.DS.TPAttribute
.CT.DSN.DS.NameValuePair[1].CT.DSN.DS.Value.CT.DSN.DS.Data "")
```

Configuring the eX_ePM_Batch e*Way

The **eX_ePM_Batch** e*Way requires only minor changes to the e*Way's configuration file. The implementor must edit this file and provide the information required in the **Database Setup** section as shown in **Table 2 on page 40**.

You can also specify the protocol type of the messages to be batched, if required. The **eBusiness Type** is specified in the **eBusiness Type Settings** section. The available parameters are:

- ALL—All protocol types are retrieved. This is the default setting.
- NCPDP—For future use.
- UN/EDIFACT—Only UN/EDIFACT messages are retrieved.
- X12—Only X12 messages are retrieved.

Scaling of eX_ePM _Batch e*Way

You can create multiple eX_ePM_Batch e*Ways to improve performance. To use multiple e*Ways you need to modify the configuration. For example, if you want three eX_ePM_Batch e*Ways you need to create and configure them as follows:

- Copy the eX_ePM_Batch e*Way.
- Create separate configuration files for each eX_ePM_Batch e*Way.
  - Open the configuration file for each new e*Way and save with a different name. Ensure that the e*Way refers to this configuration file, not the original one.
- Modify the configuration files, as shown in the Table 3.

Table 3 Configuration File Parameters

e*Way	Number of Batch eWays Parameter	Batch eWay Instance Number Parameter
eX_ePM_Batch	3	1
eX_ePM_Batch_0	3	2
eX_ePM_Batch_1	3	3

The functionality used by each e*Way is a modulo. Therefore, if three e*Ways are used, any one e*Way picks up every third record.

Important: *Do not use this if message sequencing is desired.*

eX_ePM_Batching Collaboration

This Collaboration is not user configurable.

Subscribed Event Type: eX_External_Evt

The **eX_ePM_Batch** e*Way uses this Event Type to communicate with the e*Xchange database.

Published Event Type: eX_To_Trading_Partner, eX_SMTP, eX_HTTP, eX_BATCH, eX_HTTPS

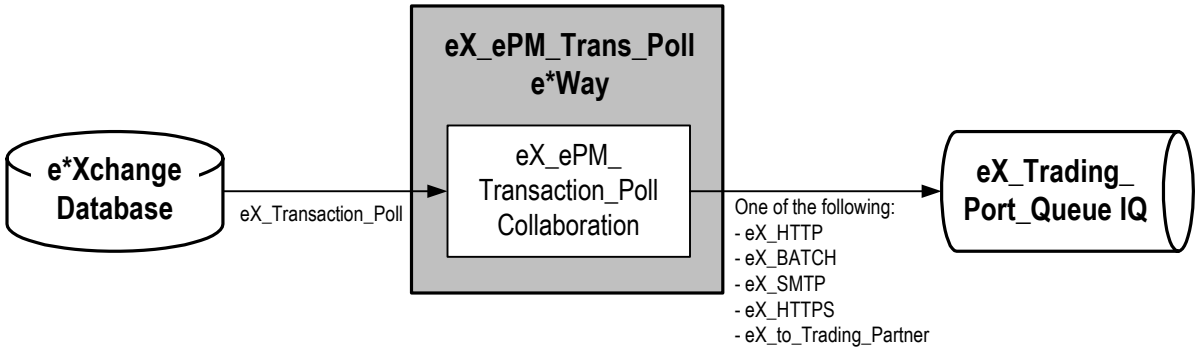
This Event carries the multiple X12 transactions that have been bundled together and enveloped by the **eX_ePM_Batch** e*Way.

5.3.4 eX_ePM_Trans_Poll e*Way

The **eX_ePM_Trans_Poll** e*Way monitors a "staging area" in the database for outbound Events pending interactive transfer. It uses a database access script (**tran_poll.dsc**) called by the **Exchange Data With External Function** parameter in the e*Way's configuration file to retrieve these Events from the e*Xchange database. The **eX_Transaction_Poll** Collaboration then publishes the Events to the **eX_Trading_Port_Queue** IQ under **eX_HTTP**, **eX_BATCH**, **eX_SMTP**, **eX_HTTPS**, or **eX_to_Trading_Partner** Event Type.

Figure 10 illustrates the **eX_ePM_Trans_Poll** e*Way.

Figure 10 eX_ePM_Trans_Poll e*Way



Configuring the eX_ePM_Trans_Poll e*Way

The **eX_ePM_Trans_Poll** e*Way requires only minor changes to the e*Way's configuration file. The implementor must edit this file and provide the information required in the **Database Setup** section as shown in **Table 2 on page 40**.

eX_ePM_Transaction_Poll Collaboration

This is a Collaboration that does two things:

1. Changes the name of the Event from **eX_Transaction_Poll** to one of the following:
   - **eX_BATCH**
   - **eX_HTTPS**
   - **eX_HTTP**
   - **eX_SMTP**
   - **eX_to_Trading_Partner**
2. Publishes it to the **eX_Trading_Port_Queue** IQ.

Subscribed Event Type: eX_Transaction_Poll

This Event Type is used by the **eX_ePM_Trans_Poll** e*Way to retrieve the enveloped Events from the e*Xchange database.

Published Event Types: eX_Batch, eX_HTTP, eX_HTTPS, eX_SMTP, eX_to_Trading_Partner

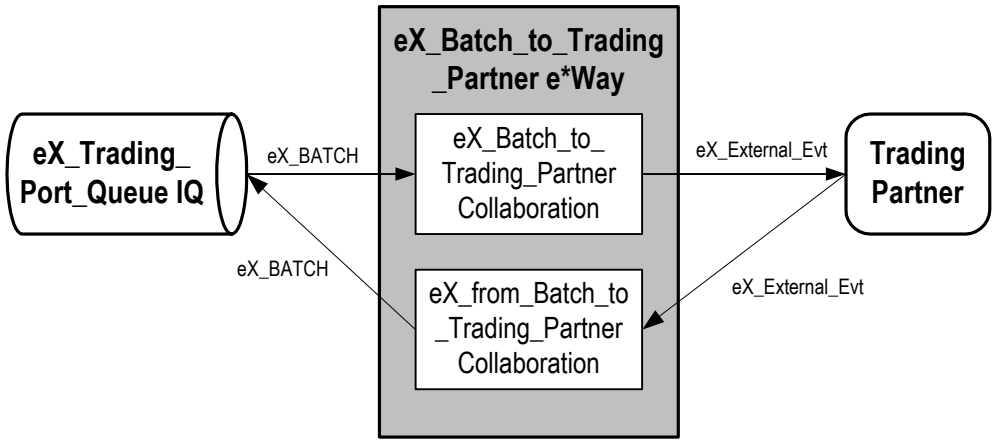
This Event Type carries enveloped Events intended for a trading partner.

5.3.5 eX_Batch_to_Trading_Partner e*Way

The **eX_Batch_to_Trading_Partner** e*Way sends enveloped eBusiness messages designated for batch transmission to trading partners using FTP.

Figure 12 illustrates the **eX_Batch_to_Trading_Partner** e*Way.

Figure 11 eX_Batch_to_Trading_Partner e*Way Detail



The destination file location for each Event is carried as part of the e*Xchange Event data passed to the **eX_Batch_to_Trading_Partner** e*Way. The file location is maintained in the e*Xchange database and applied to the Event at the same time that the Event is enveloped for a specific trading partner. When the **eX_Batch_to_Trading_Partner** e*Way receives an Event, they send the data to the file location specified within the Event itself.

Configuring the eX_Batch_to_Trading_Partner e*Way

No configuration is required.

eX_Batch_to_Trading_Partner Collaboration

This is a Pass Through Collaboration used to publish the Event outside of e*Gate. The communication portion of the **eX_Batch_to_Trading_Partner** e*Way then takes the Event and sends via FTP to the appropriate trading partner.

Subscribed Event Type: eX_BATCH

This Event carries the enveloped Event along with the destination information (URL) used by the **eX_Batch_to_Trading_Partner** e*Way for transmission to the trading partners.

Published: eX_External_Evt

This Event carries the eBusiness message to the communications half of the e*Way where it is forwarded to the trading partner.

eX_from_Batch_to_Trading_Partner Collaboration

This Collaboration is used when the e*Way failed to connect to an external host. The user has three options:

- Resend—Re-publishes the message to the eX_Batch_to_Trading_Partner e*Way through the queue.

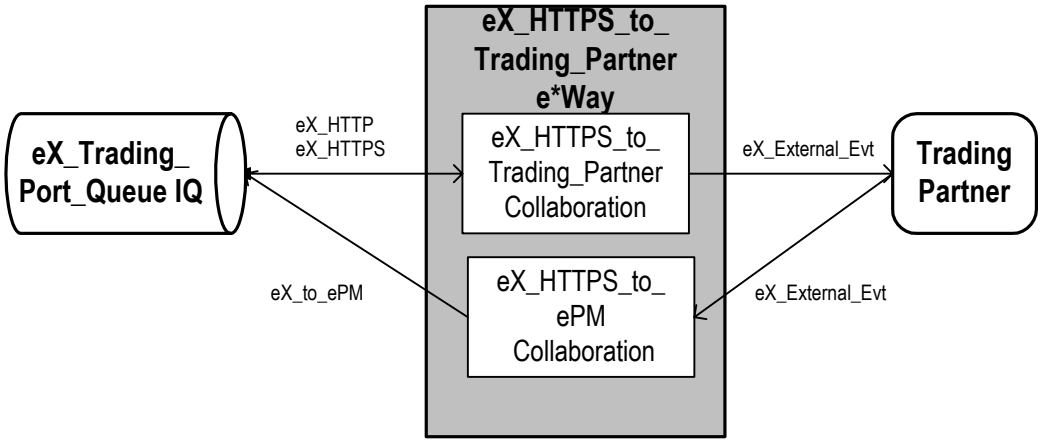
- Rollback—Retries the message within the eX_Batch_to_Trading_Partner e*Way the number of times specified in the configuration for the eX_Batch_to_Trading_Partner e*Way, in the General settings, **Max Resends Per Message**. If the resend count is reached without success, the e*Way shuts down.
- Skip—Skips the message. Resends the number of times specified in the trading partner profile (**Return Inner Envelope** tab) in the user interface.

5.3.6 eX_Https_to_Trading_Partner e*Way

The **eX_Https_to_Trading_Partner** e*Way is an HTTPS e*Way that sends eBusiness messages enveloped by the e*Xchange Partner Manager to trading partners over a secure (HTTPS) or unsecure (HTTP) communication link. The secure link encrypts the data, the unsecure link does not.

Figure 12 illustrates the **eX_Https_to_Trading_Partner** e*Way.

Figure 12 eX_Https_to_Trading_Partner e*Way Detail



The destination URL for each Event is carried as part of the e*Xchange Event passed to the **eX_Https_to_Trading_Partner** e*Way. The URL is maintained in the trading partner database and applied to the Event at the same time that the Event is enveloped for a specific trading partner. When the **eX_Https_to_Trading_Partner** e*Way receives an Event, it sends the data to the URL specified within the Event itself.

Whether or not the Event is sent over a secure channel (encrypted) using HTTPS or over an unsecure channel (not encrypted) using HTTP protocol is also determined by the presence of "https" in the URL string. For example, a URL sting such as **http://tradingpartner.com** indicates the use of the unsecure mode, whereas a string such as **https://tradingpartner.com** indicates the use of the secure mode.

Note: *When using the secure mode, the eSecurityManager components (both the GUI and the back end) must be installed and the appropriate security key fields populated in the trading partner profile. See the* ***e*Xchange Partner Manager User's Guide*** *for information on setting up e*Xchange to use the eSM features of e*Xchange.*

Configuring the eX_Https_to_Trading_Partner e*Way

No configuration is required if the HTTP protocol is used.

If the HTTPS protocol is used, you must ensure that the configuration for the TrustStore is correct. The trust store file contains information about Web servers that accept messages from your system. The file **<egate>\client\pkicerts\truststore\trustcacertsjks** is created when e*Gate is installed and this default file can be used with e*Xchange. However, if you have not installed e*Gate on your C drive, or you want to use a different file or location, you need to update the configuration file for the **eX_ePM_Https_eWay_Con** e*Way connection. The file name for the trust store file is defined in the SSL section, TrustStore parameter.

For more information about the TrustStore, see *HTTPS e*Way Intelligent Adapter User's Guide*.

eX_Https_to_Trading_Partner Collaboration

This is a Java Collaboration used to publish the Event outside of e*Gate. The communication portion of the **eX_Https_to_Trading_Partner** e*Way then takes the Event and sends it to the appropriate trading partner.

Subscribed Event Type: eX_HTTPS

This Event carries the data and destination information (URL) used by the **eX_Https_to_Trading_Partner** e*Way for sending the message to the trading partner.

Published: eX_External_Evt

This Event carries the eBusiness message to the trading partner.

eX_Https_to_ePM Collaboration

This is a Pass Through Collaboration used to receive a response from the trading partner.

Subscribed Event Type: eX_External_Evt

This Event carries the response from the trading partner.

Published: eX_to_ePM

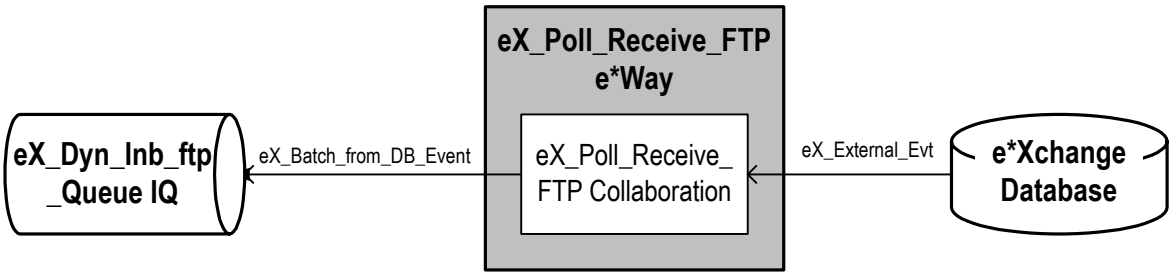
This Event forwards the response to e*Xchange.

5.3.7 eX_Poll_Receive_FTP e*Way

The **eX_Poll_Receive_FTP** e*Way polls the e*Xchange database for information on trading partners that have data to be retrieved via FTP. This information is provided in the Trading Partner profile. The information about each Trading Partner is then passed to the **eX_Batch_From_Trading_Partner** e*Way.

Figure 14 illustrates the **eX_Poll_Receive_FTP** e*Way.

Figure 13 eX_Poll_Receive_FTP e*Way Detail



The Event is sent to eX_Batch_from_Trading_Partner.

The eX_Dyn_Inb_ftp_Queue IQ is configured to use a Subscriber Pool. This ensures that when multiple eX_Batch_from_Trading_Partner e*Ways are used, the information about each Trading Partner is passed to every e*Way in turn.

Configuring the eX_Poll_Receive_FTP e*Way

The **eX_ePM_Trans_Poll** e*Way requires only minor changes to the e*Way's configuration file. The implementor must edit this file and provide the information required in the **Database Setup** section as shown in **Table 2 on page 40**.

eX_Poll_Receive_FTP Collaboration

This is a Pass Through Collaboration used to take the Event received from the eX_Poll_Receive_FTP e*Way and send it to the eX_Trading_Port_Queue.

Subscribed Event Type: eX_External

This Event carries information about the trading partners that have files to be retrieved by FTP.

Published: eX_Batch_from_DB_Event

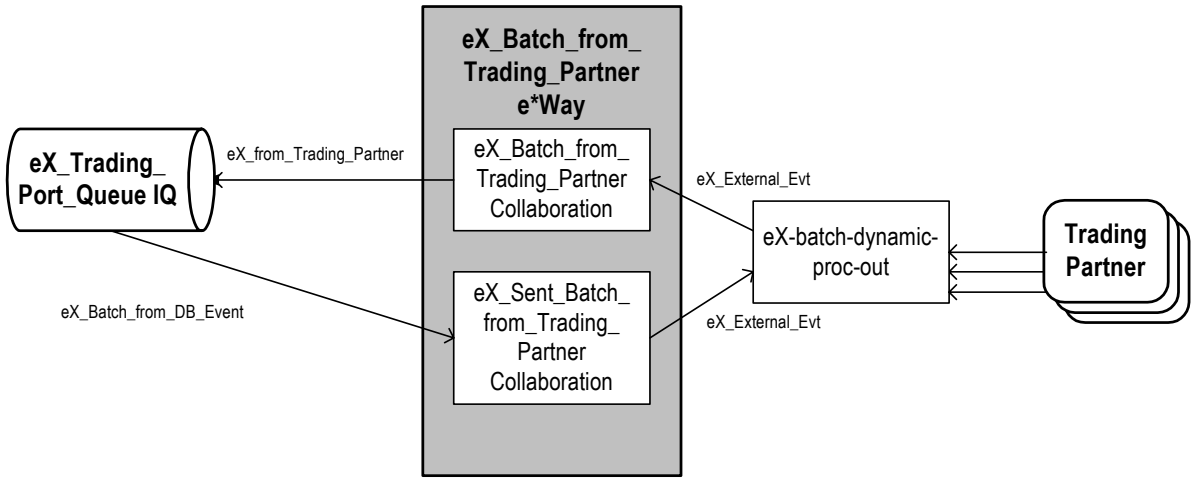
This Event carries the trading partner configuration information to the eX_Trading_Port_Queue. It is then retrieved by the eX_Batch_from_Trading_Partner e*Way.

5.3.8 eX_Batch_from_Trading_Partner e*Way

The **eX_Batch_from_Trading_Partner** e*Way sends enveloped eBusiness messages to e*Xchange.

Figure 14 illustrates the **eX_Batch_from_Trading_Partner** e*Way.

Figure 14 eX_Batch_from_Trading_Partner e*Way Detail



The Event is sent to e*Xchange.

Configuring the eX_Batch_from_Trading_Partner e*Way

The eX_Batch_from_Trading_Partner e*Way supports minor changes regarding the naming and location of files after the transfer has successfully completed and the messages have been published to the eX_Trading_Port_Queue IQ. By default, the remote files are renamed and local files are deleted.

The parameters that determine what happens to these files are set in the **Subscribe to External** section.

Table 4 Subscribe to External Parameters

Parameter	Setting	Description
Remote Command after Transfer	archive	The file is moved from the <path> defined in the trading partner profile to <path>\ARCHIVE Important: This directory must be created manually. Note: Unix is case-sensitive.
	delete	The file is deleted.
	none	This is not supported with this e*Way.
	rename	The file is renamed to <filename>.backup. The location remains the same.

eX_Sent_Batch_from_Trading_Partner Collaboration

This is a Pass Through Collaboration used to take the Event received from the eX_Poll_Receive_FTP e*Way and send it to the external. The communication portion of the **eX_Sent_Batch_from_Trading_Partner** e*Way then takes the Event and uses the trading partner configuration information to issue the relevant FTP command to the appropriate trading partner.

Subscribed Event Type: eX_Batch_from_DB_Event

This Event carries information about the trading partners that have files to be retrieved by FTP.

Published: eX_External_Evt

This Event carries the trading partner configuration information to the communications half of the e*Way where it is used to format the FTP command sent to the trading partner.

eX_Batch_from_Trading_Partner Collaboration

This is a Monk Collaboration used to take the Event received from the Trading Partner and send to e*Gate.

Subscribed Event Type: eX_External_evt

This Event carries the Event retrieved from the trading partners.

Published: eX_from_Trading_Partner

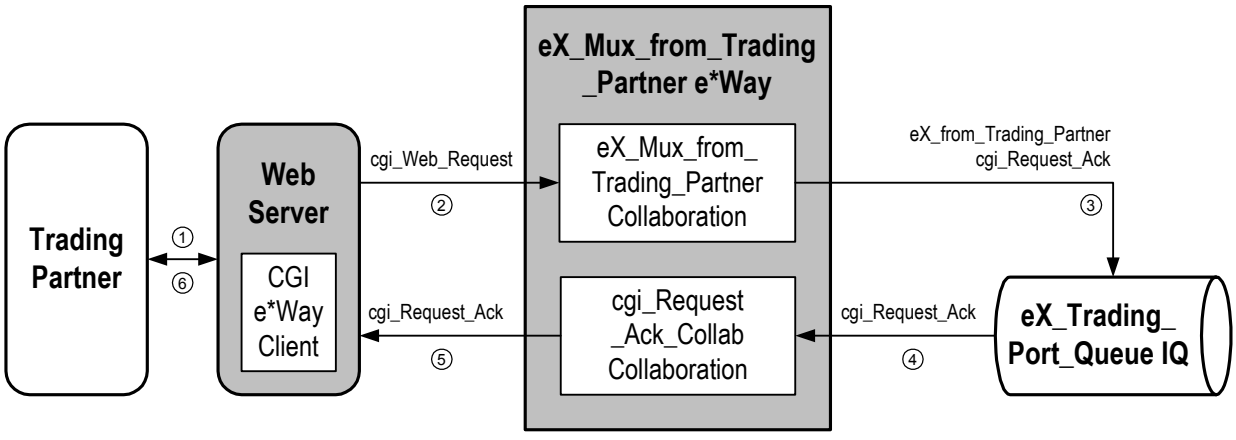
This Event carries the Event retrieved from the trading partners and forwards it to e*Gate.

5.3.9 eX_Mux_from_Trading_Partner e*Way

The **eX_Mux_from_Trading_Partner** e*Way is a CGI Web Server e*Way that communicates with a CGI e*Way client running on a Web server. This allows information sent to the Web server by a trading partner to be picked up and processed by e*Xchange.

Figure 12 illustrates the **eX_Mux_from_Trading_Partner** e*Way.

Figure 15 eX_Mux_from_Trading_Partner e*Way Detail



How the CGI Web Server e*Way Works

① The trading partner posts eBusiness data to the Web server.

② The CGI e*Way client program adds tracking information and forwards the eBusiness message to the **eX_Mux_from_Trading_Partner** e*Way.

③ The **eX_Mux_from_Trading_Partner** Collaboration extracts the trading partner name from the tracking information provided by the CGI e*Way client and publishes the eBusiness message data as a standard e*Xchange Event (**eX_from_Trading_Partner**) to the **eX_Trading_Port_Queue** IQ. In addition, the Collaboration creates an acknowledgment Event (**cgi_Request_Ack**) with the same Mux tracking number as the original post and publishes it to the **eX_Trading_Port_Queue** IQ.

④ The **cgi_Request_Ack_Collab** Collaboration picks up the acknowledgment Event and forwards it to the CGI client on the Web server.

⑤ The CGI client matches up the acknowledgment with the original post.

⑥ The trading partner receives a response from the Web server indicating that the data has been sent successfully to e*Xchange.

Configuring the eX_Mux_from_Trading_Partner e*Way

Configuring the **eX_Mux_from_Trading_Partner** e*Way is a two step process.

1. Set up the cgi client on the Web server.
2. Specify the port over which the CGI e*Way server listens for connections from the CGI e*Way client.

Setting up the CGI client

The following is a brief discussion of setting up the CGI e*Way client. For more information about setting up client software used by the CGI e*Way, see the *CGI Web Server e*Way User's Guide*.

1. Install the cgi client files on the Web server.

   The files **stc_common.dll**, **stc_ewipmpclnt.dll**, and **stcewcgi.exe**, provided as part of the CGI e*Way add-on installation, should be placed at the document root location on the Web server host machine.

2. Modify the configuration file for the Web server.

   You must add the following environment variables to the virtual host setup for the port over which the trading partners send data to the Web server. These variables are used by the CGI e*Way Client.

   - **STC_EW_SERVER_NAME**—The host name or IP address of the machine running the CGI Web server e*Way. If the parameter is not supplied, the default is localhost.
   - **STC_EW_SERVER_PORT**—The port number on which the CGI Web server e*Way is listening. If the parameter is not supplied, or a value of zero (0) is supplied, the default port number is used.
   - **STC_EW_SECONDS_TO_EXPIRE**—The number of seconds the message is active in the e*Gate system. host name or IP address of the machine running the

CGI Web server e*Way. If the parameter is not supplied, the default is zero (0), indicating that the message remains active indefinitely.

- **STC_EW_MILLISECONDS_TO_WAIT**—The number of milliseconds the CGI e*Way Client waits for the response from the CGI e*Way Server. The CGI e*Way Client displays an error message if the CGI e*Way Server fails to respond in the given time period. If the parameter is not supplied, a value of ten thousand (10,000) is the default.
- **DocumentRoot**—The location on the Web server taken as the starting point for relative paths to files for this virtual host setup. For example, if the **DocumentRoot** for port **690** is **opt/web/htdocs/exchange** and a request is made to the following URL **http://www.stc.com:690/4.1.2/stcewcgi.exe**, the file is found at the location **opt/web/htdocs/exchange/4.1.2/stcewcgi.exe** on the Web server.

The virtual host setup containing all the above environment variables can be kept in a separate file and called using the **include** command from within the Web server's configuration file.

Specifying the Request Reply IP Port

You must edit the **eX_Mux_from_Trading_Partner** e*Way's configuration file and enter the appropriate port number. This port should be the same as the port specified by the **STC_EW_SERVER_PORT** environment variable used by the CGI e*Way client.

eX_Mux_from_Trading_Partner Collaboration

This Collaboration takes the information in the **cgi_Web_Request** Event and uses it to construct a standard e*Xchange Event (**eX_from_Trading_Partner**) to be processed by e*Xchange. In addition, it creates an acknowledgment Event (**cgi_Request_Ack**) to be sent back to the trading partner telling it that the information has been successfully placed into e*Xchange for processing.

The Collaboration creates the trading partner name used by e*Xchange by concatenating the values of **HTTP_HOST** and **REQUEST_URI** as follows

https://<HTTP_HOST><REQUEST_URI>

Subscribed Event Type: cgi_Web_Request

This is the Event is provided by the CGI client. It contains the 24 byte Mux header used to match up the request with the reply, URL tracking information, and the data contained in the Web server post.

Published Event Types:

- **eX_from_Trading_Partner**—This is the e*Xchange Event created from the Web server post and contains the post information along with the required e*Xchange tracking information.
- **cgi_Request_Ack**—This Event contains the 24 byte Mux header along with the acknowledgment text "Post Accepted".

cgi_Request_Ack_Collab Collaboration

This is a Pass Through Collaboration that picks up the **cgi_Request_Ack** Event and publishes it outside of e*Gate where it is picked up by the CGI e*Way client running on the Web server.

Subscribed Event Type: cgi_Request_Ack

This Event contains the 24 byte Mux header along with the acknowledgment text "Post Accepted".

Published Event Types: cgi_Request_Ack

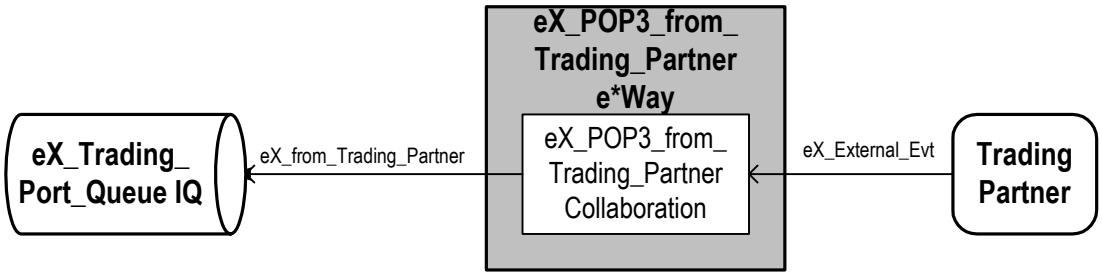
This Event contains the 24 byte Mux header along with the acknowledgment text "Post Accepted".

5.3.10 eX_POP3_from_Trading_Partner e*Way

The **eX_POP3_from_Trading_Partner** e*Way sends enveloped eBusiness messages to e*Xchange.

Figure 12 illustrates the **eX_POP3_from_Trading_Partner** e*Way.

Figure 16 eX_POP3_from_Trading_Partner e*Way Detail



The Event is sent to e*Xchange.

Configuring the eX_POP3_from_Trading_Partner e*Way

No configuration is required.

eX_POP3_from_Trading_Partner Collaboration

This is a Pass Through Collaboration used to send the Event to e*Gate. The communication portion of the **eX_POP3_from_Trading_Partner** e*Way receives the Event via POP 3 and sends into e*Gate.

Subscribed Event Type: eX_External_Evt

This Event carries the enveloped Event along with the destination information (URL) used by the **eX_POP3_from_Trading_Partner** e*Way for transmission to the trading partners.

Published: eX_External_Evt

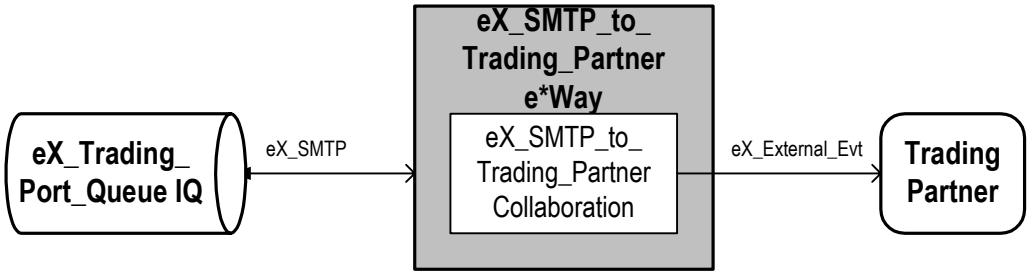
This Event carries the eBusiness message to the communications half of the e*Way where it is forwarded to the trading partner.

5.3.11 eX_SMTP_to_Trading_Partner e*Way

The **eX_SMTP_to_Trading_Partner** e*Way sends enveloped eBusiness messages designated for batch transmission to trading partners using email.

Figure 12 illustrates the **eX_SMTP_to_Trading_Partner** e*Way.

Figure 17 eX_SMTP_to_Trading_Partner e*Way Detail



The destination information for each Event is carried as part of the e*Xchange Event data passed to the **eX_SMTP_to_Trading_Partner** e*Way. The information is maintained in the e*Xchange database and applied to the Event at the same time that the Event is enveloped for a specific trading partner.

Configuring the eX_SMTP_to_Trading_Partner e*Way

No configuration is required.

eX_SMTP_to_Trading_Partner Collaboration

This is a Pass Through Collaboration used to publish the Event outside of e*Gate. The communication portion of the **eX_SMTP_to_Trading_Partner** e*Way then takes the Event and sends via FTP to the appropriate trading partner.

Subscribed Event Type: eX_BATCH

This Event carries the enveloped Event along with the destination information (URL) used by the **eX_SMTP_to_Trading_Partner** e*Way for transmission to the trading partners.

Published: eX_External_Evt

This Event carries the eBusiness message to the communications half of the e*Way where it is forwarded to the trading partner.

5.3.12 Send_to_ePM e*Way

In an e*Xchange implementation, e*Xchange requires one or more e*Gate components to feed data into and take data from e*Xchange. Typically, these components are e*Ways that connect to the business application providing the eBusiness data that is sent to trading partners via e*Xchange.

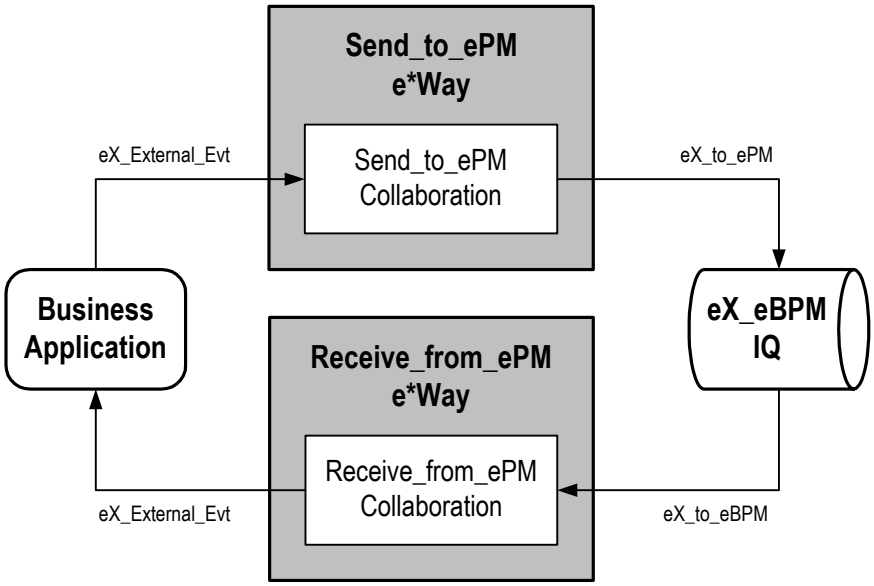
These e*Ways are user defined, and the type of e*Way used depends on the source for the eBusiness data. For example, if the source is a business application the e*Gate e*Way that connects to that business application (such as Siebel) is used. If the source of the data is already available in e*Gate, you can use a BOB instead of an e*Way.

The eXSchema installed as the e*Xchange backend includes two placeholder e*Ways, **Send_to_ePM** and **Receive_from_ePM** that can be used as the starting point for this functionality.

These e*Ways have all the Collaborations and routing defined, but the e*Way executable must be selected and the configuration file must be created.

The following diagram illustrates these e*Ways.

Figure 18 Send_to_ePM and Receive_from_ePM e*Ways Detail



The **Send_to_ePM** e*Way initiates the process of sending data to e*Xchange. This e*Way is user-defined, and its type is dependent upon the communication protocol and/or application-specific requirements of the customer. The Collaboration within the e*Way is also user-defined. It converts the external, proprietary data format supplied by the application to the internal XML format required by e*Xchange.

Configuring the Send_to_ePM e*Way

The configuration details for this eWay depend on the type of external system to which it connects. In general you must

- Choose the e*Way executable
- Create the configuration file
- Edit the Collaboration Rules Script used by the **Send_to_ePM** Collaboration

See the *e*Way User's Guide* for the e*Way type you wish to use for this e*Way for more detailed configuration information.

Send_to_ePM Collaboration

The **Send_to_ePM** Collaboration must use the data it receives from the business application to create the Event sent to the e*Xchange engine. Specifically it must do the following:

- convert the data to an XML-compatible format and put it in the e*Xchange payload node of the e*Xchange standard Event
- populate the e*Xchange-required tracking nodes in the e*Xchange standard Event

Subscribed Event Type: eX_External_Evt

This Event Type corresponds to the inbound data provided by the external application.

Published Event Type: eX_to_PM

This Event carries the e*Xchange formatted data to the e*Xchange engine.

Converting Business Application Data to e*Xchange Format

You must copy the eBusiness message data to the payload node (**TP_EVENT.CT.DSN.DS.Payload.CT.DSN.Data)** of the **eX_Standard_Event.ssc** ETD for the Event that is sent to e*Xchange.

If this data contains characters that conflict with the XML structure of the e*Xchange standard Event, the data must be converted to base 64 encoding prior to being copied.

You can convert the data in the **START_BP** Collaboration by using the Monk function **raw->base64**.

Note: *Make sure that the **stc_monkutils.dll** that contains the function **raw->base64** is loaded before using **raw->base64** in a Collaboration Rules Script. For example, you may use the command: **load-extension "stc_monkutils.dll"** in the CRS itself or you may put path to a file that loads in the **initialization file** box in the Collaboration Rule that uses the CRS. See* **"Convert the Event to Base 64 Encoding" on page 73** *for an example of how to do the later.*

e*Xchange-required Tracking Nodes

The following three nodes in the **eX_Standard_Event.ssc** ETD must be populated in the Event sent to the e*Xchange engine:

- **TP_EVENT.CT.DSN.DS.MessageID.CT.DSN.Data** node must be filled with a unique ID for the Event, if the message is outbound with no validation checking.

- **TP_EVENT.CT.DSN.DS.Direction.CT.DSN.Data** node must be filled with the string "I" designating it an inbound to e*Xchange Event.
- **TP_EVENT.CT.DSN.DS.PartnerName.CT.DSN.Data** node must be filled with the logical name of the trading partner to which the EDI message is being sent. This name corresponds to the case sensitive text found in the **Logical Name** box on the **General** tab of the outer envelope for the trading partner in the e*Xchange GUI.

5.3.13 Receive_from_ePM e*Way

The **Receive_from_ePM** e*Way takes eBusiness data received from trading partners and processed by e*Xchange and formats it for use by the destination business application that requires it. This e*Way is user-defined, and the type chosen depends upon the communication protocol and/or application-specific requirements of the business application or other external to e*Gate destination to which it connects. The Collaboration within the e*Way is also user-defined. It must convert the eBusiness data from the e*Xchange XML format back into the format required by the destination system.

Configuring the Receive_from_ePM e*Way

The **Receive_from_ePM** e*Way must be configured by the user. You must select the type of e*Way create the configuration file, and then edit the Collaboration Rules Script used by the **Receive_from_ePM** Collaboration.

See the *e*Way User's Guide* for the e*Way type you wish to use as the **Receive_from_ePM** e*Way for more detailed configuration information.

Receive_from_ePM Collaboration

This Collaboration takes data from a trading partner, processed by e*Xchange, and formats it for use by the business application to which the **Receive_from_ePM** e*Way connects. You must edit the placeholder Collaboration (**Receive_from_ePM**) provided with the default **Receive_from_ePM** e*Way so that it performs this translation.

Subscribed: eX_to_eBPM

This Event carries the de-enveloped trading partner data to the eX_eBPM IQ where it is picked up by the Receive_from_ePM Collaboration.

Published: eX_External_Evt

This Event carries the properly formatted data received from a trading partner to the business application.

5.3.14 eX_from_Trading_Partner e*Way

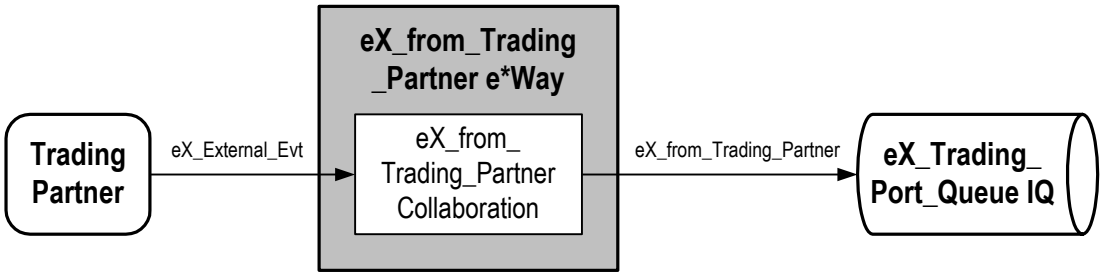
The **eX_from_Trading_Partner** e*Way is not included as part of the default eXSchema installation. Because of the wide variety of methods used by trading partners to transmit eBusiness data, it is impossible to predict in advance the type of e*Way needed (Batch, HTTP, TCP/IP, etc.) or the type of data translation required to bring the data into e*Xchange for a particular trading partner. Therefore, this e*Way must be added to

the e*Xchange schema and configured for a trading partner on a per entity basis. Typically, there are several e*Ways of this type that need to be added to a large e*Xchange implementation.

The following discussion focuses on the requirements for the e*Xchange Event that this e*Way type must create and send to e*Xchange. The **eX_from_Trading_Partner** e*Way must ensure that the data coming into the e*Xchange system is in the proper XML format, and that the nodes in the e*Xchange standard Event, required for processing by the e*Xchange and the e*Insight, are populated.

The following diagram illustrates an example of an inbound e*Way.

Figure 19 eX_from_Trading_Partner e*Way Detail



Configuring the eX_from_Trading_Partner e*Way

The configuration for the **eX_from_Trading_Partner** e*Way depends on the type of e*Way it is, Batch, HTTP, TCP/IP, etc. and the amount of required to get the data into standard e*Xchange format. See the *e*Way User's Guide* for the e*Way type selected for more detailed information on configuration.

eX_from_Trading_Partner Collaboration

This user-defined Collaboration must put the data coming in from the external trading partner into the e*Xchange standard format. In addition it must ensure that the required tracking information is included in the Event sent to the e*Xchange engine. Specifically, this includes:

- The Message ID (a unique identifier for the message)

  If this is an acknowledgment to a previously sent out eBusiness message, it should use the same identifier as the original Event.

- The Direction of the Event ("I" = inbound to e*Xchange)
- The Partner Name

  This must correspond exactly to the Logical Name used in the outer envelope of the trading partner profile.

- Message Type ("RAW", "PROCESSED", or "ENCRYPTED")

These nodes are explained in more detail in and in the e*Xchange Partner Manager *User's Guide*. Also, see **Populate the Required e*Xchange Nodes** on page 74 for an

example of a Monk Collaboration that does this, or see **Populate the Required e*Xchange Nodes** on page 81 for an example of a Java Collaboration that does this.

Subscribed Event Type: eX_External_Evt

This Event carries the data that comes from the trading partner. Depending on the amount of pre-processing the data has received, this Event may or may not be in the XML format, with the required nodes populated as needed by the e*Xchange system.

Published Event Type: eX_from_Trading_Partner

This Event carries e*Xchange formatted data to the e*Xchange engine. All inbound e*Xchange e*Ways publish data from the trading partner using this Event Type.

Chapter 6

Using the Monk e*Xchange ETD

e*Xchange uses a single Event Type Definition (ETD) named **eX_Standard_Event.ssc** to define Events as they move from one component to another in the e*Xchange system. The ETD is an XML DTD in SeeBeyond's proprietary messaging format. For a description of the XML DTD see **Appendix A**.

All data going into and coming out of the e*Xchange components is parsed according to the e*Xchange ETD. Understanding this ETD is the key to creating the Collaboration Rules scripts necessary to process the data according to the rules determined by the business process.

Note: *The BP_EVENT location in the **eX_Standard_Event.ssc** contains information for e*Insight Business Process Manager. You can ignore this section if your implementation does not use e*Insight Business Process Manager. For more information on the BP_EVENT location, refer to the e*Insight Business Process Manager Implementation Guide.*

6.1 ETD Structure

The first step in using the ETD is understanding the structure of the nodes in the context of the XML message being created. Each level is structured in the same way.

The ETD contains a number of nodes that do not explicitly correlate to the XML DTD but are required by the Monk engine to parse the XML data correctly. Table 5 lists these *facilitator* nodes.

Table 5 Facilitator Nodes in the ETD

Name	Description
CT	A container node for an XML element. This node allows the short and long forms of XML tags to coexist in the structure.
DSN	Identifies a data section within an XML element. This is the long form of the XML tag.
DS	Identifies a data set within an XML element. The sub-elements within a data set can occur in any order.
Empty	The short form of the corresponding DSN node XML tag.
CM	XML comment.

Table 5 Facilitator Nodes in the ETD

Name	Description
Data	Holds the data for the element.
AS	Identifies an XML attribute set within an XML element.
EQ	The equals sign ("=") within an XML attribute.
Value	Holds the value for the XML attribute.

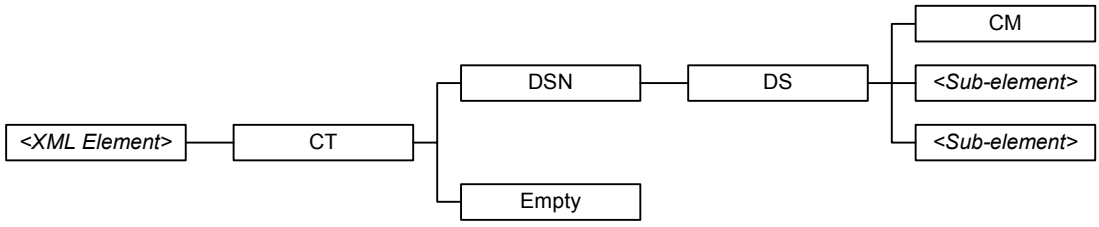
The facilitator nodes always occur in a set order and define the structure of the XML message. In the e*Xchange ETD, the facilitator nodes define three types of branches:

- XML element with sub-elements
- XML element without sub-elements
- XML attribute

6.1.1 XML Element with Sub-elements

The following diagram illustrates the ETD structure for an XML element that has sub-elements.

Figure 20 XML Element with Sub-elements



Each XML element contains one child node, **CT**. **CT** identifies the parent node as an XML element. The **CT** node contains two child nodes: **DSN** and **Empty**. **DSN** is the long form of the XML tag (</*tag*>) and **Empty** is the short form (</>).

The **DSN** and **DS** nodes always occur as parent-child pairs. In this type of branch, **DS** is the parent node for two types of child nodes:

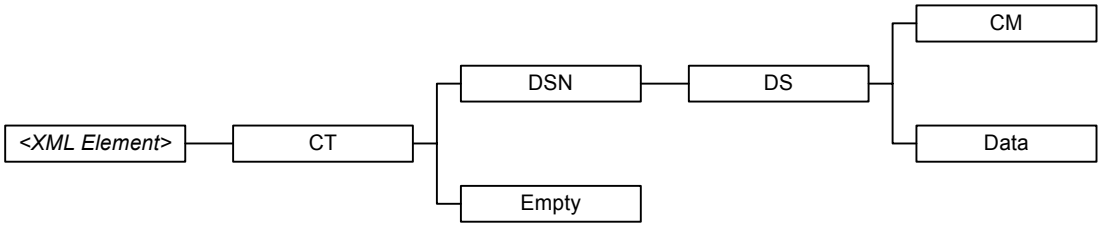
- **CM**, which holds XML comments for the element
- ***<sub-element>***, the name of a sub-element of the parent element

The **DS** node always contains a **CM** child node to hold XML comments. Each ***<sub-element>*** node contains an ETD structure of its own, with the ***<sub-element>*** node as the parent node for the branch.

6.1.2 XML Element without sub-elements

The following diagram illustrates the ETD structure for an XML element that does not have sub-elements.

Figure 21 XML Element without sub-elements

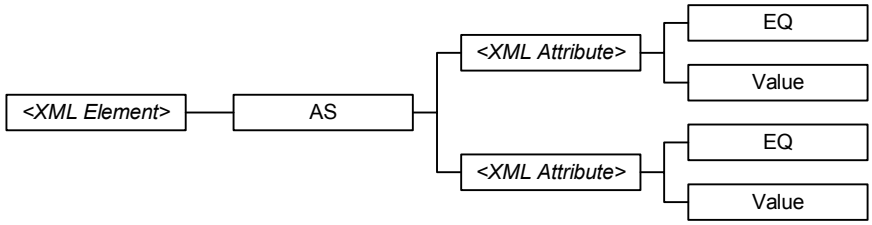


Notice that the only difference between this diagram and the previous diagram is a **Data** child node in place of the ***<sub-element>*** child nodes above. The **Data** node contains the actual data for the XML element that is defined. When creating Collaboration Rules scripts, you must map the XML element data to the **Data** nodes at the terminal end of the element's branch.

6.1.3 XML Attribute

The following diagram illustrates the ETD structure for an XML attribute.

Figure 22 XML Attribute

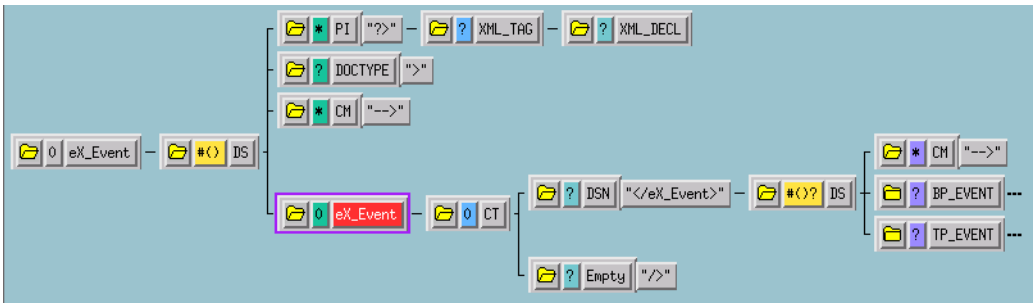


In this case, the XML element contains one child node, **AS**, which identifies the branch as XML attributes of the parent element. The **AS** node contains the ***<XML Attribute>*** nodes as child nodes. Each ***<XML Attribute>*** node has two child nodes: **EQ** to represent the equal sign (=) in the attribute and **Value** which holds the actual value for the attribute. When creating Collaboration Rules scripts, you must map the XML attribute value to the **Value** nodes at the terminal end of the attribute's branch.

6.2 Element Overview

The following diagram illustrates the entire e*Xchange ETD tree. Note that this is only a diagrammatic representation of the tree, since the actual tree conforms to the node structure described in **"ETD Structure" on page 64**.

Figure 23 The e*Xchange ETD



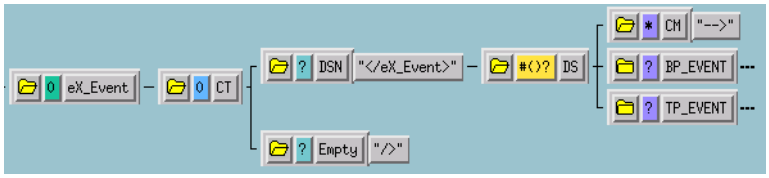
All data pertinent to e*Xchange is contained in the XML element **eX_Event**. **eX_Event** contains two distinct "trees": **BP_EVENT** and **TP_EVENT**. **BP_EVENT** contains all of the information pertaining to e*Insight. **TP_EVENT** contains all of the information pertaining to e*Xchange. Both **BP_EVENT** and **TP_EVENT** are optional nodes in the ETD. So if you use e*Insight to track business process activities but do not use e*Xchange to send data to and from trading partners, you do not need to populate the **TP_EVENT** element. Conversely, if you use e*Xchange to send data to and from trading partners but do not track business process activities in e*Insight, you do not need to populate the **BP_EVENT** element in your Collaboration Rules scripts.

Example: XML Element with Sub-elements

eX_Event is an example of a top-level XML element.

In this example, the **CT**, **DSN**, **DS**, **Empty**, and **CM** facilitator nodes describe the top-level XML element **eX_Event**. Figure 24 shows the ETD structure for this element.

Figure 24 XML Element **eX_Event**



The **eX_Event** parent node contains one child node, **CT**. **CT** identifies **eX_Event** as an XML element. The **CT** node contains two child nodes: **DSN** and **Empty**. **DSN** is the long form of the XML tag (</eX_Event>) and **Empty** is the short form (</>).

The **DSN** and **DS** nodes always occur as parent-child pairs. **DS** is the parent node for three child nodes:

- A **CM** node to hold XML comments for the element.
- **BP_EVENT**, a sub-element of **eX_Event**.
- **TP_EVENT**, a sub-element of **eX_Event**.

The **DS** node always contains a **CM** child node to hold XML comments. In this example, the eX_Event element does not hold data directly, but contains two sub-

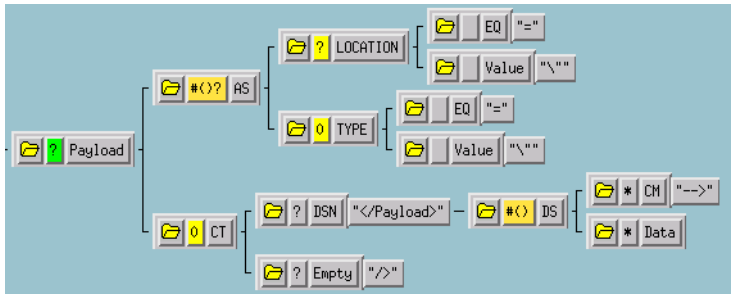
elements—**BP_EVENT** and **TP_EVENT**—which have similar facilitator node branches associated with them.

The following example explains the structure of XML attributes.

Example: XML Element with Attributes

In this example, the **AS** and **EQ** facilitator nodes describe the XML attributes **TYPE** and **LOCATION**. Both are XML attributes of the **Payload** element. Figure 25 shows the ETD structure for these attributes.

Figure 25 XML Attribute **Type**



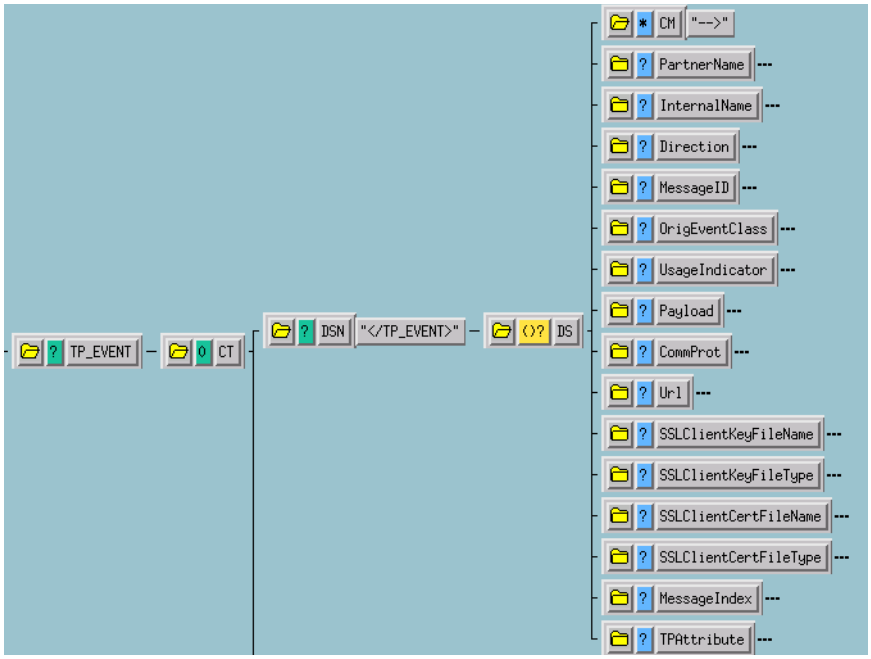
6.3 Using the ETD in e*Xchange

The e*Xchange engine uses the e*Xchange ETD to carry out enveloping and de-enveloping the EDI messages it sends to and receives from trading partners. The **TP_EVENT** location in the e*Xchange ETD contains data the e*Xchange engine uses to track the EDI Event. **TP_EVENT** also contains the actual EDI message stored in the **Payload** node.

TP_EVENT

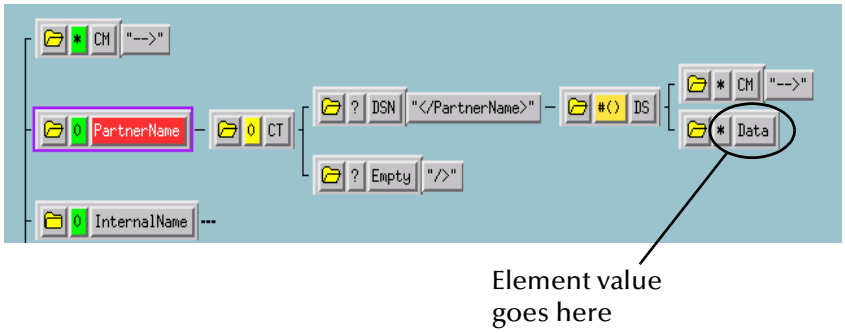
All data relevant to e*Xchange processing is contained in the parent node **TP_EVENT**. **TP_EVENT** contains fifteen elements as shown in Figure 26.

Figure 26 TP_EVENT



Because each of the categories is implemented as an XML element in the e*Xchange ETD structure, the value for the element goes in the **Data** node at the end node structure, as shown in Figure 27.

Figure 27 Location of Element Value



PartnerName

The value for this element must match the **Logical Name** in the Outer Envelope of the trading partner profile.

InternalName

The name of the internal system sending the original data.

Direction

Direction of the transaction. Possible values are "**O**" for outbound Events going to the trading partner or "**I**" for inbound Events coming from the trading partner.

MessageID

A unique ID for each Event originating from a particular internal system. This tag correlates data moving to and from a trading partner, with the original request sent from the internal system.

OrigEventClass

The original Event classification. This tag is used to classify Events, not necessarily originating from the same system, according to functional group. For example, a request for price and availability could originate from one of many systems, but the classification of the Events (QPA) would be the same.

UsageIndicator

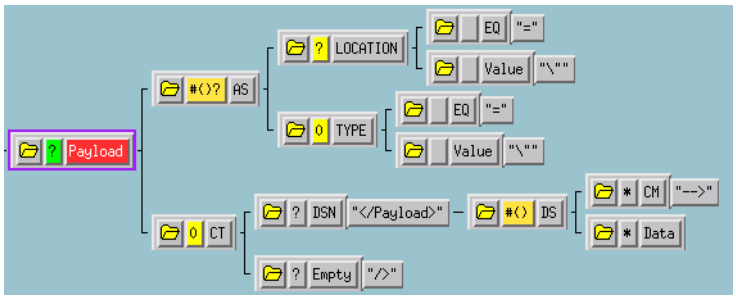
Determines whether the Events being sent to the trading partner are for testing purposes only or are part of a production environment. Possible values are "**T**" for test or "**P**" for production.

Payload

This is the node structure in which you place the EDI message to be processed by e*Xchange.

Unlike the other **TP_EVENT** elements, the **Payload** element has XML attributes associated with it. These attributes qualify the value contained in the terminal **Data** node. Figure 28 shows the **Payload** element's node structure in the e*Xchange ETD.

Figure 28 Payload Node



You must supply values for the element, **Payload**, as well as the attributes for the element, **LOCATION** and **TYPE**.

The following table lists acceptable values for **LOCATION**.

Value	Purpose
"FILE"	Indicates that the value for the element can be found in the file at the location specified in the **Data** node.
"DB"	Indicates that the value for the element can be found in the e*Xchange database at the location specified in the **Data** node.
"URL"	Indicates that the value for the element can be found at the URL location specified in the **Data** node.
"EMBEDDED"	Indicates that the value for the element is contained in the current e*Xchange Event in the **Data** node. This is the default value.
"AUTO"	Reserved for future use.

The following table lists acceptable values for **TYPE**.

Value	Purpose
"RAW"	Indicates that the data in the **Data** node is in ASCII format, but not XML data that has been converted to ASCII using base 64 or some other conversion. The data must not contain any characters that would conflict with the XML nature of the e*Xchange ETD (for example, EDI delimiters that are the same as XML control characters).
"PROCESSED"	Indicates that the data in the **Data** node is XML data that has been encoded using the scheme described in the **ENCODING** node. Currently only base 64 encoding is supported.
"ENCRYPTED"	Indicates that the data in the **Data** node has been encrypted, and must be decrypted before it can be processed by e*Xchange.

The value for **Payload**, the EDI message to be processed by e*Xchange, must be placed in the **Data** node at the end of the **Payload** element's node structure.

CommProt

The communication protocol used by the trading partner. Possible values are **BATCH**, **HTTP**, **HTTPS**, or **TCPIP**.

Url

The destination URL for the trading partner. The data is sent according to the value in this field.

SSLClientKeyFileName

This node contains HTTPS security information.

SSLClientKeyFileType

This node contains HTTPS security information.

SSLClientCertFileName

This node contains HTTPS security information.

SSLClientCertFileType

This node contains HTTPS security information.

MessageIndex

This node contains information used by the "Fast Batch" feature of e*Xchange. Using this feature a number of transactions of the same type can be bundled together and sent out as a single batch transaction to a trading partner. Transactions using fast batch must populate this node in the Event that is sent to e*Xchange using the following format:

<transaction number in bundle>|<total number of transactions to bundle together>

For example, if the Event sent to e*Xchange contains the 4th transaction of 20 that are sent out together, this field must contain "4|20". This is analogous to the "page X of total pages" page numbering format used by some documents. When e*Xchange receives the last transaction in the bundle, labeled 20|20, it sends all 20 transaction out together.

TPAttribute

This is a repeating node containing a name/value pair that is used to add future functionality to e*Xchange without having to change the structure of the eX ETD.

6.4 Sending Data to e*Xchange

You must create a Collaboration Rules Script that prepares the data coming into the e*Xchange system. How complicated this task is depends on the state of the data before the Collaboration processes it.

The Collaboration Rules Script must do the following:

- put the data into the appropriate format

- convert the data to base 64 encoding
- populate the required nodes in the e*Xchange Event sent to e*Xchange for processing

Put the Data into the Required Format

This step is only necessary if the data is not already in the appropriate format required by the trading partner (X12, RosettaNet, or BizTalk). This would be the case where data was coming from SAP in IDoc format and was not preprocessed into the correct format by another Collaboration. In such a case, the Collaboration Rules Script must translate the data into the required format before sending to e*Xchange.

This involves creating an ETD corresponding to the initial state of the data and an ETD corresponding to the required EDI format. Most of these standard ETDs are already pre-created and made available in the e*Xchange suite of tools. Next you build a Collaboration that maps one format to the other. This mapping translation could be called as a sub-translation from the main Collaboration prior to converting the entire message to base 64.

Convert the Event to Base 64 Encoding

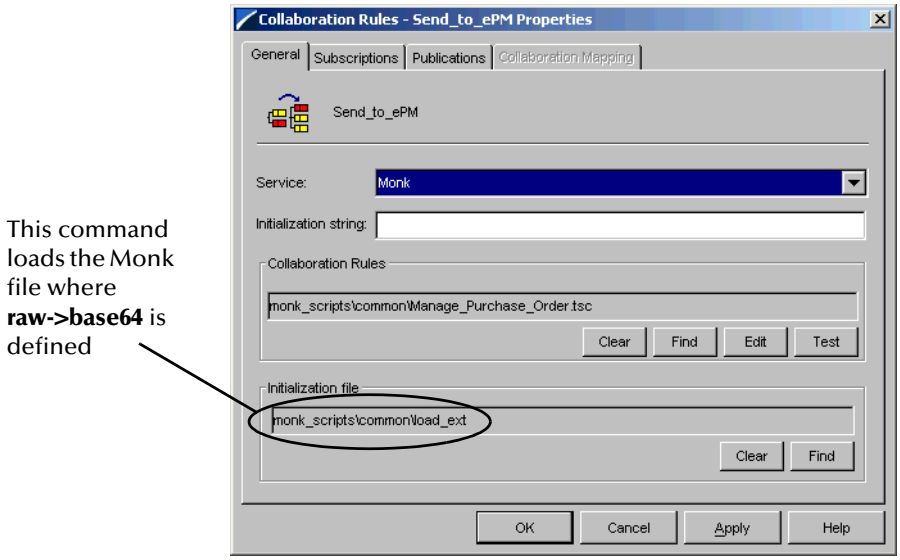
The Collaboration Rules Script must ensure that the data going into e*Xchange doesn't include any characters that cause problems for the XML structure of the standard e*Xchange Event (for example, characters that are the same as the XML control characters). This is done by converting the entire EDI message to base 64 encoding using the Monk function **raw->base64**, before copying it to the payload node of the **eX_Standard_Event** ETD.

Important: *To use the Monk function **raw->base64**, you must make sure the file containing this function has been loaded. You can do this by adding the following command to the **Initialization file** box in the **Collaboration Rules** dialog box for the Collaboration that uses this function:*

```
monk_scripts\common\load_ext
```

This is shown in Figure 29.

Figure 29 Send_to_ePM Collaboration Rules Properties Dialog Box



Populate the Required e*Xchange Nodes

In addition to copying the base 64 encoded message to the payload node of the TP_EVENT portion of the e*Xchange standard Event, you must provide e*Xchange with tracking information about the message.

e*Xchange Tracking Information

e*Xchange needs to know certain things about an message before it can apply the proper enveloping and send it out to the trading partner. The Collaboration Rules Script must supply this information by populating certain required nodes in the Event that is sent to e*Xchange. At a minimum you must tell e*Xchange:

- Direction (inbound or outbound)
- Partner Name (logical name from the outer envelope in the e*Xchange)

All of these requirements can be met by copying the appropriate information to the corresponding nodes in the TP section of the e*Xchange ETD (**eX_Standard_Event.ssc**).

The **TP_EVENT.CT.DSN.DS.Direction.CT.DSN.DS.Data** node must contain the direction of the Event: "O" for outbound to the trading partner or "I" for inbound from a trading partner.

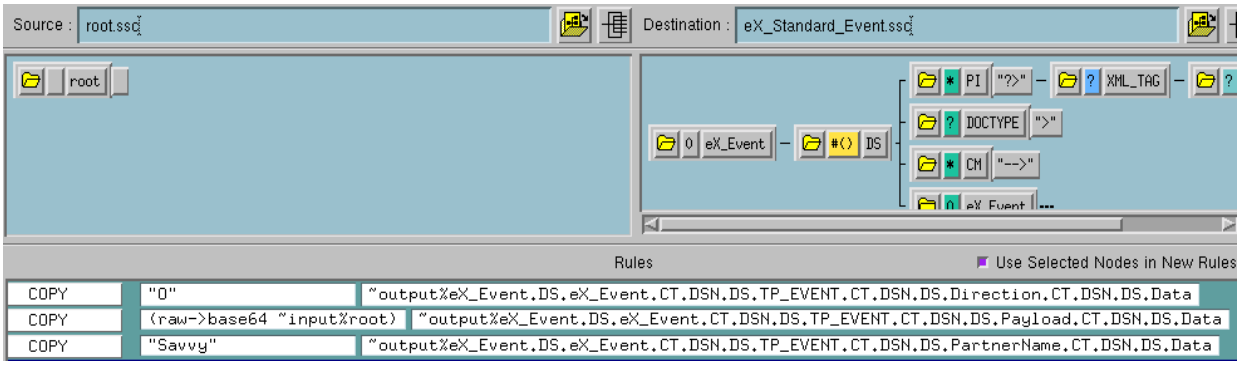
The **TP_EVENT.CT.DSN.DS.PartnerName.CT.DSN.DS.Data** node must contain the name (case-sensitive) of the trading partner from the **Logical Name** box on the **General** tab of the outer envelope for this message.

The e*Xchange Payload

In addition to the tracking information, the **TP_EVENT.CT.DSN.DS.Payload.CT.DSN.DS.Data** node must be filled with the entire base 64 encoded message.

Figure 30 shows a Collaboration Rules Script with the necessary code to populate **eX_Standard_Event.ssc** with the required values.

Figure 30 Example Monk Collaboration Rules Script



Chapter 7

Using the Java e*Xchange ETD

e*Xchange generally uses the Monk service for collaborations. However, the Java collaboration service can be used to transform an event from or to another format when sending the Event into e*Gate.

7.1 Understanding the Java e*Xchange ETD

e*Xchange uses a Java Event Type Definition (ETD) named **eX_StandardEvent.xsc** to define Events as they move from one component to another in the e*Xchange system. The ETD is an XML DTD in SeeBeyond's proprietary messaging format. For a description of the XML DTD see **Appendix A**.

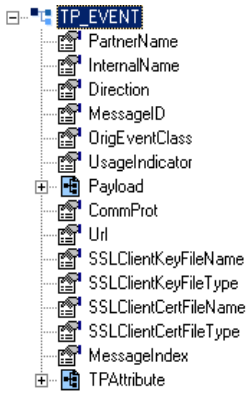
All data going into and coming out of the e*Xchange components is parsed according to the e*Xchange ETD. Understanding this ETD is the key to creating the Collaboration Rules scripts necessary to process the data according to the rules determined by the business process.

Note: *When you install the e*Gate e*Xchange Schema,* ***eIX_StandardEvent.xsc*** *is also created. This ETD has a BP_EVENT location that is used for e*Insight Business Process Manager. You should use this ETD if your implementation requires both e*Insight and e*Xchange. For more information on the BP_EVENT location, refer to the e*Insight Business Process Manager Implementation Guide.*

7.2 Element Overview

The following diagram illustrates the entire e*Xchange ETD tree.

Figure 31 The e*Xchange ETD



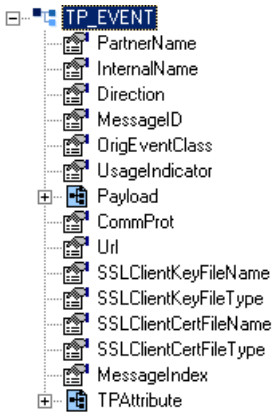
7.2.1 Using the ETD with e*Xchange

The e*Xchange engine uses the e*Xchange ETD to carry out enveloping and de-enveloping messages it sends to and receives from trading partners. The **TP_EVENT** location in the e*Xchange ETD contains data the e*Xchange engine uses to track the Event. **TP_EVENT** also contains the actual EDI message stored in the **Payload** node.

TP_EVENT

All data relevant to e*Xchange processing is contained in the parent node **TP_EVENT**. **TP_EVENT** contains fifteen elements as shown in Figure 32.

Figure 32 TP_EVENT



PartnerName

The value for this element must match the **Logical Name** in the B2B Protocol section of the trading partner profile.

InternalName

The name of the internal system sending the original data.

Direction

Direction of the transaction. Possible values are "**O**" for outbound Events going to the trading partner or "**I**" for inbound Events coming from the trading partner.

MessageID

A unique ID for each Event originating from a particular internal system. This tag correlates data moving to and from a trading partner, with the original request sent from the internal system.

OrigEventClass

The original Event classification. This tag is used to classify Events, not necessarily originating from the same system, according to functional group. For example, a request for price and availability could originate from one of many systems, but the classification of the Events (QPA) would be the same.

UsageIndicator

Determines whether the Events being sent to the trading partner are for testing purposes only or are part of a production environment. Possible values are "**T**" for test or "**P**" for production.

Payload

This is the node structure in which you place the EDI message to be processed by e*Xchange.

Unlike the other **TP_EVENT** elements, the **Payload** element has XML attributes associated with it. These attributes qualify the value contained in the **$text** node. Figure 33 shows the **Payload** element's node structure in the e*Xchange ETD.

Figure 33 Payload Node

You must supply values for the element, **$text**, as well as the attributes for the element, **LOCATION** and **TYPE**.

The following table lists acceptable values for **LOCATION**.

Value	Purpose
"FILE"	Indicates that the value for the element can be found in the file at the location specified in the **Data** node.
"DB"	Indicates that the value for the element can be found in the e*Xchange database at the location specified in the **Data** node.
"URL"	Indicates that the value for the element can be found at the URL location specified in the **Data** node.
"EMBEDDED"	Indicates that the value for the element is contained in the current e*Xchange Event in the **Data** node. This is the default value.

Value	Purpose
"AUTO"	Reserved for future use.

The following table lists acceptable values for **TYPE**.

Value	Purpose
"RAW"	Indicates that the data in the **Data** node is in ASCII format, but not XML data that has been converted to ASCII using base 64 or some other conversion. The data must not contain any characters that would conflict with the XML nature of the e*Xchange ETD (for example, EDI delimiters that are the same as XML control characters).
"PROCESSED"	Indicates that the data in the **Data** node is XML data that has been encoded using the scheme described in the **ENCODING** node. Currently only base 64 encoding is supported.
"ENCRYPTED"	Indicates that the data in the **Data** node has been encrypted, and must be decrypted before it can be processed by e*Xchange.

The value for **Payload**, the EDI message to be processed by e*Xchange, must be placed in the **Data** node at the end of the **Payload** element's node structure.

CommProt

The communication protocol used by the trading partner. Possible values are **BATCH**, **HTTP**, **HTTPS**, or **TCPIP**.

Url

The destination URL for the trading partner. The data is sent according to the value in this field.

SSLClientKeyFileName

This node contains HTTPS security information.

SSLClientKeyFileType

This node contains HTTPS security information.

SSLClientCertFileName

This node contains HTTPS security information.

SSLClientCertFileType

This node contains HTTPS security information.

MessageIndex

This node contains information used by the "Fast Batch" feature of e*Xchange. Using this feature a number of transactions of the same type can be bundled together and sent out as a single batch transaction to a trading partner. Transactions using fast batch must populate this node in the Event that is sent to e*Xchange using the following format:

<transaction number in bundle>|<total number of transactions to bundle together>

For example, if the Event sent to e*Xchange contains the 4th transaction of 20 to be sent out together, this field must contain "4|20". This is analogous to the "page X of total pages" page numbering format used by some documents. When e*Xchange receives the last transaction in the bundle, labeled 20|20, it sends all 20 transaction out together.

TPAttribute

This is a repeating node containing a name/value pair that is used to add future functionality to e*Xchange without having to change the structure of the eX ETD.

7.3 Sending a Message to e*Xchange

The Collaboration Rules Script that sends data to e*Xchange must do the following:

- put the data into the appropriate format
- convert the data to base 64 encoding
- populate the required nodes in the e*Xchange Event sent to e*Xchange for processing

Put the Data into EDI Format

This step is only necessary if the data is not already in the appropriate format required by the trading partner (X12, RosettaNet or BizTalk). This would be the case where data was coming from SAP in IDoc format and was not preprocessed into the required format by another Collaboration. In such a case, the e*Way must translate the data into the required format before sending to e*Xchange.

This involves creating an ETD corresponding to the initial state of the data and an ETD corresponding to the required format. Most of these standard ETDs are already pre-created and made available in the e*Xchange suite of tools. Next you build a Collaboration that maps one format to the other. This mapping translation could be called as a sub-translation from the main Collaboration prior to converting the entire message to base 64.

Convert the Event to Base 64 Encoding

The Collaboration must ensure that the data going into e*Xchange doesn't include any characters that will cause problems for the XML structure of the standard e*Xchange Event (for example, characters that are the same as the XML control characters). This is done by converting the entire EDI message to base 64 encoding using the method **string2base64**, before copying it to the payload node of the **eX_StandardEvent** ETD.

For example:

```
getOut().getTP_EVENT().getPayload().set$Text(Base64.string2base64(get
In().getData()))
```

This converts the contents of **getIn().getData()** to base 64 encoding before copying to the **$text** node.

Populate the Required e*Xchange Nodes

In addition to copying the base 64 encoded message to the payload node of the TP_EVENT portion of the e*Xchange standard Event, you must provide e*Xchange with tracking information about the message.

e*Xchange Tracking Information

e*Xchange needs to know certain things about a message before it can apply the proper enveloping and send it out to the trading partner. The Collaboration Rules Script must supply this information by populating certain required nodes in the Event that is sent to e*Xchange. At a minimum you must tell e*Xchange:

- Direction (inbound or outbound)
- Partner Name (logical name from the outer envelope in the e*Xchange)

All of these requirements can be met by copying the appropriate information to the corresponding nodes in the e*Xchange ETD (**eX_StandardEvent.xsc**).

The **TP_EVENT.Direction** node must contain the direction of the Event: "O" for outbound to the trading partner or "I" for inbound from a trading partner.

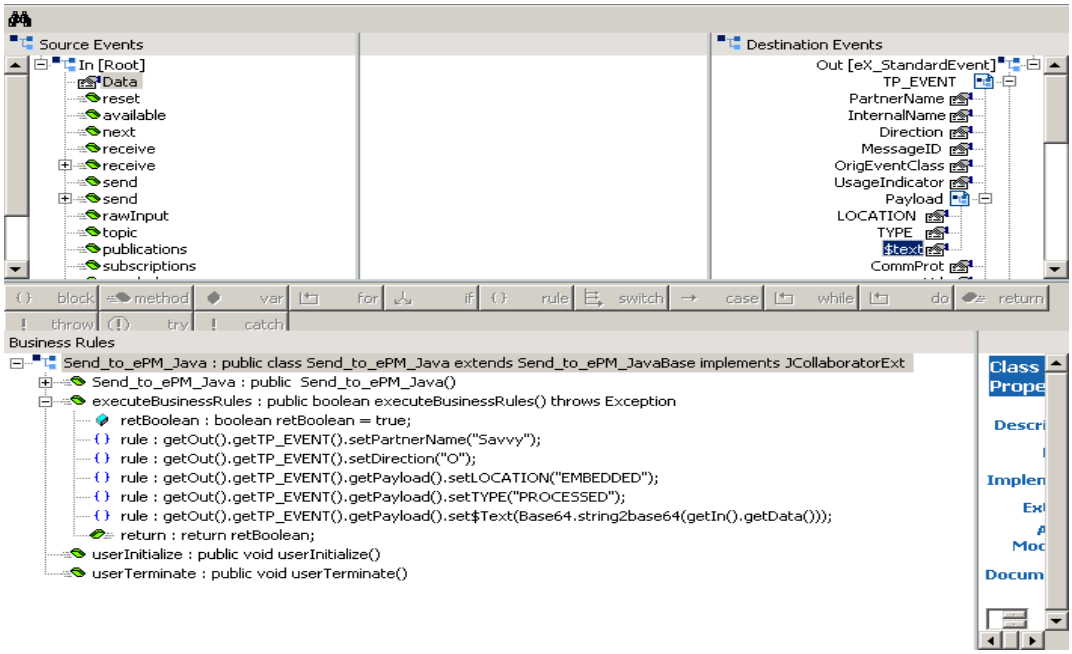
The **TP_EVENT.PartnerName** node must contain the name (case-sensitive) of the trading partner from the **Logical Name** box on the **General** page of the B2B Protocol for this message.

The e*Xchange Payload

In addition to the tracking information, the payload data must be provided. The **TP_EVENT.Payload.$text** node must be filled with the entire base 64 encoded EDI message.

Figure 34 shows a Collaboration Rules Script with the necessary code to populate **eX_StandardEvent.xsc** with the required values.

Figure 34 Example Java Collaboration Rules Script



Chapter 8

Implementation Overview

This chapter provides a high-level overview of the steps involved in an e*Xchange implementation, and by doing so provides background information for the case study chapters that follow it.

8.1 Basic Information

Implementing an e*Xchange system is the process of translating the vision of the business analyst into a functioning system. Once the analyst has determined that a certain business task must be accomplished with e*Xchange, it is the job of the implementor to make this a reality.

You implement e*Xchange by using the e*Xchange GUIs to enter the relevant data into the e*Xchange database. Then you combine the e*Xchange e*Gate components with other e*Gate components you add to create a complete e*Xchange schema. The e*Xchange components are mostly pre-configured and do not require any (or very slight) modification by the implementor. The components that you add are completely user-defined. However, the e*Xchange GUIs and this guide provide a framework for integrating these user-defined components into a working e*Xchange system.

8.1.1 Types of e*Xchange Implementations

The e*Xchange system is designed for the large-scale integration of information systems, both inside and outside of an enterprise, in order to run and monitor business processes. The details of the business processes themselves depend on the nature of the business.

Not every business process takes advantage of every feature built into the e*Xchange suite. Because of this, some e*Xchange implementations can use a simplified eXSchema.

8.2 Implementation Road Map

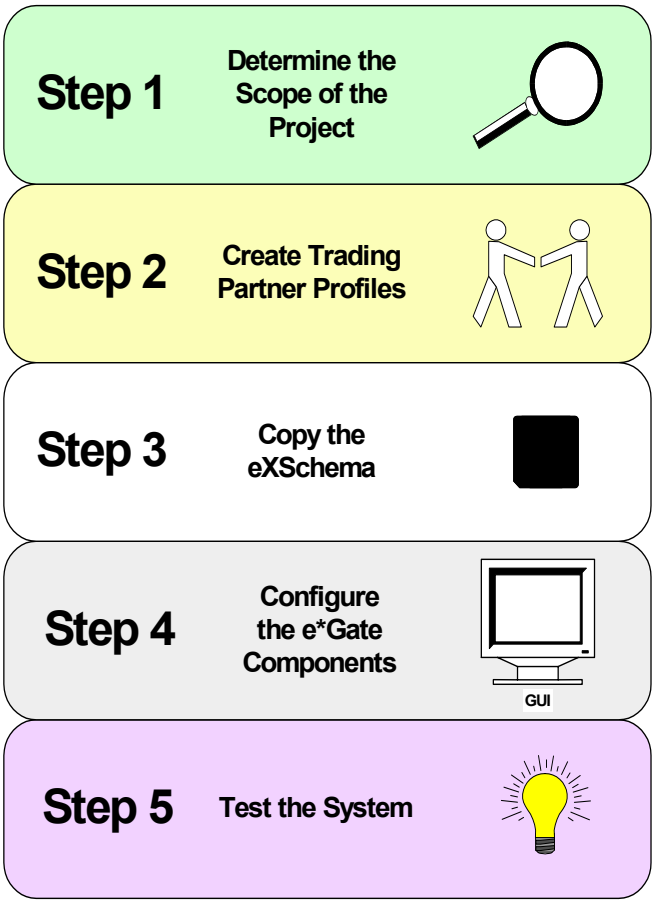
Clearly, each type of implementation involves a different approach. However, at a high level, there are certain similarities.

In general, the work of implementing an end-to-end scenario with e*Xchange involves taking what is created in e*Xchange and integrating it into a working e*Gate schema. e*Gate powers every e*Xchange scenario, and a successful e*Xchange implementation is dependent on a successful e*Gate implementation.

To give you an overview of the complete process, the following implementation road map contains high-level steps for a full e*Xchange implementation. The road map is further refined and given more detail in the case study chapters that immediately follow this one.

Figure 35, on the next page, illustrates the major steps in the integration process for an e*Xchange implementation.

Figure 35 Integration Road Map



Step 1: Determine the Scope of the Project

Determine the type of implementation

The tasks involved in implementing e*Xchange differ depending on the type of implementation.

Analyze the business process

The business analyst must perform the standard tasks of analysis to develop a clear representation of the business process. It is a good idea to have diagrams of the process and a list of the data that must be tracked within the business process. These provide highly beneficial starting points for working with the e*Xchange GUIs.

Step 2: Create Trading Partner Profiles

1. Create the custom validation Collaborations you need. For X12 or UN/EDIFACT protocol implementations, use the Validation Rules Builder tool to help create these validation Collaborations.
2. Enter the trading partner information into the e*Xchange database.

For information on entering Trading Partner information, see the *e*Xchange Partner Manager User's Guide*.

Step 3: Copy the eXSchema

When beginning an integration project, make a copy of the e*Xchange schema, **eXSchema**, that is installed from the CD. Don't make any modifications to eXSchema itself; keep it as a template. Make changes to the copy of the eXSchema that you create. Use this copy as your starting point in e*Gate for supporting e*Xchange.

Use the following procedure to create a copy of the eXSchema:

1. Open the eXSchema in the e*Gate Enterprise Manager GUI.
   - A. Start the e*Gate Enterprise Manager.
   - B. Log in to eXSchema.
2. Export the eXSchema to a file **c:\eGate\client\<*eXSchema backup file name*>**.
   - A. Select **Export Schema Definitions to File ...** from the **File** pull-down menu.
   - B. In the **Select archive File** dialog box enter <*eXSchema backup file name*> in the **File name** text box, then click **Save**.
3. Create a new schema using the eXSchema export file as a template.
   - A. Select **New Schema** from the **File** pull-down menu.
   - B. Enter <*new e*Xchange schema name*> in the text box.
   - C. Mark the **Create from export** check box.
   - D. Click **Find** and browse for the <*eXSchema backup file name*> file created in step 2 above.
   - E. Click **Open**.

     The Enterprise Manager creates a copy of the eXSchema with the schema name entered in step 3B above.

Step 4: Configure the e*Gate Components

Configuring the e*Gate components forms the majority of the integration work done. In this step, you will:

- add and configure the e*Ways that send data into and out of the e*Xchange system
- make all user-configurable associations in the e*Gate GUI

Step 5: Test and Tune the System

It is a good idea to test the system in stages. For example, make sure that one activity works properly before you try to run the entire business process. One good approach is to start with the "upstream" activities at the beginning of the business process, and work your way down to the last activity.

Once you have the entire system working, make adjustments as necessary to improve performance.

Chapter 9

e*Xchange Implementation—X12

This chapter discusses the steps involved to create an e*Xchange implementation that transfers X12 data.

The components for this implementation are provided on your installation CD. For instructions on installing and using the implementation components, see **"Using the Implementation Sample" on page 90**.

9.1 Overview

An e*Xchange implementation makes use of the features designed to add and remove the EDI enveloping information for messages exchanged between trading partners.

In an e*Xchange implementation, use the e*Xchange Partner Manager Web interface to set up the trading partner information, and the e*Gate Enterprise Manager GUI to add user-defined e*Gate components to provide connectivity to the business application or trading partner. Once this is done, the pre-configured e*Xchange e*Gate schema components handle enveloping and de-enveloping Events as they travel through the e*Xchange system.

The major steps for an e*Xchange implementation are as follows:

1. Create any needed validation Collaborations.
2. Create the Trading Partner profiles.
3. Configure the user-defined e*Ways that will connect the business application to e*Xchange.
4. Configure the e*Xchange e*Way.
5. Run and test the scenario.

9.1.1 Case Study: Sending an X12 850 Purchase Order

The case study discussed in this chapter illustrates one possible implementation of sending out a purchase order to a trading partner.

In this example, an X12 Version 4010 850 purchase order is sent out from a (simulated) internal application to an external trading partner using a Batch e*Way. The X12 enveloping is automatically added to the message by e*Xchange based on trading

partner information retrieved from the e*Xchange database, before it is sent to the outbound Batch e*Way.

Typically, the purchase order information would be provided by a business application and may or may not be in X12 format. A user-defined e*Way must be created to connect to a business application in order to receive the data and put it into the proper X12 format. The schema contains an e*Way named **Send_to_ePM** that you can use as a starting point. In order to simplify this example, the purchase order information is provided in the form of a text file that is already in X12 850 format.

This example provides instructions for creating e*Ways that use both the Java and Monk Collaboration Services to create the event that is sent to e*Xchange.

Note: *This example does not use a return acknowledgment. Therefore, the step covering configuration of the e*Ways used to receive data back from a trading partner is not covered in this chapter.*

Figure 36 e*Xchange Scenario Data Flow

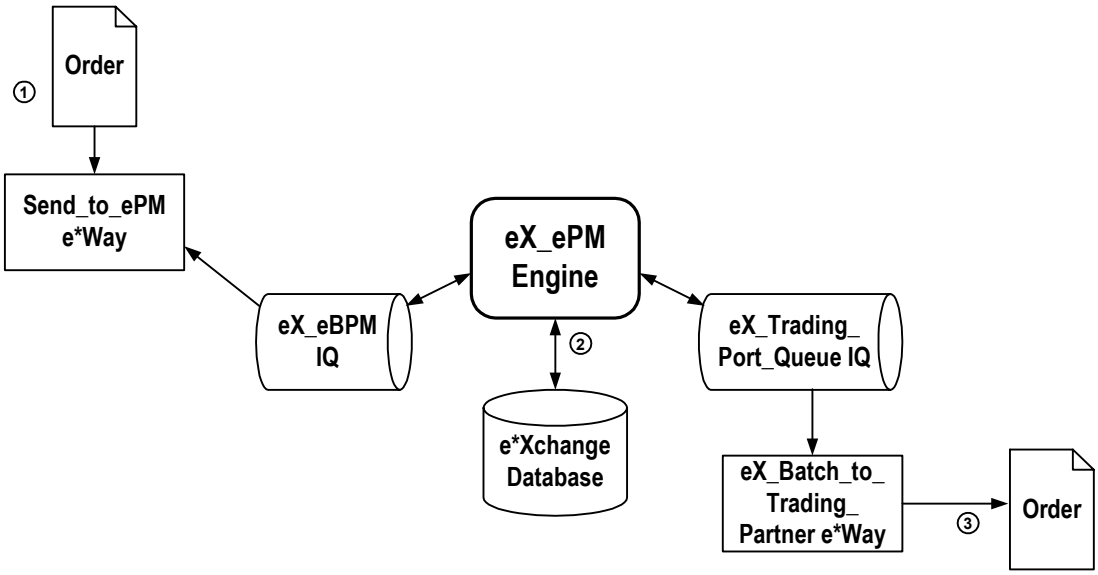


Figure 36 data flow description

① The **Send_to_ePM_Java** or **Send_to_ePM_Monk** e*Way picks up the text file containing the 850 Purchase order, puts the data into standard e*Xchange format, adds the tracking information required by e*Xchange, and then publishes it to the **eX_eBPM** IQ in e*Gate.

② The **eX_ePM** e*Way picks up the e*Xchange Event from the **eX_eBPM** IQ, retrieves the trading partner information from the e*Xchange database, and then uses the retrieved trading partner information to add the X12 enveloping to the Event, and then places it in the **eX_Trading_Port_Queue** IQ using the **eX_BATCH** Event Type.

③ The **eX_Batch_to_Trading_Partner** e*Way picks up the **eX_BATCH** Event from the **eX_Trading_Port_Queue** IQ, and then sends the message via FTP to the trading partner.

9.2 Using the Implementation Sample

The components for this implementation are provided on your installation CD, and are located in **\setup\ex\sample\X12_Implementation_Sample.zip**. Follow these steps to install the components:

1 Unzip the file to a local directory.

2 Install the e*Gate schema using one of the following commands. The instructions refer to the schema name **X12**, however, this is user-defined.

Note: *The default registry port number is 23001.*

A For Unix

```
sh install_po.sh <egate_registry_host_name> <schema_name>
<user_name> <password> <egate_registry_port_num>
```

B For Windows

```
install_po.bat <egate_registry_host_name> <schema_name>
<user_name> <password> <egate_registry_port_num>
```

3 Use the e*Xchange Import function in e*Xchange Client for Windows or e*Xchange Repository Manager, to import **Savvy_Toy_Company.exp** into e*Xchange Partner Manager.

4 Copy eXchange_PO.~in to**<egate>\client\data\eXchange**.

5 Refer to **"Step 3: Set Up the B2B Protocol Information" on page 95** and ensure that the directory referred to in the **File Name** parameter matches the location of the file **eXchange_PO.~in**, as set up in step 4 above. If it does not, change the **File Name** parameter value.

6 Refer to either **Table 12 on page 98** (Java) or **Table 15 on page 101** (Monk) and ensure that the directory referred to in the **PollDirectory** parameter exists. If it does not, either create the directory or change the **PollDirectory** parameter value.

7 Configure the **eX_ePM** e*Way as described in **"Configure the eX_ePM e*Way" on page 104**.

The steps on the following pages describe how the components for this implementation were created. See **"Run and Test the e*Xchange Scenario" on page 105** for instructions to run the implementation.

9.3 Create Necessary Validation Collaborations

Creating an X12 validation Collaboration is a two-step process:

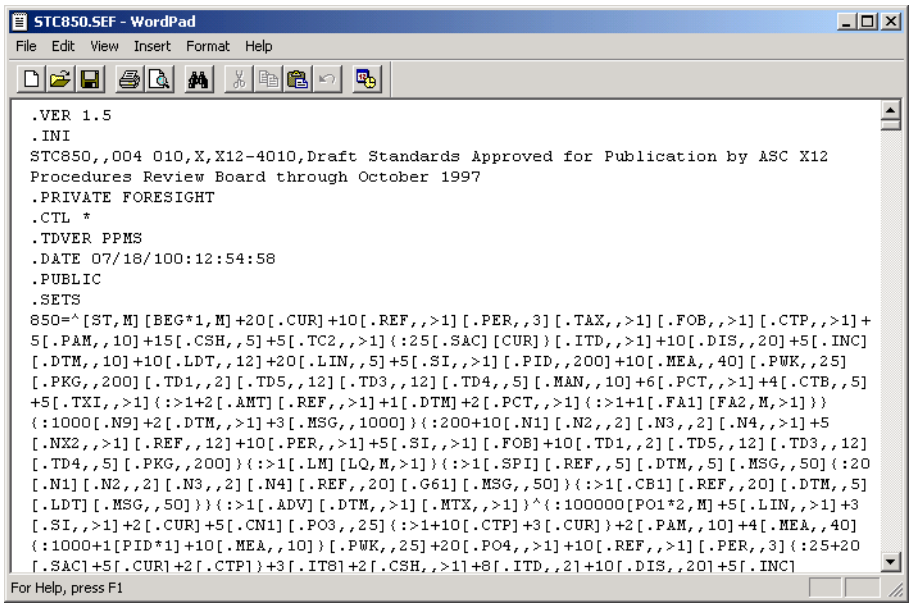
1. Create the Standard Exchange Format (SEF) file.
2. Use the Validation Rules Builder (VRB) tool to create the validation Collaborations based on the SEF file.

9.3.1 Create the SEF File

e*Xchange includes many generic X12 validation Collaborations that you can use to verify the format of X12 EDI messages. You can use these as supplied, or modify them to fit your particular needs. In addition, the e*Xchange suite includes a tool, the VRB, for building customized validation Collaborations directly from a SEF file. You create the SEF file with a third-party Implementation Guide editor using the EDI Implementation Guide for your industry.

Figure 37 shows a portion of the SEF file created for the e*Xchange example, using Foresight's EDISIM software.

Figure 37 e*Xchange SEF File

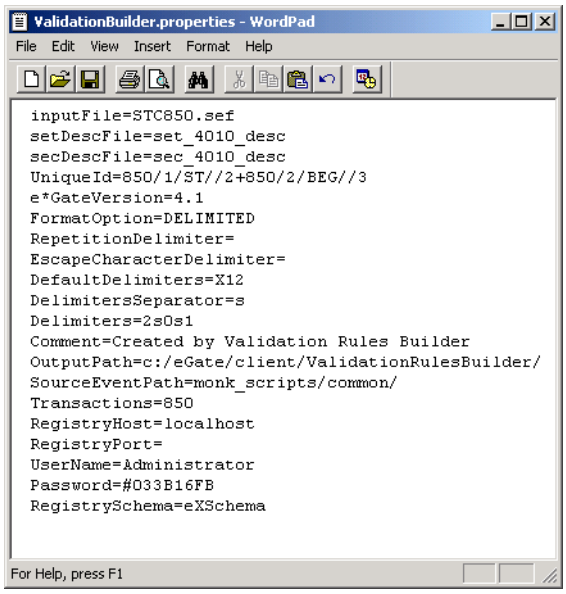


9.3.2 Create the Validation Collaboration with the VRB

To create the validation Collaboration using the VRB, you must edit the VRB properties file, and then run the VRB tool against the SEF file. This process generates the corresponding CRS (**.tsc**) and ETD (**.ssc**) files needed by e*Xchange.

Figure 38 shows the edited **ValidationBuilder.properties** file used to create the validation Collaboration used in this example.

Figure 38 Edited VRB Properties File



Once the VRB file has been edited and saved you must run the VRB tool.

- From a command prompt within the **ValidationRulesBuilder** directory, enter the following command:

```
java -jar c:\eGate\client\classes\ValidationBuilder.jar
```

The VRB tool creates the following two files and places them in the **ValidationRulesBuilder** directory:

- **X12_850PurcOrde_4010.ssc**
- **X12_850PurcOrde_4010.tsc**

The VRB also commits these two files to the e*Gate Registry under the eXSchema.

See "Using the Validation Rules Builder" in the *e*Xchange Partner Manager User's Guide* for more information on how to create validation Collaborations using this tool.

9.4 Create the Trading Partner Profiles

The trading partner profiles in e*Xchange Partner Manager act as the repositories for the information necessary to send EDI messages back and forth between the entities. They contain all of the information to properly envelope an Event and forward it to its correct destination.

When creating trading partner profiles, check your values carefully before saving or leaving a section/screen, because many values cannot be changed once they are committed to the database due to auditing restrictions. You can inactivate erroneous information and add the correct information under a different company, B2B Protocol, and so on.

Refer to the *e*Xchange Partner Manager User's Guide* for detailed assistance with the process of creating trading partner profiles.

Trading Partner Information Hierarchy

e*Xchange stores trading partner information at various levels. The process of creating a trading partner profile proceeds from the most general inclusive level, that of a company with which you do business, to the most specific information regarding an message that you wish to send (the message profile).

9.4.1 The Savvy Toy Company Trading Partner

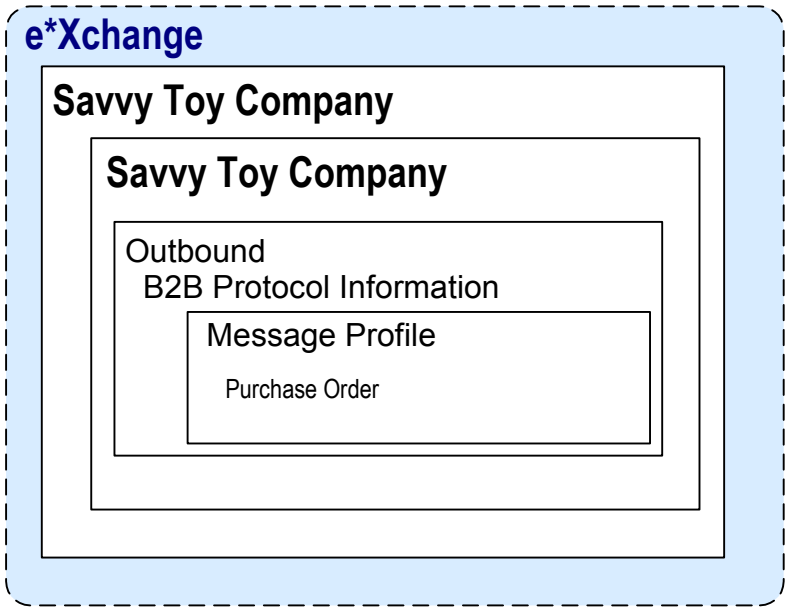
The Savvy Toy Company (Savvy) is a manufacturer of high quality toys that uses the X12 format to exchange business data with its customers. In our example we send a purchase order to Savvy for one of their products, "the Millennium Pet Rock."

The following procedure and accompanying tables were used to create the Savvy trading partner for this example.

Figure 39 shows an overview of the components that you need to create for this example, including,

- Company
- Trading Partner
- B2B Protocol Information
- Message Profiles

Figure 39 The Savvy Toy Company Overview



To configure the Savvy Toy Company Trading Partner Profile you must follow the steps listed below:

- **Step 1: Create the Company** on page 94
- **Step 2: Create the Trading Partner** on page 94
- **Step 3: Set Up the B2B Protocol Information** on page 95
- **Step 4: Create the Message Profile** on page 96

Step 1: Create the Company

1. Log in to the e*Xchange Web interface.
2. From the **Main** page, click **Profile Management**.
3. From the **Company** page, click **New**.
4. In the **Company - Adding** page, enter the **Company** name, "Savvy Toy Company".
5. Click **Next**.

   This saves your changes and returns to the **Company** page.

Note: *The security information is automatically configured for the current user.*

Step 2: Create the Trading Partner

1. From the **Company** page, ensure that "Savvy Toy Company" is selected, and click **Continue: Trading Partner**.

2 From the **Trading Partner** page, click **New** to access the **Trading Partner - adding** page.

3 Enter the **Trading Partner Name**, "Savvy Toy Company".

4 Click **Next**.

This saves your changes and returns to the **Trading Partner** page.

The required security information defaults from the company level.

Step 3: Set Up the B2B Protocol Information

1 From the **Trading Partner** page, ensure that the "Savvy Toy Company" is selected, and click **Continue: B2B Protocol**.

2 From the **B2B Protocol** page, click **New** to access the **B2B Protocol - Adding** page.

3 Enter the information listed in Table 6.

In an actual implementation, your local administrator can provide you with the B2B Protocol information. For an explanation of the B2B Protocol parameters, see the *e*Xchange Partner Manager User's Guide*.

Table 6 B2B Protocol Information

Parameter	Value
eBusiness Protocol	X12
Version	4010
Direction	Outbound

4 Click **Next** to save your changes and access the **General** section.

5 Enter the information listed in Table 7.

Table 7 B2B Protocol Information, General Page

Parameter	Value
Logical Name	Savvy
Status	Active
Communication Protocol	FTP (BATCH)

6 Click **Next** to save your changes and access the **Transport Component** section.

7 In the **File Name** window, enter **<drive>\SavvyOut\Savvy850_Out_%d_%3#.dat**.

Note: *You must create the directory* ***<drive>\SavvyOut*** *before running the Schema.*

8 Click **Next** to accept the values and access the **Message Security** page.

9 No changes are required. Click **Finish** to save the information and return to the **B2B Protocol** page.

Step 4: Create the Message Profile

1 From the **B2B Protocol** page, click **Continue: Message Profile**.

2 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

3 Enter the information listed in Table 8.

Note: *This table only lists the attributes required to make this scenario work.*

Table 8 General (X12_850PurcOrde_4010)

Name	Value
Name	Savvy 850 Outbound PO
Transfer Mode	Interactive
Validation Collaboration	X12_850PurcOrde_4010

4 Click **Next** to access the **Interchange Control Envelope** section. Enter the information listed in Table 9.

Table 9 Interchange Control Envelope (X12_850PurcOrde_4010)

Name	Value
ISA06 Interchange Sender Identifier	eBiz01
ISA08 Interchange Receiver Identifier	Savvy01
ISA13 IC Control Number	2

5 Click **Next** to access the **Functional Group Envelope** section. Enter the information listed in Table 10.

Note: *This table only lists the attributes required to make this scenario work.*

Table 10 Functional Group Envelope (X12_850PurcOrde_4010)

Name	Value
GS01 Functional Identification Code	PO
GS02 Application Sender Code	eBiz01
GS03 Application Receiver Code	Savvy01
GS06 Group Control Number	2

6 Click **Next** to access the **Transaction Set Envelope** section. Enter the information listed in Table 11.

Note: *This table only lists the attributes required to make this scenario work.*

Table 11 Transaction Set Envelope (X12_850PurcOrde_4010)

Name	Value
ST01 Transaction Set Identification Code	850
ST02 TS Control Number	1

7 Click **Next** to access the **Return Messages** section.

8 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

9.5 Clone the eXSchema

The supplied schema named eXSchema contains the components required to run e*Xchange. Make a copy of this schema and then configure the copy for this implementation.

To make a copy of eXSchema

1 Open eXSchema in the e*Gate Enterprise Manager GUI.

2 Export eXSchema.

3 Create a new schema named **X12** using the exported file.

9.6 Configure the e*Way to Send the Message to e*Xchange

The component (e*Way or BOB) that feeds data into e*Xchange must put the data into the appropriate business protocol format. It must also populate the required fields in the e*Xchange Event that is processed by e*Xchange.

This component is entirely user-defined and must be added to the eXSchema. The type of component to use depends on whether a connection to a system outside e*Gate must be made, and if so, what type of system. Typically, this component is an e*Way that connects to a business application such as SAP that sends out electronic messages. These messages may or may not be in the format required by the trading partner to which they are being sent. If the data is not in the correct format, the e*Way must translate the data into the required format before it is sent to the e*Xchange system for enveloping and forwarding to the trading partner.

The e*Xchange Send_to_ePM e*Way

This example simulates the publication of an electronic purchase order from an internal (to the company, but not to e*Gate) accounting application to a shared location on the network file system. This file, which is already in X12 850 format, is then picked up by a file e*Way and moved into e*Xchange.

This example shows the configuration steps for creating the e*Way using the Java Collaboration Service (see **"Configuring the Send_to_ePM_Java e*Way"**) and the Monk Collaboration Service (see **"Configuring the Send_to_ePM_Monk e*Way" on page 101**).

9.6.1 Configuring the Send_to_ePM_Java e*Way

Follow these steps to configure **Send_to_ePM_Java** e*Way.

1. Create the configuration file.
2. Create the ETDs.
3. Create the Collaboration Rule and Collaboration Rules Script.
4. Create the Collaboration.

Step 1: Edit the Send_to_ePM_Java e*Way Configuration File

1. In the **Configuration file** area of the **General** tab, in the **e*Way Properties** dialog box, click **Clear**, and then click **New**.
2. Configure the **Send_to_ePM_Java** e*Way parameters using Table 12.

Table 12 Send_to_ePM_Java e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\eXchange
	(All others)	(Default)
Performance Testing	(All)	(Default)

3. When finished editing the e*Way configuration file, save your work and close the e*Way editor.
4. Click **OK** to close the **e*Way Properties** dialog box.

Step 2: Create the Send_to_ePM_Java ETDs

In the case where the **Send_to_ePM_Java** e*Way connects to a business application, you must create an ETD that corresponds to the business application. For example, an SAP system sends out EDI messages in IDoc (SAP proprietary) format. In order to work with these messages you must create an ETD that corresponds to the IDoc.

In the present example, since the data is already in standard X12 850 format for a purchase order, you can bring in the Event without parsing it. To do this, all that is required is an ETD with a root node.

Step 3: Create the Send_to_ePM_Java Collaboration Rule and Collaboration Rule Script

The **Send_to_ePM_Java.xpr** CRS used in the present example is shown in Figure 40. It does the following:

- Converts the X12 850 message to base 64 encoding, and copies it to the Payload node of the TP_EVENT section of the e*Xchange standard Event.
- Sets the Payload type and location.
- Copies "O" for outbound to the direction node of the TP_EVENT section.
- Copies the trading partner logical name "Savvy" to the PartnerName node of the TP_EVENT section.

To create and configure the Send_to_ePM_Java Collaboration Rule and Collaboration Rule Script

1. Create a new Collaboration Rule named **Send_to_ePM_Java**.
2. From **Send_to_ePM_Java** Collaboration Rule properties, select the **General** tab. Select **Java** as the **Service**.
3. Select the **Collaboration Mapping** tab. Create two instances, and configure as shown in Table 13.

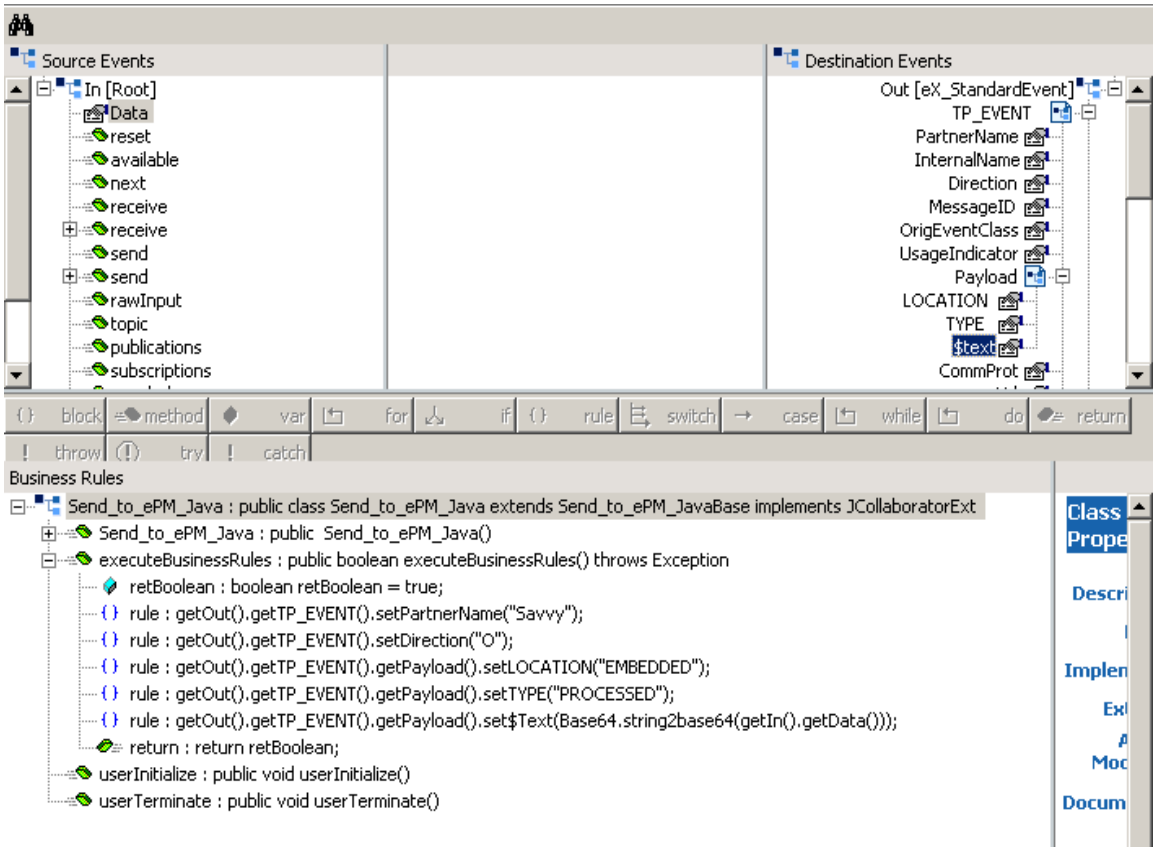
Table 13 Send_to_ePM_Java CR configuration - Collaboration Mapping Tab

Instance Name	ETD	Mode	Manual Publish
In	Root.xsc	In	N/A
Out	eX_StandardEvent.xsc	Out	

4. Select the **General** tab, and then click **New**.

   The Collaboration Editor opens.
5. Add the rules shown in Figure 40.

Figure 40 Send_to_ePM_Java.xpr



Step 4: Create the Send_to_ePM_Java Collaboration

Once the CRS has been created, you must set up the Collaboration Properties for the **Send_to_ePM_Java** Component in the Enterprise Manager GUI.

To create and configure the Send_to_ePM_Java Collaboration

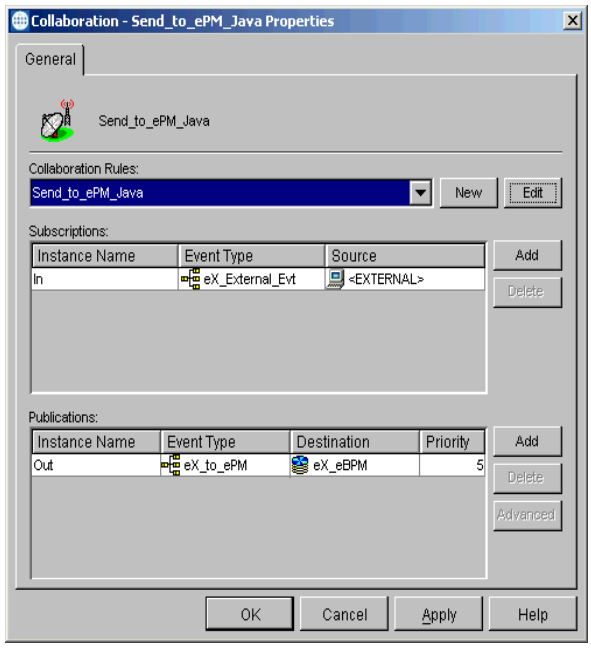
1. Select the **Send_to_ePM_Java** e*Way.
2. Create a new Collaboration named **Send_to_ePM_Java**.
3. Configure the **Send_to_ePM_Java** Collaboration properties using Table 14.

Table 14 Send_to_ePM_Java Collaboration configuration

Section	Value
Collaboration Rule	Send_to_ePM_Java
Subscriptions	Instance: In Event Type: eX_External_Evt Source: <EXTERNAL>
Publication	Instance: Out Event Type: eX_to_ePM Destination: eX_eBPM

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 41.

Figure 41 Send_to_ePM_Java Collaboration Properties



9.6.2 Configuring the Send_to_ePM_Monk e*Way

Follow these steps to configure **Send_to_ePM_Monk** e*Way.

1. Create the configuration file.
2. Create the ETDs.
3. Create the Collaboration Rule and Collaboration Rules Script.
4. Create the Collaboration.

Step 1: Edit the Send_to_ePM_Monk e*Way Configuration File

1. In the **Configuration file** area of the **General** tab, in the **e*Way Properties** dialog box, click **Clear**, and then click **New**.
2. Configure the **Send_to_ePM_Monk** e*Way parameters using Table 15.

Table 15 Send_to_ePM_Monk e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\eXchange
	(All others)	(Default)

Table 15 Send_to_ePM_Monk e*Way Parameters

Screen	Parameter	Setting
Performance Testing	(All)	(Default)

3 When finished editing the e*Way configuration file, save your work and close the e*Way editor.

4 Click **OK** to close the **e*Way Properties** dialog box.

Step 2: Create the Send_to_ePM_Monk ETDs

In the present example, since the data is already in standard X12 850 format for a purchase order, you can bring in the Event without parsing it. To do this, all that is required is an ETD with a root node.

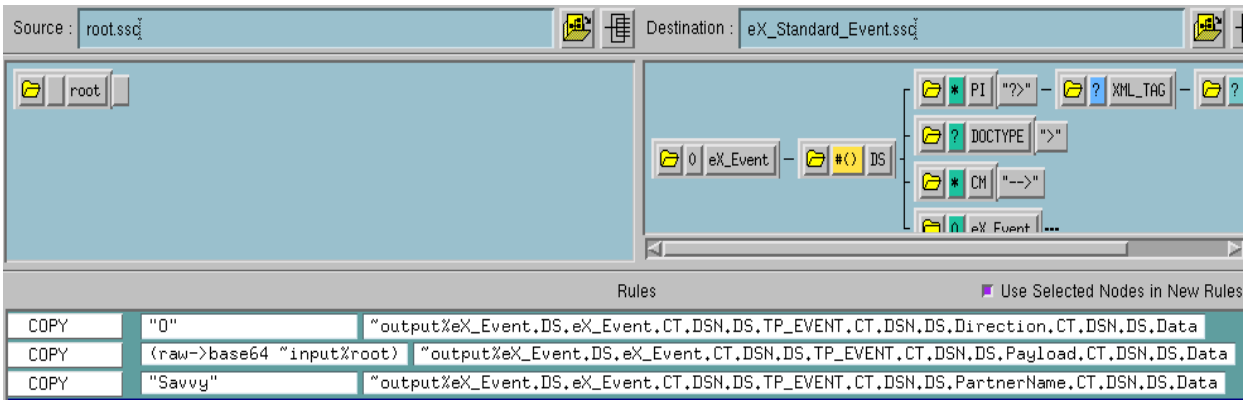
Step 3: Create the Send_to_ePM_Monk Collaboration Rules Script

The **Send_to_ePM_Monk.tsc** CRS used in this example is shown in Figure 42. It does the following:

- Converts the X12 850 message to base 64 encoding, and copies it to the Payload node of the TP_EVENT section of the e*Xchange standard Event.
- Copies "O" for outbound to the direction node of the TP_EVENT section.
- Copies the trading partner logical name "Savvy" to the PartnerName node of the TP_EVENT section.

Figure 42 shows the CRS used in this example.

Figure 42 Send_to_ePM_Monk.tsc



Step 4: Create the Send_to_ePM_Monk Collaboration Rule

To create and configure the Send_to_ePM_Monk Collaboration Rule

1 Create a new Collaboration Rule named **Send_to_ePM_Monk**.

2 From the **Send_to_ePM_Monk** Collaboration Rule properties, select the **General** tab. Configure as shown in Table 16.

Table 16 Send_to_ePM_Monk CR configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	Send_to_ePM_Monk
Initialization File	monk_scripts\common\load_ext

3 Select the **Subscriptions** tab. Select **eX_External_Evt** and move it to the right pane.

4 Select the **Publications** tab. Select **eX_to_ePM** and move it to the right pane.

Step 5: Create the Send_to_ePM_Monk Collaboration

Once the CRS has been created, you must set up the Collaboration Properties for the **Send_to_ePM_Monk** Component in the Enterprise Manager GUI.

To create and configure the Send_to_ePM_Monk Collaboration

1 Select the **Send_to_ePM_Monk** e*Way.

2 Create a new Collaboration named **Send_to_ePM_Monk**.

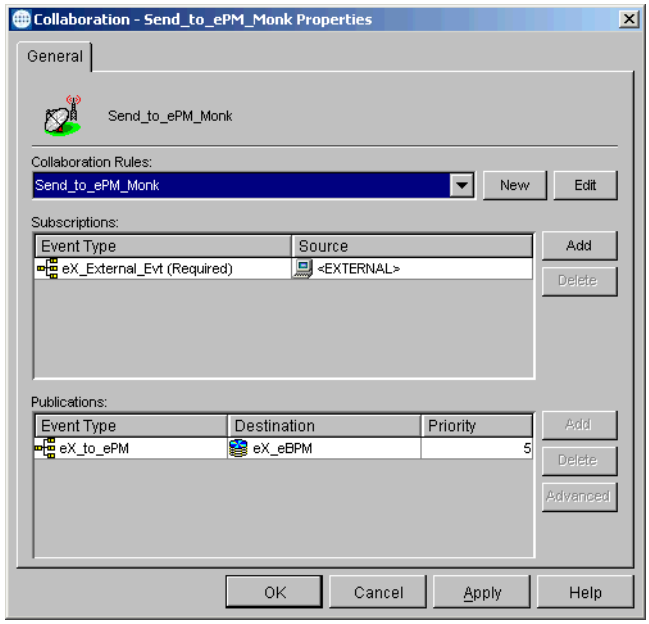
3 Configure the **Send_to_ePM_Monk** Collaboration properties using Table 17.

Table 17 Send_to_ePM_Monk Collaboration Configuration

Section	Value
Collaboration Rule	Send_to_ePM_Monk
Subscriptions	Event Type: eX_External_Evt Source: <EXTERNAL>
Publications	Event Type: eX_to_ePM Destination: eX_eBPM

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 43.

Figure 43 Send_to_ePM_Monk Collaboration Properties



9.7 Configure the eX_ePM e*Way

The **eX_ePM** e*Way requires only minimal configuration. You must give it the logon information for the e*Xchange database.

To configure the eX_ePM configuration file

1. In the **eX_ePM** e*Way properties, select the **General** tab.
2. In the **Configuration File** area, click **Edit**.
3. Configure the parameters as shown in Table 18.

Table 18 eX_ePM e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Communication Setup	(All)	(Default)
Monk Configuration	(All)	(Default)

Table 18 eX_ePM e*Way Parameters

Screen	Parameter	Setting
Database Setup	Database Name)	(Service name of the e*Xchange database)
	User name	ex_admin
	Password	ex_admin
	(All others)	(Default)

9.8 Configure Any Other e*Gate Components

The remaining component in the e*Xchange schema is the **eX_Batch_to_Trading_Partner** e*Way. This component works without any user configuration.

The Batch e*Way uses the information in the trading partner profile to FTP the message to the trading partner. In this example this is simulated by making a local copy of the file.

9.9 Run and Test the e*Xchange Scenario

Once the schema has been set up in e*Gate you can run and test the scenario.

1. Make a final check of the e*Gate schema. Also, make sure the **eX_ePM**, and **eX_Batch_to_Trading_Partner**, and either **Send_to_ePM_Java** or **Send_to_ePM_Monk**, components are set to auto-start.

Important: *Verify that any other components in the schema that are not being used are not set to auto-start or are moved to an unused host.*

2. At the command line, start the schema:

```
stccb.exe -rh localhost -rs <schema name> -ln localhost_cb -un
Administrator -up STC
```

3. Start the e*Gate Monitor and check the status of all the components. Any components used in the e*Xchange scenario that are red, indicating they are not running, should be investigated before feeding data into the system.
4. Using Windows Explorer (or the equivalent) navigate to the location for the input data file, **eXchange_PO.~in** (**<eGate>\client\data\eXchange**).
5. Change the extension to ".fin". Watch as the data file name changes to ".~in" indicating that the data file has been picked up.
6. Navigate to the location to which you are sending the output file by FTP. If everything is working correctly, an output file should appear in the directory indicating successful completion of the EDI exchange. Note that the enveloping

information has been added to the information contained in the input file by the e*Xchange.

Viewing the Results in Message Tracking

You can view the results of the message processing by using the Message Tracking feature of e*Xchange.

To view the outbound message in Message Tracking

1. From e*Xchange Web Interface, **Main** page, select **Message Tracking**.

   The **TP Profile Selection** page appears.
2. In the **Company Profile** field, select **Savvy Toy Company**.
3. In the **Trading Partner Profile** field, select **Savvy Toy Company**.
4. In the **eBusiness Protocol** field, select **X12**.
5. In the **Direction** field, select **Outbound**.
6. Click the **Message Profile Selection**.
7. Select the **Savvy 850 Outbound PO** message.
8. Click the **Message Details** link to view the resulting list.

9.10 Editing the Data File

If you want to send the data again, you need to edit the data to ensure that it is unique. Open **<eGate>\client\data\eXchange\eXchange_PO.~in** and change the field shown in bold in to a unique value.

Figure 44 Sample data file

```
ST*850*0001~BEG*01*BK*99AKDF9DAL393*394839201938 43*20C
```

Chapter 10

e*Xchange Implementation—UN/EDIFACT

This chapter discusses the steps involved to create an e*Xchange implementation that transfers UN/EDIFACT data.

The components for this implementation are provided on your installation CD. For instructions on installing and using the implementation components, see **"Using the Implementation Sample" on page 110**.

10.1 Overview

An e*Xchange implementation makes use of the features designed to add and remove the EDI enveloping information for messages exchanged between trading partners.

In an e*Xchange implementation, use the e*Xchange Web Interface to set up trading partner information, and the e*Gate Enterprise Manager GUI to add user-defined e*Gate components to provide connectivity to the business application or trading partner. Once this is done, the pre-configured e*Xchange e*Gate schema components handle enveloping and de-enveloping Events as they travel through the e*Xchange system.

The major steps for an e*Xchange implementation are as follows:

1. Create any needed validation Collaborations.
2. Add the new validation Collaborations and configure envelope profiles in the e*Xchange GUI.
3. Create the trading partner profiles.
4. Configure the user-defined e*Ways that will connect the business application to e*Xchange and exchange messages with the trading partner.
5. Configure the e*Xchange e*Way.
6. Run and test the scenario.

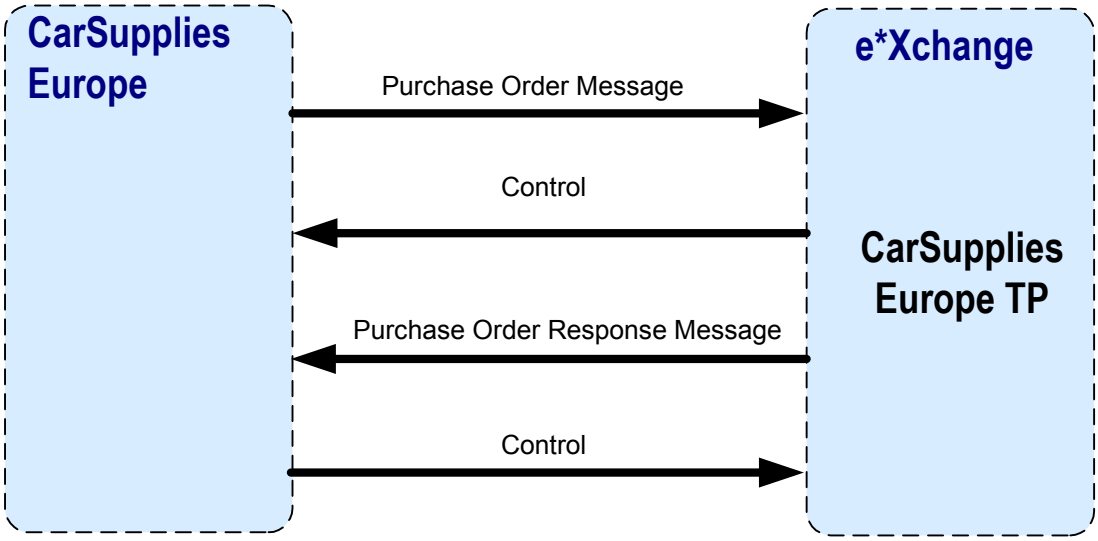
10.1.1 Case Study: Sending an UN/EDIFACT Purchase Order

The case study discussed in this chapter illustrates one possible implementation of receiving a purchase order from a trading partner.

In this example, a UN/EDIFACT purchase order is received from an external trading partner. The UN/EDIFACT enveloping is automatically removed from the message by

e*Xchange based on trading partner information retrieved from the e*Xchange database, and then it is sent to an internal system. A control message is immediately returned to the Trading Partner. Then the purchase order response is sent to the Trading Partner, and the Trading Partner returns a control message to complete the cycle. Figure 45 shows the message flow.

Figure 45 UN/EDIFACT Message Flow



Typically, the purchase order information would be provided by a business application and may or may not be in UN/EDIFACT format. A user-defined e*Way must be created to connect to a business application in order to receive the data and put it into the proper UN/EDIFACT format. In order to simplify this example, the purchase order information is provided in the form of a text file that is already in UN/EDIFACT format.

Figure 46 e*Xchange Scenario Data Flow

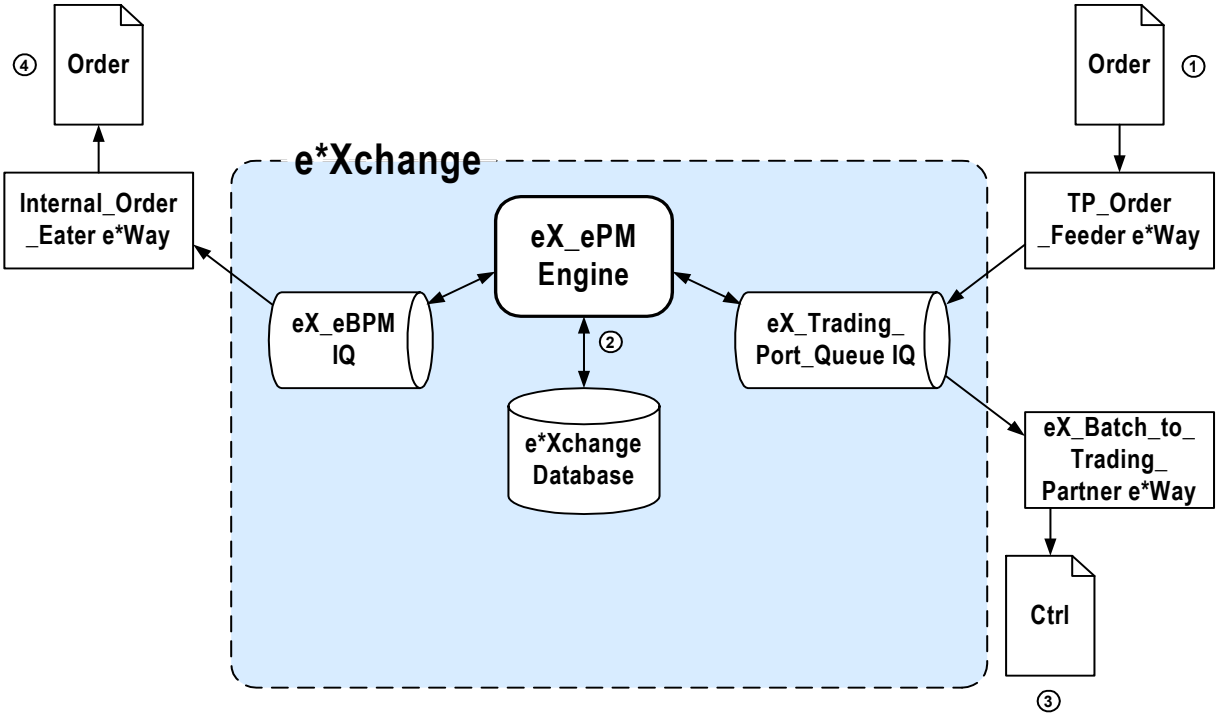


Figure 46 data flow description

① The **TP_Order_Feeder** e*Way picks up the order message and publishes it to the eX_trading_Port_Queue IQ.

② e*Xchange engine picks it up from the IQ, validates it, saves it to the database, and publishes two messages:

- Control message to the eX_Trading_Port_Queue IQ
- Order message to the eX_eBPM IQ.

③ eX_Batch_to_Trading_Partner e*Way sends out the control message to the trading partner.

④ Internal_Order_Eater e*Way picks up the message from the eX_eBPM IQ and sends it to the internal system.

10.2 Using the Implementation Sample

The components for this implementation are provided on your installation CD, and are located in **\setup\eXchange\sample\EDIFACT_SAMPLE_IMPLEMENTATION.zip**. Follow these steps to install the components:

1 Unzip the file to a local directory.

2 Install the e*Gate schema using one of the following commands. The schema name is user defined.

Note: *The default registry port number is 23001.*

A For Unix

```
sh install_edifact_po.sh <egate_registry_host_name> <schema_name>
<user_name> <password> <egate_registry_port_num>
```

B For Windows

```
install_edifact_po.bat <egate_registry_host_name> <schema_name>
<user_name> <password> <egate_registry_port_num>
```

3 Use the e*Xchange Import function to import **EDIFACT.exp** into e*Xchange Partner Manager.

4 Copy the data folder to the **<egate>** directory.

5 If e*Gate is not installed on your C drive, update the **Transport Component** file location as described in **"Step 5: Set up outbound B2B Protocol Information" on page 116**.

6 Configure the **eX_ePM** e*Way as described in **"Configure the eX_ePM e*Way" on page 128**.

The steps on the following pages describe how the components for this implementation were created. See **"Running the Scenario" on page 130** for instructions to run the implementation.

10.3 Create the Trading Partner Profiles

Trading partner profiles in e*Xchange act as repositories for the information necessary to send EDI messages back and forth between entities. They contain all of the information needed to properly envelope an Event and forward it to its correct destination.

When creating trading partner profiles, check your values carefully before saving or leaving a section/screen, because many values cannot be changed once they are committed to the database due to auditing restrictions. You can inactivate erroneous information and add the correct information under a different company, B2B Protocol, and so on.

Refer to the *e*Xchange Partner Manager User's Guide* for detailed assistance with the process of creating trading partner profiles.

Trading Partner Information Hierarchy

e*Xchange stores trading partner information at various levels. The process of creating a trading partner profile proceeds from the most general inclusive level, that of a company with which you do business, to the most specific information regarding an message that you wish to send (the message profile).

10.3.1 The Car Interiors Trading Partner

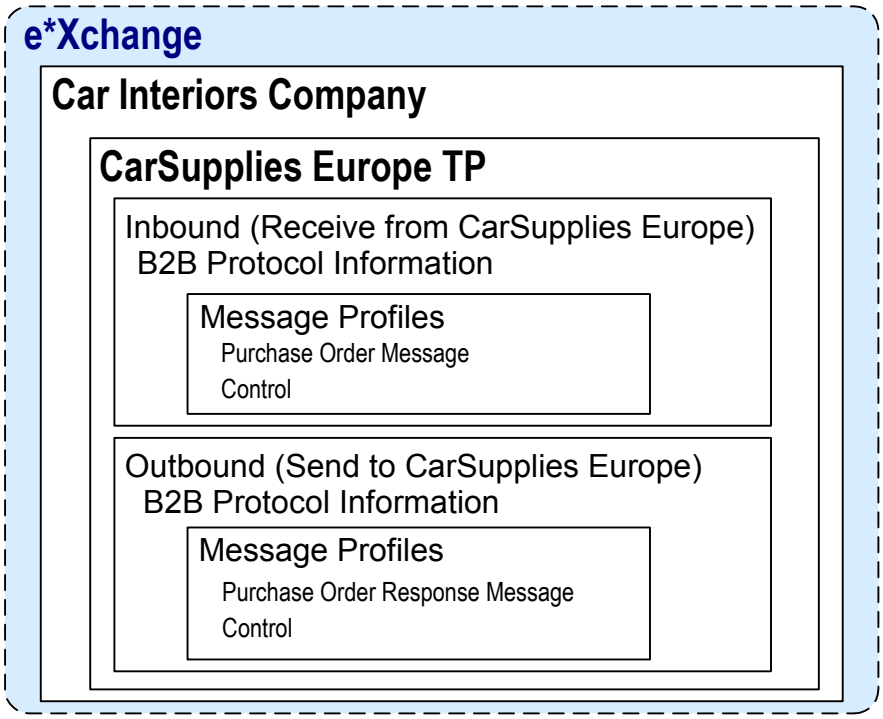
Car Interiors is a manufacturer of high quality car interiors that uses the UN/EDIFACT format to exchange business data with its customers. In our example we send a purchase order to Car Interiors.

The following procedure and accompanying tables were used to create the Car Interiors trading partner for this example.

Figure 47 shows an overview of the components that you need to create for this example, including:

- Company
- Trading Partner
- B2B Protocol Information
- Message Profiles

Figure 47 Car Interiors Overview



To configure the CarSupplies Europe trading partner profile you must follow the steps listed below:

- **Step 1: Create the Company** on page 112
- **Step 2: Create the Trading Partner** on page 113
- **Step 3: Set up the Inbound B2B Protocol Information** on page 113
- **Step 4: Create the Inbound Message Profiles** on page 114
- **Step 5: Set up outbound B2B Protocol Information** on page 116
- **Step 6: Create the Outbound Message Profiles** on page 116
- **Step 7: Configure Return Messages for Inbound** on page 119

Step 1: Create the Company

1. Log in to the e*Xchange Web interface.
2. From the **Main** page, click **Profile Management**.
3. From the **Company** page, click **New**.
4. In the **Company - adding** page, enter the **Company** name, “Car Interiors”.
5. Click **Next**.

   This saves your changes and returns to the **Company** page.

Note: *The security information is automatically configured for the current user.*

Step 2: Create the Trading Partner

1 From the **Company** page, ensure that "Car Interiors" is selected, and click **Continue: Trading Partner**.

2 From the **Trading Partner** page, click **New** to access the **Trading Partner - adding** page.

3 Enter the **Trading Partner Name**, "CarSupplies Europe".

4 Click **Next**.

This saves your changes and returns to the **Trading Partner** page.

The required security information defaults from the company level.

Step 3: Set up the Inbound B2B Protocol Information

To set up the inbound B2B Protocol Information

1 From the **Trading Partner** page, ensure that the "CarSupplies Europe" is selected, and click **Continue: B2B Protocol**.

2 From the **B2B Protocol** page, click **New** to access the **B2B Protocol - adding** page.

3 Enter the information listed in Table 19.

In an actual implementation, your local administrator can provide you with the B2B Protocol information. For an explanation of the B2B Protocol parameters, see the *e*Xchange Partner Manager User's Guide*.

Table 19 B2B Protocol Information

Parameter	Value
eBusiness Protocol	UN/EDIFACT
Version	4B
Direction	Inbound

4 Click **Next**, to save your changes and access the **General** section.

5 Enter the information listed in Table 20.

Table 20 B2B Protocol Information, General Page

Parameter	Value
Logical Name	TP_001
Status	Active
Communication Protocol	FTP(BATCH)

6 Click **Next**, to save your changes and access the **Transport Component** section.

7 No changes are required. Click **Next** to access the **Message Security** section.

8 No changes are required. Click **Finish** to save the information and return to the **B2B Protocol** page.

Step 4: Create the Inbound Message Profiles

For the purposes of this scenario, you must set up the following inbound message profiles:

- Purchase Order Message (EDF_ORDERSPurcOrdeMess_D99B)
- Control (EDF_CONTROL)

To set up the EDF_ORDERSPurcOrdeMess_D99B Order inbound message profile

1. From the **B2B Protocol** page, click **Continue: Message Profile**.
2. From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.
3. Enter the information listed in Table 21.

Note: *This table only lists the attributes required to make this scenario work.*

Table 21 General (EDF_ORDERSPurcOrdeMess_D99B)

Name	Value
Name	EDF_ORDERSPurcOrdeMess_D99B
Transfer Mode	Interactive
Validation Collaboration	EDF_ORDERSPurcOrdeMess_D99B

4. Click **Next** to access the **Interchange Control Envelope** section. Enter the information listed in Table 22.

Table 22 Interchange Control Envelope (EDF_ORDERSPurcOrdeMess_D99B)

Name	Value
Interchange Recipient Identifier	987654321
Interchange Recipient Identification Qualifier	1
Interchange Sender Identifier	123456789
Interchange Sender Identification Qualifier	1

5. Click **Next** to access the **Functional Group Envelope** section. Enter the information listed in Table 23.

Note: *This table only lists the attributes required to make this scenario work.*

Table 23 Functional Group Envelope (EDF_ORDERSPurcOrdeMess_D99B)

Name	Value
Application Receiver Identification Code	987654321
Application Sender Identification Code	123456789

6. Click **Next** to access the **Message Envelope** section.

7 In the **Message Type Identifier** window, type **ORDERS**.

8 Click **Next** to access the **Return Messages** section.

9 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

To set up the EDF_CONTROL inbound inner envelope

1 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

2 Enter the information listed in Table 24.

Note: *This table only lists the attributes required to make this scenario work.*

Table 24 General (EDF_CONTROL)

Name	Value
Name	EDF_CONTROL
Transfer Mode	Interactive
Validation Collaboration	EDF_CONTROL

3 Click **Next** to access the **Interchange Control Envelope** section. Enter the information listed in Table 25.

Table 25 Interchange Control Envelope (EDF_CONTROL)

Name	Value
Interchange Recipient Identifier	987654321
Interchange Recipient Identification Qualifier	1
Interchange Sender Identifier	123456789
Interchange Sender Identification Qualifier	1

4 Click **Next** to access the **Functional Group Envelope** section. Enter the information listed in Table 26.

Note: *This table only lists the attributes required to make this scenario work.*

Table 26 Functional Group Envelope (EDF_CONTROL)

Name	Value
Application Receiver Identification Code	987654321
Application Sender Identification Code	123456789

5 Click **Next** to access the **Message Envelope** section.

6 In the **Message Type Identifier** window, type **CONTRL**.

7 Click **Next** to access the **Return Messages** section.

8 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

Step 5: Set up outbound B2B Protocol Information

To set up the outbound B2B Protocol Information

As a shortcut, you can copy the Inbound B2B Protocol Information as a model for the Inbound B2B Protocol Information.

1 On the **B2B Protocol** page, select the UN/EDIFACT-4B-Inbound protocol that you created in **"To set up the inbound B2B Protocol Information" on page 113**.

2 Click **Copy**.

The **Copy Type** page appears.

3 Clear the **Include Sub-components** check box and then click **OK**.

The **B2B Protocol - copying** page appears.

4 In the **Direction** field, ensure that **Outbound** is selected.

5 Click **Next**.

The **B2B Protocol - copying, General** page appears.

6 No changes are needed: click **Next** to accept the values and access the **Transport Component** page.

7 In the **File Name** window, enter **<egate>\data\TP\eater\order_response_%d_%3#.dat**.

8 Click **Next** to accept the values and access the **Message Security** page.

9 No changes are required. Click **Finish** to save the information and return to the **B2B Protocol** page.

Step 6: Create the Outbound Message Profiles

For the purposes of this scenario, you must set up the following outbound message profiles:

- Purchase Order Response Message (EDF_ORDRSPPurcOrdeRespMess_D99B)
- Control (EDF_CONTROL)

To set up the EDF_ORDRSPPurcOrdeRespMess_D99B Order inbound message profile

1 From the **B2B Protocol** page, click **Continue: Message Profile**.

2 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

3 Enter the information listed in Table 27.

Note: *This table only lists the attributes required to make this scenario work.*

Table 27 General (EDF_ORDRSPPurcOrdeRespMess_D99B)

Name	Value
Name	EDF_ORDRSPPurcOrdeRespMess_D99B
Transfer Mode	Interactive
Validation Collaboration	EDF_ORDRSPPurcOrdeRespMess_D99B

4 Click **Next** to access the **Interchange Control Envelope** section. Enter the information listed in Table 28.

Table 28 Interchange Control Envelope (EDF_ORDRSPPurcOrdeRespMess_D99B)

Name	Value
Interchange Recipient Identifier	123456789
Interchange Recipient Identification Qualifier	1
Interchange Sender Identifier	987654321
Interchange Sender Identification Qualifier	1

5 Click **Next** to access the **Functional Group Envelope** section. Enter the information listed in Table 29.

Note: *This table only lists the attributes required to make this scenario work.*

Table 29 Functional Group Envelope (EDF_ORDERSPurcOrdeMess_D99B)

Name	Value
Application Receiver Identification Code	123456789
Application Sender Identification Code	987654321

6 Click **Next** to access the **Message Envelope** section.

7 In the **Message Type Identifier** window, type **ORDRSP**.

8 Click **Next** to access the **Return Messages** section.

9 Select the return message (select the **Include** check box), and enter the values, as shown in Table 30.

Table 30 Return Message Values: Outbound

Name	Response Time	Period	# Retries
EDF_CONTROL	2	Minutes	2

10 Click **Finish** to save the information and return to the **Message Profile** page.

To set up the EDF_CONTROL inbound inner envelope

1 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

2 Enter the information listed in Table 31.

Note: *This table only lists the attributes required to make this scenario work.*

Table 31 General (EDF_CONTROL)

Name	Value
Name	EDF_CONTROL
Transfer Mode	Interactive
Validation Collaboration	EDF_CONTROL

3 Click **Next** to access the **Interchange Control Envelope** section. Enter the information listed in Table 32.

Table 32 Interchange Control Envelope (EDF_CONTROL)

Name	Value
Interchange Recipient Identifier	123456789
Interchange Recipient Identification Qualifier	1
Interchange Sender Identifier	987654321
Interchange Sender Identification Qualifier	1

4 Click **Next** to access the **Functional Group Envelope** section. Enter the information listed in Table 33.

Note: *This table only lists the attributes required to make this scenario work.*

Table 33 Functional Group Envelope (EDF_CONTROL)

Name	Value
Application Receiver Identification Code	123456789
Application Sender Identification Code	987654321

5 Click **Next** to access the **Message Envelope** section.

6 In the **Message Type Identifier** window, type **CONTRL**.

7 Click **Next** to access the **Return Messages** section.

8 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

Step 7: Configure Return Messages for Inbound

To set up the Return Inner Envelope values for Inbound

Once you have set up inbound and outbound message profiles, you can specify return messages.

1. From the **B2B Profile** page, select **UN/EDIFACT-4B-Inbound**.
2. Click **Continue: Message Profile**.
3. From the **Message Profile** page, select **EDF_ORDERSPurcOdeMess_D99B** from the drop-down list.
4. Click the **Return Messages** link to access the **Return Messages** section.
5. Click **Edit**.
6. Select the return messages (select the check boxes), and enter the values, as shown in Table 34.

Table 34 Return Message Values: Inbound

Name	Response Time	Period	# Retries
EDF_ORDRSPPurcOrdeRespMess_D99B	1	Day	0
EDF_CONTROL	2	Minutes	0

7. Click **Apply** to save the information and return to the **Message Profile** page.
8. Click **OK**.

10.4 Clone the eXSchema

The supplied schema named eXSchema contains the components required to run e*Xchange. Make a copy of this schema and then configure the copy for this implementation.

To make a copy of eXSchema

1. Open eXSchema in the e*Gate Enterprise Manager GUI.
2. Export eXSchema.
3. Create a new schema named **EDIFACT** using the exported file.

10.5 Configure the TP_Order_Feeder e*Way

The component (e*Way or BOB) that feeds data into e*Xchange must put the data into the appropriate business protocol format. It must also populate the required fields in the e*Xchange Event that is processed by e*Xchange.

This component is entirely user-defined and must be added to the EDIFACT schema. The type of component to use depends on whether a connection to a system outside e*Gate must be made, and if so, what type of system. Typically, this component is an e*Way that connects to a business application such as SAP that sends out electronic messages. These messages may or may not be in the format required by the trading partner to which they are being sent. If the data is not in the correct format, the e*Way must translate the data into the required format before it is sent to the e*Xchange system for enveloping and forwarding to the trading partner.

The e*Xchange TP_Order_Feeder e*Way

The e*Xchange example simulates the publication of an electronic purchase order from a trading partner. This file, which is already in UN/EDIFACT format, is picked up by a file e*Way and moved into the e*Xchange system.

Configuration Steps

Follow these steps to configure the TP_Order_Feeder e*Way.

1. Create and configure the e*Way.
2. Create the ETDs.
3. Create the Collaboration.

10.5.1 Step 1: Create and configure the TP_Order_Feeder e*Way

1. Create an e*Way called **TP_Order_Feeder**.
2. In the **e*Way Properties** dialog box, in the **Executable file** area of the **General** tab, browse for **stcewfile.exe.**
3. In the **e*Way Properties** dialog box, in the **Configuration file** area of the **General** tab, click **New**.
4. Configure the **TP_Order_Feeder** e*Way parameters using Table 35.

Table 35 TP_Order_Feeder e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\Data\TP\feeder
	MultipleRecordsPerFile	NO
	(All others)	(Default)
Performance Testing	(All)	(Default)

5. When finished editing the e*Way configuration file, save your work and close the e*Way editor.
6. Click **OK** to close the **e*Way Properties** dialog box.

10.5.2 Step 2: Create the TP_Order_Feeder ETDs

In the present example, since the data is already in standard UN/EDIFACT format for a purchase order, you can bring in the Event without parsing it. To do this, all that is required is an ETD with a root node.

To create the root ETD

1. Create a new ETD called **root.ssc**. In the **Type** box, select Delimited, and select **Other** from the drop-down list.
2. Add a single node to the structure. The ETD is shown in Figure 48.

Figure 48 root.ssc Event Type Definition

3. Save the ETD.

10.5.3 Step 3: Create the TP_Order_Feeder Collaboration

The **TP_Order_Feeder** Collaboration must prepare the data coming into the e*Xchange system. How complicated this task is depends on the state of the data before the **TP_Order_Feeder** Collaboration processes it.

The **TP_Order_Feeder** Collaboration must do the following:

- convert the data to base 64 encoding
- populate the required nodes in the e*Xchange Event sent to e*Xchange for processing

Convert the Event to Base 64 Encoding

The **TP_Order_Feeder** Collaboration must ensure that the data going into e*Xchange doesn't include any characters that will cause problems for the XML structure of the standard e*Xchange Event (for example, characters that are the same as the XML control characters). This is done by converting the entire EDI message to base 64 encoding using the Monk function **raw->base64**, before copying it to the payload node of the **eX_Standard_Event** ETD.

Populate the Required e*Xchange Nodes

In addition to copying the base 64 encoded message to the payload node of the TP_EVENT portion of the e*Xchange standard Event, you must provide e*Xchange with tracking information about the EDI message.

e*Xchange Tracking Information

e*Xchange needs to know certain things about an EDI message before it can process it. The **TP_Order_Feeder** Collaboration must supply this information by populating

certain required nodes in the Event that is sent to e*Xchange. At a minimum you must tell e*Xchange:

- Direction (inbound or outbound)
- Partner Name (logical name from the outer envelope in e*Xchange)

All of these requirements can be met by copying the appropriate information to the corresponding nodes in the TP section of the e*Xchange ETD (eX_Standard_Event.ssc).

The **TP_EVENT.CT.DSN.DS.Direction.CT.DSN.DS.Data** node must contain the direction of the Event: "O" for outbound to the trading partner or "I" for inbound from a trading partner.

The **TP_EVENT.CT.DSN.DS.PartnerName.CT.DSN.DS.Data** node must contain the logical name (case-sensitive) of the trading partner defined in the **B2B Protocol**, **General** page.

The e*Xchange Payload

In addition to the tracking information, the **TP_EVENT.CT.DSN.DS.Payload.CT.DSN.DS.Data** node must be filled with the entire base 64 encoded EDI message.

The e*Xchange TP_Order_Feeder CRS

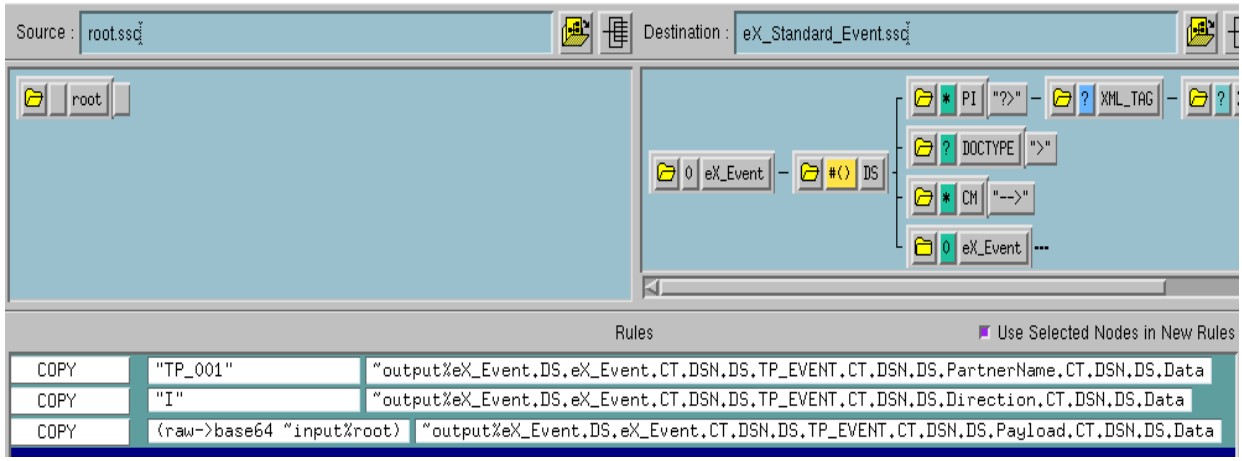
The **TP_Order_Feeder.tsc** CRS does the following:

- Converts the UN/EDIFACT message to base 64 encoding, and copies it to the Payload node of the TP_EVENT section of the e*Xchange standard Event.
- Copies "I" for outbound to the direction node of the TP_EVENT section.
- Copies the trading partner logical name "TP_001" to the PartnerName node of the TP_EVENT section.

To create and configure the TP_Order_Feeder Collaboration Rule Script

1. Open the Collaboration Editor.
2. Create a new Collaboration Rules script named **TP_Order_Feeder.tsc**. The Source Event Type Definition is **root.ssc**. The Destination Event Type Definition is **eX_Standard_Event.ssc**.
3. Add the rules shown in Figure 49.

Figure 49 TP_Order_Feeder.tsc



TP_Order_Feeder Collaboration Properties Setup

Once the CRS has been created, you must set up the Collaboration and Collaboration Rules Properties for the **TP_Order_Feeder** Component in the Enterprise Manager GUI.

To create and configure the TP_Order_Feeder Collaboration Rule

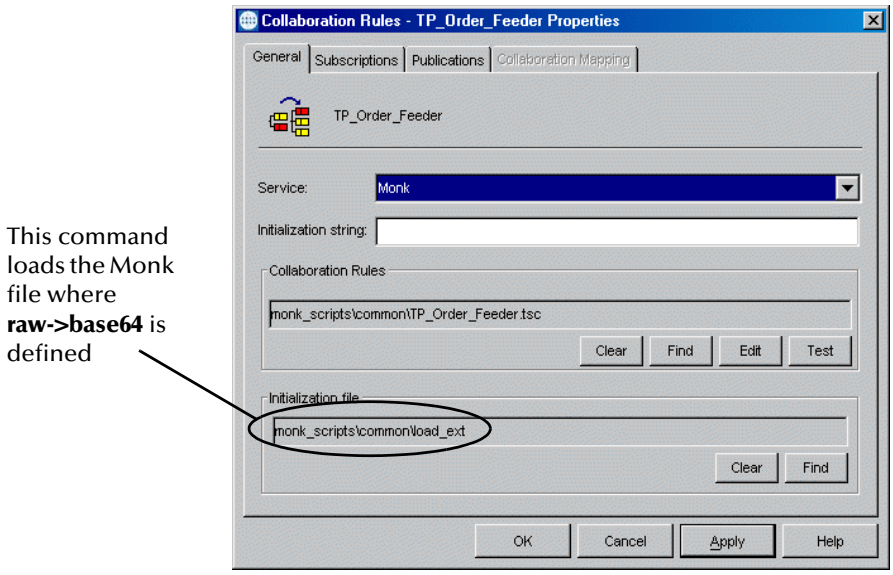
1. Create a new Collaboration Rule named **TP_Order_Feeder**.
2. From **TP_Order_Feeder** Collaboration Rule properties, select the **General** tab. Configure as shown in Table 36.

Table 36 TP_Order_Feeder CR configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	TP_Order_Feeder
Initialization File	monk_scripts\common\load_ext

Important: *To use the Monk function* ***raw->base64****, you must make sure the file containing this function has been loaded.*

Figure 50 TP_Order_Feeder Collaboration Rules Properties Dialog Box



3 Select the **Subscriptions** tab. Select **eX_External_Evt** and move it to the right pane.

4 Select the **Publications** tab. Select **eX_from_Trading_Partner** and move it to the right pane.

To create and configure the TP_Order_Feeder Collaboration

1 Select the **TP_Order_Feeder** e*Way.

2 Create a new Collaboration named **TP_Order_Feeder**.

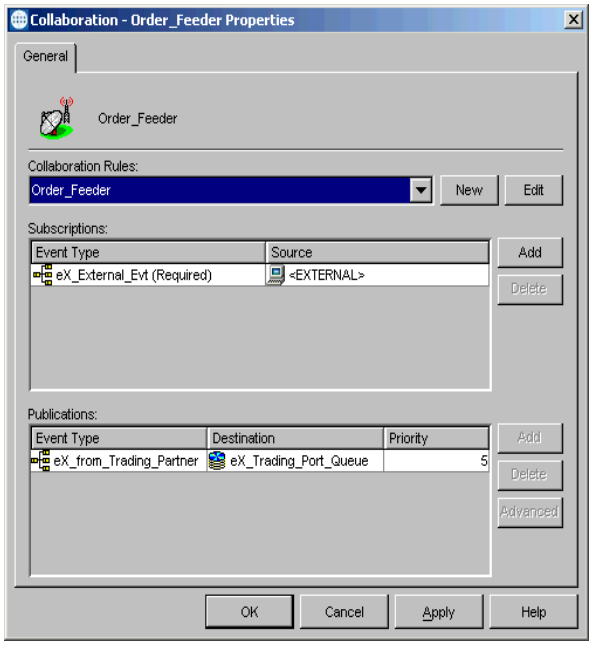
3 Configure the TP_Order_Feeder Collaboration properties using Table 37.

Table 37 TP_Order_Feeder Collaboration configuration

Section	Value
Collaboration Rules	TP_Order_Feeder
Subscriptions	Event Type: eX_External_Evt Source: <EXTERNAL>
Publications	Event Type: eX_from_Trading_Partner Destination: eX_Trading_Port_Queue

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 51.

Figure 51 TP_Order_Feeder Collaboration Properties



10.6 Configure the Internal_Order_Eater e*Way

The component (e*Way or BOB) sends the message to the internal system.

The e*Xchange Internal_Order_Eater e*Way

The e*Xchange example simulates the publication of the message to the internal system.

Configuration Steps

Follow these steps to configure the Internal_Order_Eater e*Way.

1. Create the configuration file.
2. Create the ETDs.
3. Create the Collaboration.

10.6.1 Step 1: Create and Configure the Internal_Order_Eater e*Way

1. Create an e*Way called **Internal_Order_Eater**.
2. In the **e*Way Properties** dialog box **General** tab, in the **Executable file** area browse for **stcewfile.exe.**

3 In the **e*Way Properties** dialog box **General** tab, in the **Configuration file** area click **New**.

4 Configure the **Internal_Order_Eater** e*Way parameters using Table 38.

Table 38 Internal_Order_Eater e*Way Parameters

Screen	Parameter	Setting
General Settings	AllowIncoming	NO
	AllowOutgoing	YES
Outbound (send) settings	OutputDirectory	<eGate>\data\internal\eater
	OutputFileName	output_order%d.dat
	(All others)	(Default)
Poller (inbound) settings	(All)	(Default)
Performance Testing	(All)	(Default)

5 When finished editing the e*Way configuration file, save your work and close the e*Way editor.

6 Click **OK** to close the **e*Way Properties** dialog box.

10.6.2 Step 2: Create the Internal_Order_Eater Collaboration

The **Internal_Order_Eater** Collaboration must prepare the data leaving the e*Xchange system. How complicated this task is depends on the state of the data before the **Internal_Order_Eater** Collaboration processes it.

The **Internal_Order_Eater** Collaboration must do the following:

- put the data into the appropriate EDI format
- convert the data to raw data

The e*Xchange Internal_Order_Eater CRS

The **Internal_Order_Eater.tsc** CRS is used to convert the UN/EDIFACT message to raw data, and copies it from the Payload node of the TP_EVENT section of the e*Xchange standard Event to the output ETD.

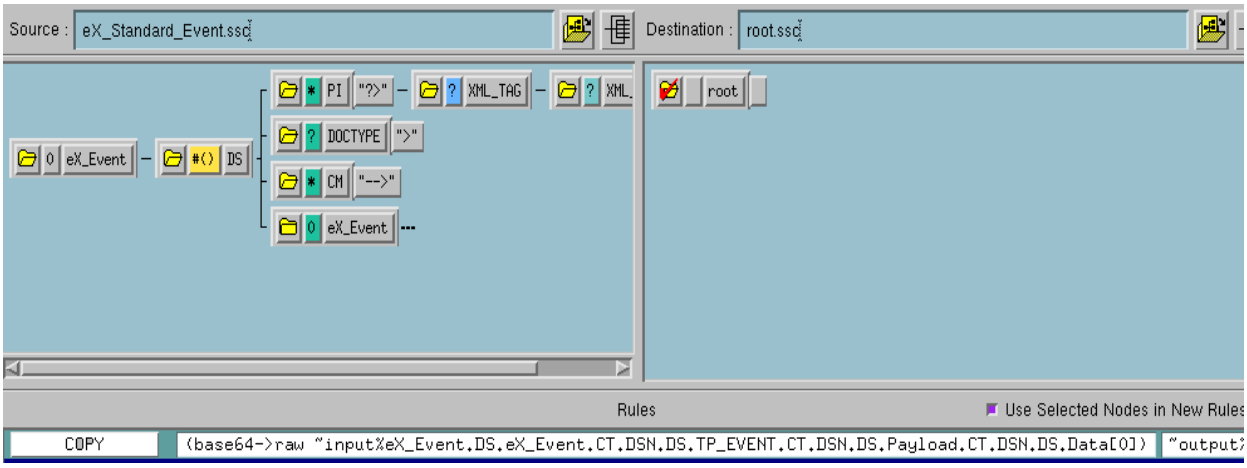
To create and configure the Internal_Order_Eater Collaboration Rule Script

1 Open the Collaboration Editor.

2 Create a new Collaboration Rules script named **Internal_Order_Eater.tsc**. The Source Event Type Definition is **eX_Standard_Event.ssc**. The Destination Event Type Definition is **root.ssc**.

3 Add the rule shown in Figure 52.

Figure 52 Internal_Order_Eater.tsc



Internal_Order_Eater Collaboration Properties Setup

Once the CRS has been created, you must set up the Collaboration and Collaboration Rules Properties for the **Internal_Order_Eater** Component in the Enterprise Manager GUI.

To create and configure the Internal_Order_Eater Collaboration Rule

1. Create a new Collaboration Rule named **Internal_Order_Eater**.
2. From Internal_Order_Eater Collaboration Rule properties, select the **General** tab. Configure as shown in Table 39.

Table 39 Internal_Order_Eater CR configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	Internal_Order_Eater
Initialization File	monk_scripts\common\load_ext

Important: *To use the Monk function* ***base64->raw****, you must make sure the file containing this function has been loaded.*

3. Select the **Subscriptions** tab. Select **eX_to_eBPM** and move to the right pane.
4. Select the **Publications** tab. Select **eX_External_Evt** and move to the right pane.

To create and configure the Internal_Order_Eater Collaboration

1. Select the **Internal_Order_Eater** e*Way.
2. Create a new Collaboration named **Internal_Order_Eater**.
3. Configure the Internal_Order_Eater Collaboration properties using Table 40.

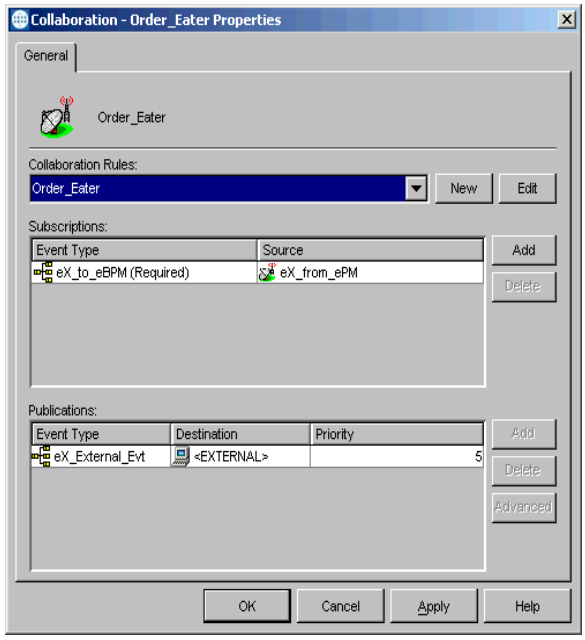
Table 40 Internal_Order_Eater Collaboration configuration

Section	Value
Collaboration Rule	Internal_Order_Eater

Section	Value
Subscriptions	Event Type: eX_to_eBPM Source: eX_from_ePM
Publications	Event Type: eX_External_Evt Destination: <EXTERNAL>

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 53.

Figure 53 Internal_Order_Eater Collaboration Properties



10.7 Configure the eX_ePM e*Way

The **eX_ePM** e*Way requires only minimal configuration. You must give it the logon information for the e*Xchange database.

To configure the eX_ePM configuration file

1. In the **eX_ePM** e*Way properties, select the **General** tab.
2. In the **Configuration File** area, click **Edit**.
3. Configure the parameters as shown in Table 41.

Table 41 eX_ePM e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)

Table 41 eX_ePM e*Way Parameters

Screen	Parameter	Setting
Communication Setup	(All)	(Default)
Monk Configuration	(All)	(Default)
Database Setup	Database Name	(service name of the e*Xchange database)
	User name	ex_admin
	Password	ex_admin
	(All others)	(Default)

10.8 Editing the Data Files

Before running the scenario, you must make sure that the unique ID in the input file matches that in the output file, and that both files have the expected filename and extension.

Knowing how to set these values also gives you the capability to reset the unique ID to an appropriate new value so that you can run the scenario multiple times.

To ensure the unique ID in both files matches

1. Open up the file **ORDERS.~in** (in the <**egate**>**\data\TP\feeder** folder) in a text editor such as Notepad or Wordpad.
2. Search for the following string, which is the unique ID in the files provided:

   ```
   ORDERS_000000008
   ```

3. Replace that string with the following string:

   ```
   ORDERS_000000001
   ```

4. Save and close.
5. Open up the file order_response.~in (in the <**egate**>**\data\internal\feeder** folder) in a text editor such as Notepad or Wordpad.
6. Repeat steps 2 through 4 for this file.

Note: *The last nine digits of the unique ID indicate that this is the first instance for this date. For a second and subsequent running of this scenario, increment the last three digits: 000000002, 000000003, and so forth. In each case, make sure that the value is the same in both files.*

To set the file names correctly

1. In <**egate**>**\data\TP\feeder**, change the name of the **orders.~in** file to **orders.fin**.
2. In <**egate**>**\data\internal\feeder**, change the name of the **order_response.~in** file to **order_response.fin**.

That completes the data setup. The next step is to run the scenario.

Conditional—to reset the file names

Once you have your schema running, you can run the file again by performing the following steps, in sequence:

1. Increment the last nine digits of the control number by 1. For example, if the control number is **ORDERS_000000001**, change it to **ORDERS_000000002**. Make sure that both files match.
2. Change the extension from .**~in** to .**fin** in both files.

10.9 Running the Scenario

There are two parts to running the scenario:

1. Processing the purchase order message received from the trading partner
2. Sending the response message back to the trading partner

To process the purchase order message

1. Start the Control Broker. At the command line, enter:

```
stccb.exe -rh localhost -rs <schema_name> -ln localhost_cb -un
Administrator -up STC
```

2. Open the e*Gate Monitor. Select the UN/EDIFACT schema.

Note: *If you have imported the sample schema then all the e*Ways are set to start automatically.*

3. Start the **TP_Order_Feeder** e*Way

   This e*Way retrieves the incoming message and sends it to e*Xchange.

4. Rename **<eGate>\data\TP\Feeder\ORDERS.~in** to **ORDERS.fin**.
5. Look in the **<egate>\data\TP\feeder** folder. The file name changes from **orders.fin** to **orders.~in** as the file is picked up.
6. Start the **eX_Batch_to_Trading_Partner** e*Way

   This e*Way sends the control message back to the trading partner.

7. Start the **Internal_Order_Eater** e*Way

   This e*Way sends the message to the internal system.

That completes the first part of the exercise. You can view the results in Message Tracking, in e*Xchange Partner Manager.

Viewing the Results in Message Tracking

You can view the results of the message processing by using the Message Tracking feature of e*Xchange.

Message Tracking shows two entries for the incoming message. This is because a control message is sent out immediately, and a response message will be sent out later. These two responses to the trading partner are tracked separately.

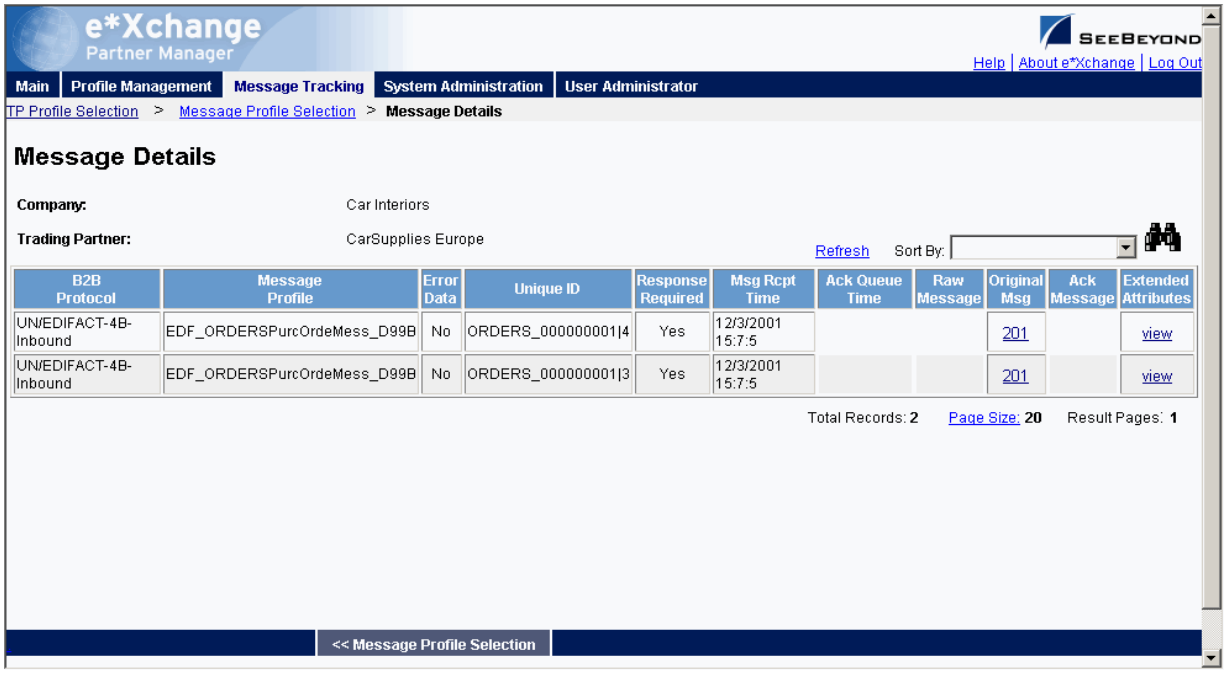
To view the inbound message in Message Tracking

1. From the e*Xchange Web interface, **Main** page, select **Message Tracking**.

   The **TP Profile Selection** page appears.
2. In the **Company Profile** field, select **Car Interiors**.
3. In the **Trading Partner Profile** field, select **CarSupplies Europe**.
4. In the **eBusiness Protocol** field, select **UN/EDIFACT**.
5. In the **Direction** field, select **Inbound**.
6. Click the **Message Profile Selection**.
7. Select the **EDF_ORDERSPurcOrdeMess_D99B** message.
8. Click the **Message Details** link to view the resulting list.

   The results are shown in Figure 54.

Figure 54 Message Tracking: Inbound



As shown in Figure 54, e*Xchange records two entries for the message. The top entry is for the original message, for which a response message will be sent. The second entry is for the control message.

For one entry, the **Ack Message** column has a link to the message information. Click it to view the acknowledgment message.

Figure 55 Control Message viewed in Message Tracking

Later, when the response message is sent out, you will be able to view it in Message Tracking. For the moment, the **Ack Message** column is not showing a link for the other message, since the response has not been sent out yet.

If you look in the **<egate>\data\TP\eater** folder, you will see the following output file:

- output1.dat—control message sent in response to the original message.

10.10 Sending the Response

This section builds on the UN/EDIFACT implementation example. You are now simulating sending a response message to the Trading Partner and e*Xchange receiving a control message back from the Trading Partner after you send out the response message.

Figure 56 e*Xchange Scenario Data Flow

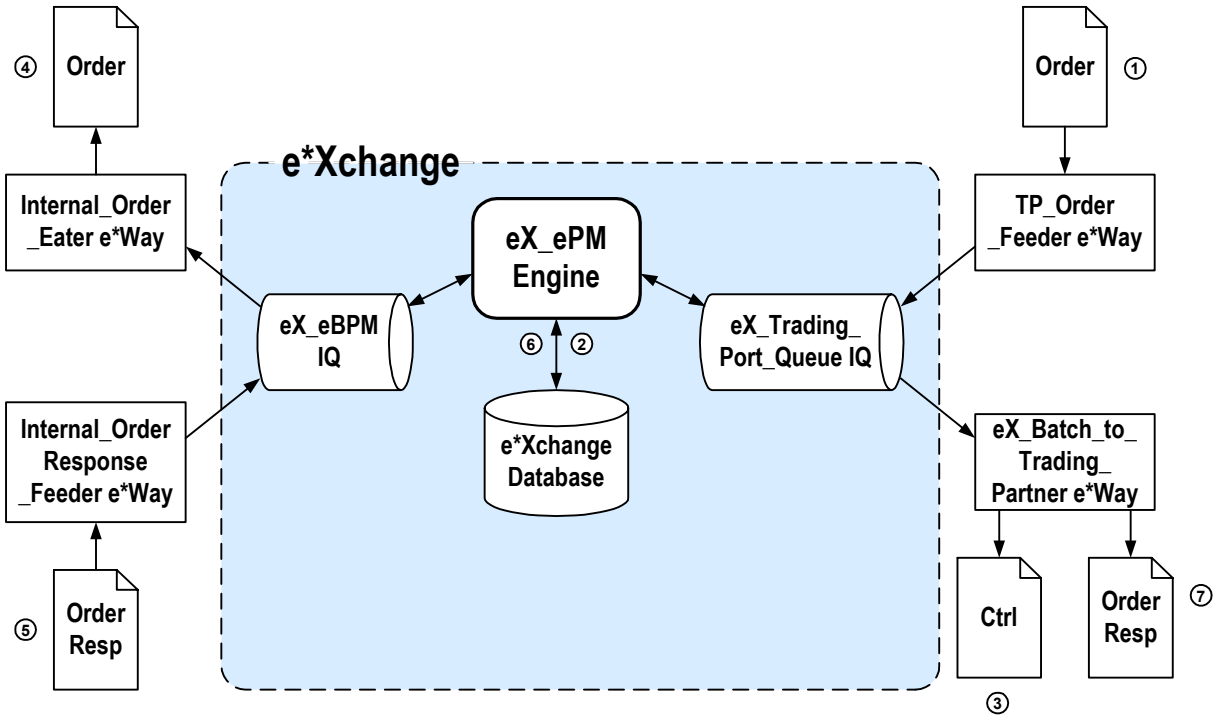


Figure 46 data flow description

① The **TP_Order_Feeder** e*Way picks up the order message and publishes it to the eX_trading_Port_Queue IQ.

② e*Xchange engine picks up from the IQ, validates it, saves it to the database, and publishes two messages:

- Control message to the eX_Trading_Port_Queue IQ
- Order message to the eX_eBPM IQ.

③ eX_Batch_to_Trading_Partner e*Way sends out the control message to the trading partner.

④ Internal_Order_Eater e*Way picks up the message from the eX_eBPM IQ and sends it to the internal system.

⑤ Internal_OrderReponse_Feeder e*Way picks up the response message and publishes it to the eX_eBPM IQ.

⑥ e*Xchange engine picks up the message from the eX_eBPM IQ, validates it, envelopes it, and saves it to the database, and publishes it to the eX_Trading_Port_Queue IQ.

⑦ eX_Batch_to_Trading_Partner e*Way picks up the message from the eX_Trading_Port_Queue IQ and sends it to the trading partner.

10.11 Configure the Internal_OrderResponse_Feeder e*Way

The component (e*Way or BOB) that feeds data into e*Xchange must put the data into the appropriate business protocol format. It must also populate the required fields in the e*Xchange Event that is processed by e*Xchange.

This component is entirely user-defined and must be added to the eXSchema. The type of component to use depends on whether a connection to a system outside e*Gate must be made, and if so, what type of system. Typically, this component is an e*Way that connects to a business application such as SAP that sends out electronic messages. These messages may or may not be in the format required by the trading partner to which they are being sent. If the data is not in the correct format, the e*Way must translate the data into the required format before it is sent to the e*Xchange system for enveloping and forwarding to the trading partner.

The e*Xchange Internal_OrderResponse_Feeder e*Way

The e*Xchange example simulates sending the response message from the internal system.

Configuration Steps

Follow these steps to configure Internal_OrderResponse_Feeder e*Way.

1. Create the configuration file.
2. Create the ETDs.
3. Create the Collaboration.

10.11.1 Step 1: Create and Configure the Internal_OrderResponse_Feeder e*Way

1. Create a new e*Way named **Internal_OrderResponse_Feeder**.
2. In the **Executable file** area of the **General** tab, in the **e*Way Properties** dialog box, browse for **stcewfile.exe.**
3. In the **Configuration file** area of the **General** tab, in the **e*Way Properties** dialog box click **New**.
4. Configure the **Internal_OrderResponse_Feeder** e*Way parameters using Table 42.

Table 42 Internal_OrderResponse_Feeder e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\data\internal\feeder
	MultipleRecordsPerFile	NO
	(All others)	(Default)

Table 42 Internal_OrderResponse_Feeder e*Way Parameters

Screen	Parameter	Setting
Performance Testing	(All)	(Default)

5 When finished editing the e*Way configuration file, save your work and close the e*Way editor.

6 Click **OK** to close the **e*Way Properties** dialog box.

10.11.2 Step 2: Create the Internal_OrderResponse_Feeder Collaboration

The **Internal_OrderResponse_Feeder** Collaboration must prepare the data coming into the e*Xchange system. How complicated this task is depends on the state of the data before the **Internal_OrderResponse_Feeder** Collaboration processes it.

The **Internal_OrderResponse_Feeder** Collaboration must do the following:

- put the data into the appropriate EDI format
- convert the data to base 64 encoding
- populate the required nodes in the e*Xchange Event sent to e*Xchange for processing

The e*Xchange Internal_OrderResponse_Feeder CRS

The **Internal_OrderResponse_Feeder.tsc** CRS does the following:

- Converts the UN/EDIFACT message to base 64 encoding, and copies it to the Payload node of the TP_EVENT section of the e*Xchange standard Event.
- Copies "O" for outbound to the direction node of the TP_EVENT section.
- Copies the trading partner logical name "TP_001" to the PartnerName node of the TP_EVENT section.

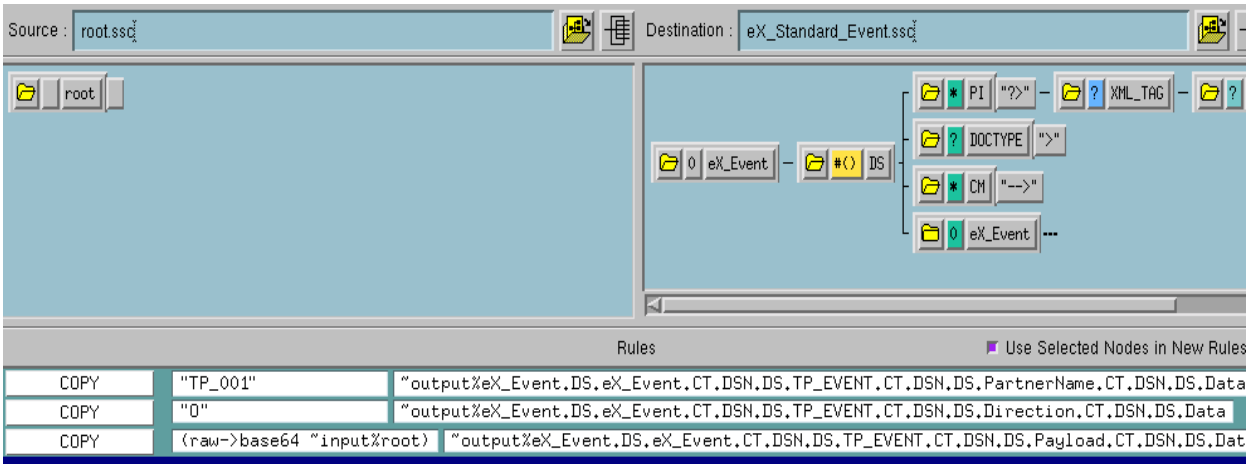
To create and configure the Internal_OrderResponse_Feeder Collaboration Rule Script

1 Open the Collaboration Editor.

2 Create a new Collaboration Rules script named **Internal_OrderResponse_Feeder.tsc**. The Source Event Type Definition is **root.ssc**. The Destination Event Type Definition is **eX_Standard_Event.ssc**.

3 Add the rules shown in Figure 57.

Figure 57 Internal_OrderResponse_Feeder.tsc



Internal_OrderResponse_Feeder Collaboration Properties Setup

Once the CRS has been created, you must set up the Collaboration and Collaboration Rules Properties for the **Internal_OrderResponse_Feeder** Component in the Enterprise Manager GUI.

To create and configure the Internal_OrderResponse_Feeder Collaboration Rule

1 Create a new Collaboration Rule named **Internal_OrderResponse_Feeder**.

2 From Internal_OrderResponse_Feeder Collaboration Rule properties, select the **General** tab. Configure as shown in Table 43.

Table 43 Internal_OrderResponse_Feeder CR configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	Internal_OrderResponse_Feeder
Initialization File	monk_scripts\common\load_ext

Important: *To use the Monk function **raw->base64**, you must make sure the file containing this function has been loaded.*

3 Select the Subscriptions tab. Select **eX_External_Evt** and move it to the right pane.

4 Select the Publications tab. Select **eX_to_ePM** and move it to the right pane.

To create and configure the Internal_OrderResponse_Feeder Collaboration

1 Select the **Internal_OrderResponse_Feeder** e*Way.

2 Create a new Collaboration named **Internal_OrderResponse_Feeder**.

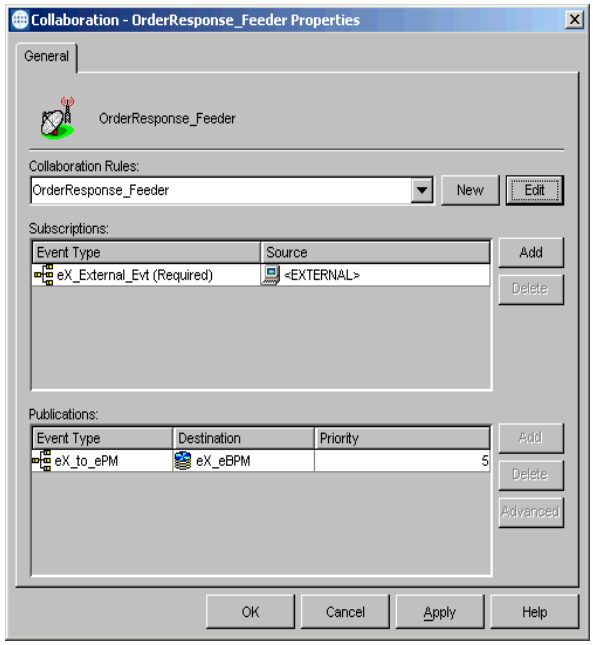
3 Configure the **Internal_OrderResponse_Feeder** Collaboration properties using Table 44.

Table 44 Internal_OrderResponse_Feeder Collaboration configuration

Section	Value
Collaboration Rule	Internal_OrderResponse_Feeder
Subscriptions	Event Type: eX_External_Evt Source: <EXTERNAL>
Publications	Event Type: eX_to_ePM Destination: eX_eBPM

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 58.

Figure 58 Internal_OrderResponse_Feeder Collaboration Properties



10.11.3 Sending and Viewing the Response Message

The next step is to send the response message.

The input file for the response message is the **<egate>\data\internal\feeder\order_response.~in** file. Instructions for preparing this file for running the first time were given in **"Editing the Data Files" on page 129**.

To send the response message:

1. In the e*Gate Monitor, start the Internal_OrderResponse_Feeder e*Way.
2. Look in the **<egate>\data\internal\feeder** folder. The file name changes from **order_response.fin** to **order_response.~in** as the file is picked up.
3. Look in the **<egate>\data\TP\eater** folder. The following output file has been created:
   - order_response#.dat—message sent out for the response.

That completes the second part of the exercise. You can view the results in Message Tracking.

Viewing the Results in Message Tracking

You can view the results of the message processing in Message Tracking.

To view the association of the response message to the original inbound message in Message Tracking

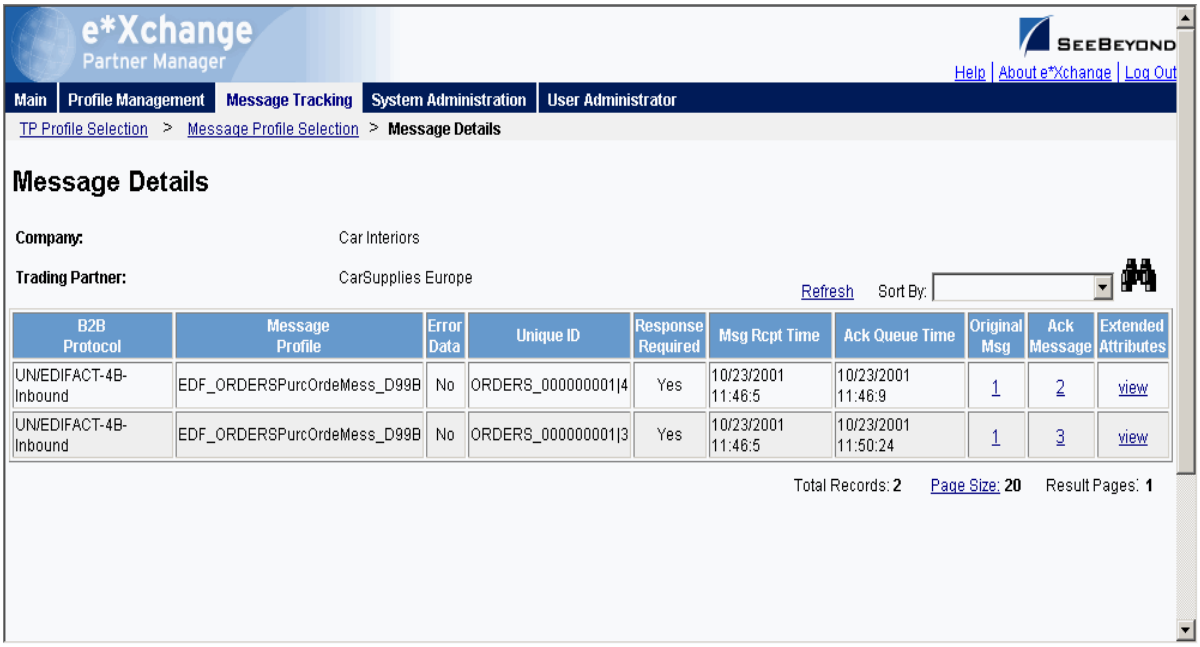
1. From the e*Xchange Web interface, **Main** page, select **Message Tracking**.

   The **TP Profile Selection** page appears.
2. In the **Company Profile** field, select **Car Interiors**.
3. In the **Trading Partner Profile** field, select **CarSupplies Europe**.
4. In the **eBusiness Protocol** field, select **UN/EDIFACT**.
5. In the **Direction** field, select **Inbound**.
6. Click the **Message Profile Selection**.
7. Select the **EDF_ORDERSPurcOrdeMess_D99B** message.
8. Click the **Message Details** link to view the resulting list.

   Notice that both entries now have responses available for viewing: one is the control message, the other is the full response message.

   The results are shown in Figure 59.

Figure 59 Message Tracking: Inbound with response



10.12 Receiving a Control Message from the Trading Partner

10.12.1 Editing the Data File

Before running the scenario, you must make some changes to your message files.

Since the control numbers in the message that comes in must match the control numbers in the message you sent out to your trading partner, you must manually update the control numbers.

There are three control numbers, one in each of three segments:

- Interchange response (UCI) segment
- Group response (UCF) segment
- Message/package response (UCM) segment

For the purposes of this scenario, the same number is used for each of these. Because of this, you can copy one number from the UNB (interchange header) segment of the outgoing response message and paste it in three places in the file that will serve for the incoming control message.

However, you must first have the outgoing response message available. Run the first two parts of the previous scenario again so that e*Xchange sends out the response message to the trading partner.

10.12.2 Preparing the Data File

The next step is to prepare your data file for running this scenario. The steps are:

- Copy the control numbers from the outgoing response message to the UCF, UCI and UCM segments of the incoming control message.
- Update the unique control numbers in the UNB and UNZ segments of the incoming control message.

10.12.3 Copying the Response Control Numbers

Next, you must copy the control number from the response message that e*Xchange sends out.

To copy the control number:

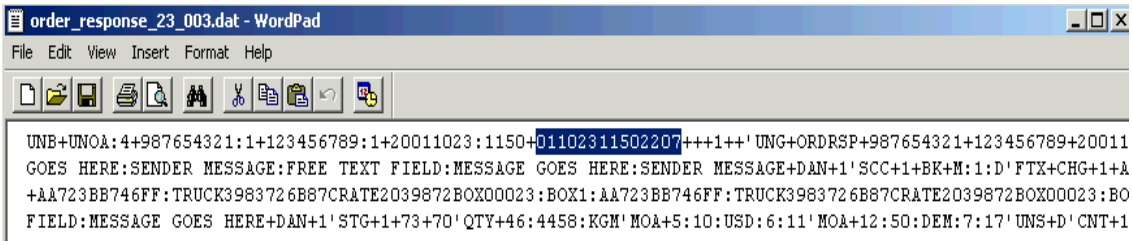
1. Go to **<egate>\data\TP\eater**.
2. Locate the output file that represents the response message that was just sent out. It will look something like the file shown in Figure 60.

Note: *The response message has a larger file size than the control message. Also, since the control message is normally sent first, the response message is likely to be output2.dat. However, this depends on what files were already there.*

3 Copy the Interchange Control Reference.

It is the fifth element of the UNB segment, as shown in Figure 60.

Figure 60 Response Message



4 Close the message file.

5 Open up the file **<egate>\data\TP\feeder\INB_CONTROL.~in**.

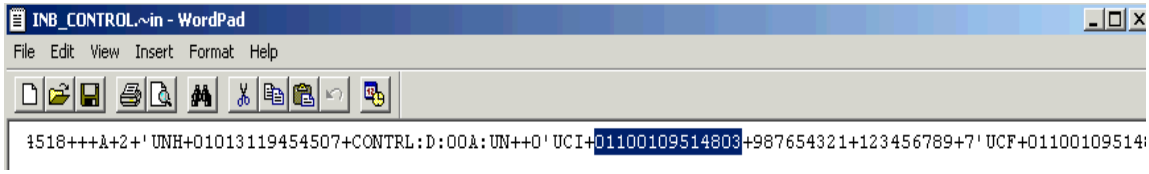
This is the inbound control message.

6 Search for **UCI**.

7 Change the next element (between + signs) to the string that you copied, as shown in Figure 61.

This updates the interchange response control number.

Figure 61 Inbound Control Message



8 Search for **UCF**.

9 Change the next element (between + signs) to the string that you copied.

This updates the functional group response control number.

10 Search for **UCM**.

11 Change the next element (between + signs) to the string that you copied.

This updates the message/package response control number.

12 Save the changes, but leave the file open.

10.12.4 Incrementing the UNB/UNZ Control Numbers

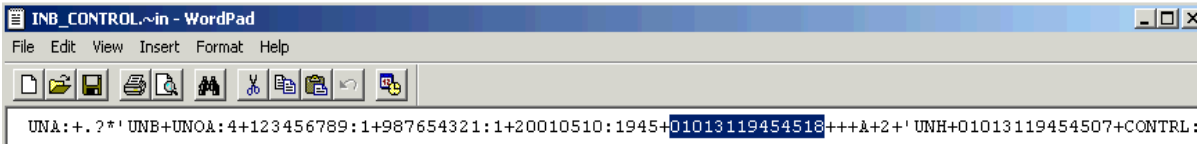
You must also increment the UNB and UNZ control numbers in the incoming control message to ensure they are unique. Make sure both control numbers are set to the same value.

To increment the UNB/UNZ control numbers:

1 In **<egate>\data\TP\feeder\inb_control.~in**, search for UNB.

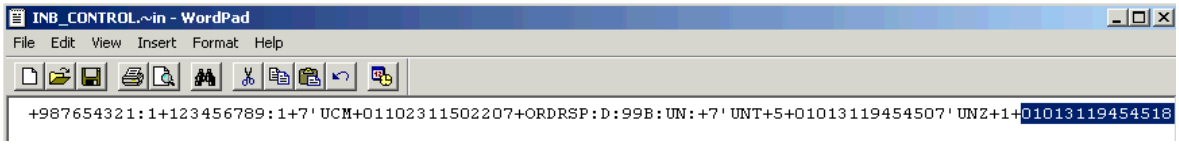
2 Go to the fifth element of the UNB segment (see Figure 62) and increment it.

Figure 62 Control Message: Incrementing the UNB Control Number



3 Go to the last segment (after UNZ) and increment it so that it matches the value for the UNB segment.

Figure 63 Control Message: Incrementing the UNZ Control Number



4 Save your change and close the file.

10.12.5 Sending and Viewing the Control Message

The final step is to send the control message.

The input file for the response message is the **<egate>\data\TP\feeder\INB_CONTROL.~in** file. Instructions for preparing this file for running were given in **"Editing the Data File" on page 139**.

To send the control message

1 Rename **INB_CONTROL.~in** to **INB_CONTROL.fin**.

2 Look in the **<egate>\data\TP\feeder** folder. The file name changes from ***INB_CONTROL.fin*** to ***INB_CONTROL.~in*** as the file is picked up.

That completes the final part of the exercise. You can view the results in Message Tracking.

To view the association of the control message to the original outbound response message in Message Tracking

1 From the e*Xchange Web interface, **Main** page, select **Message Tracking**.

The **TP Profile Selection** page appears.

2 In the **Company Profile** field, select **Car Interiors**.

3 In the **Trading Partner Profile** field, select **CarSupplies Europe**.

4 In the **eBusiness Protocol** field, select **UN/EDIFACT**.

5 In the **Direction** field, select **Outbound**.

6 Click the **Message Profile Selection**.

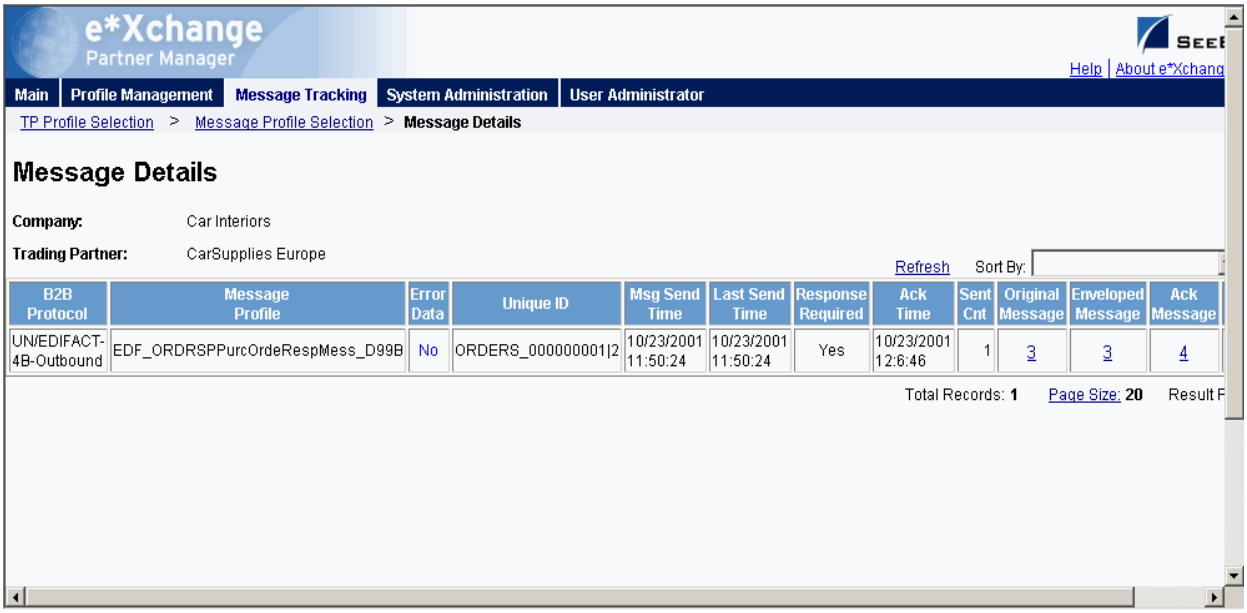
7 Select the **EDF_ORDRSPPurcOrdeRespMess_D99B** message.

8 Click the **Message Details** link to view the resulting list.

Notice that response message now has an acknowledgment available for viewing: this is the control message.

The results are shown in Figure 64.

Figure 64 Message Tracking: Outbound



Chapter 11

e*Xchange Implementation—RosettaNet

This chapter discusses the steps involved to create an e*Xchange implementation that transfers RosettaNet data.

The components for this implementation are provided on your installation CD. For instructions on installing and using the implementation components, see **"Using the Implementation Sample" on page 147**.

11.1 Overview

An e*Xchange implementation makes use of the features designed to add and remove enveloping information for messages exchanged between trading partners.

In an e*Xchange implementation, use the e*Xchange Web interface to set up trading partner information, and the e*Gate Enterprise Manager GUI to add user-defined e*Gate components to provide connectivity to the business application or trading partner. Once this is done, the pre-configured e*Xchange Schema components handle enveloping and de-enveloping Events as they travel through the e*Xchange system.

The major steps for the implementation are as follows:

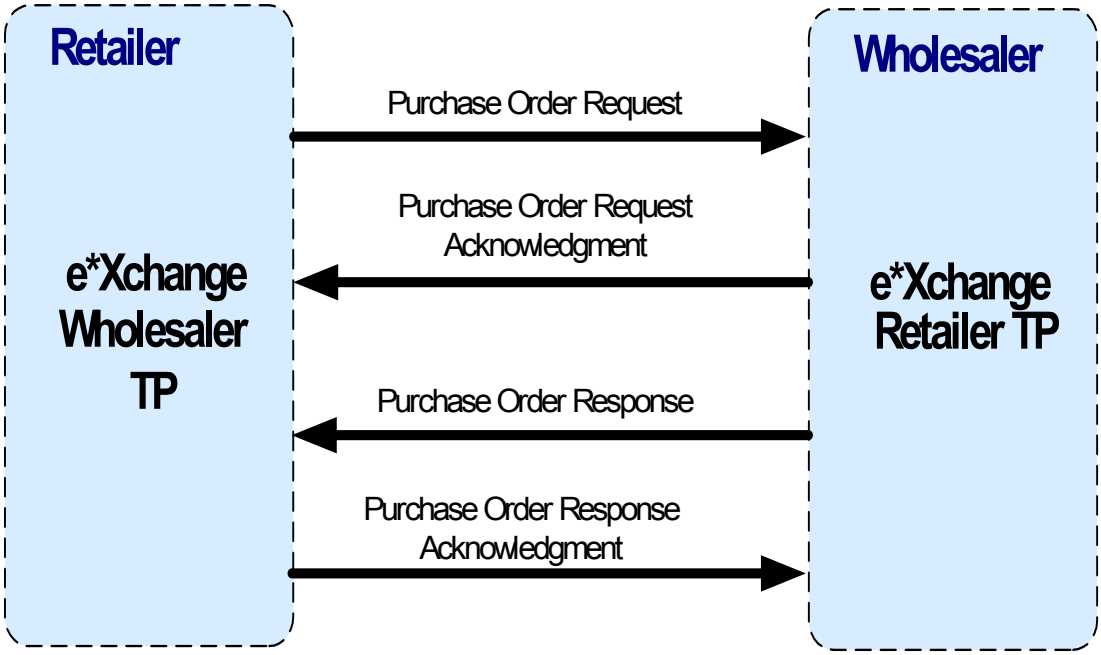
1. Create the trading partner profiles.
2. Configure the user-defined e*Ways that connect the business application to e*Xchange and exchange messages with the trading partner.
3. Configure the e*Xchange e*Way.
4. Run and test the scenario.

11.1.1 Case Study: Sending a RosettaNet Purchase Order

The case study discussed in this chapter illustrates one possible implementation of sending a purchase order to a trading partner.

In this example, a RosettaNet purchase order is sent to an external trading partner and the response is sent back. This is achieved by configuring a "loopback" using two trading partners. The flow of messages between the two trading partners is shown in Figure 65.

Figure 65 RosettaNet Implementation - Message Flow



Typically the two trading partners would be located on separate machines and transport the messages using a communication protocol such as HTTP. However, for this example you can define both trading partners on the same machine and use local files to transfer the messages between systems.

The RosettaNet enveloping is automatically added to the message by e*Xchange based on trading partner information retrieved from the e*Xchange database, and then it is sent to an external system.

Typically, the purchase order information would be provided by a business application and may or may not be in RosettaNet format. A user-defined e*Way must be created to connect to a business application in order to receive the data and put it into the proper RosettaNet format. In order to simplify this example, the purchase order information is provided in the form of a text file that is already in RosettaNet format.

Figure 66 e*Xchange Scenario Data Flow

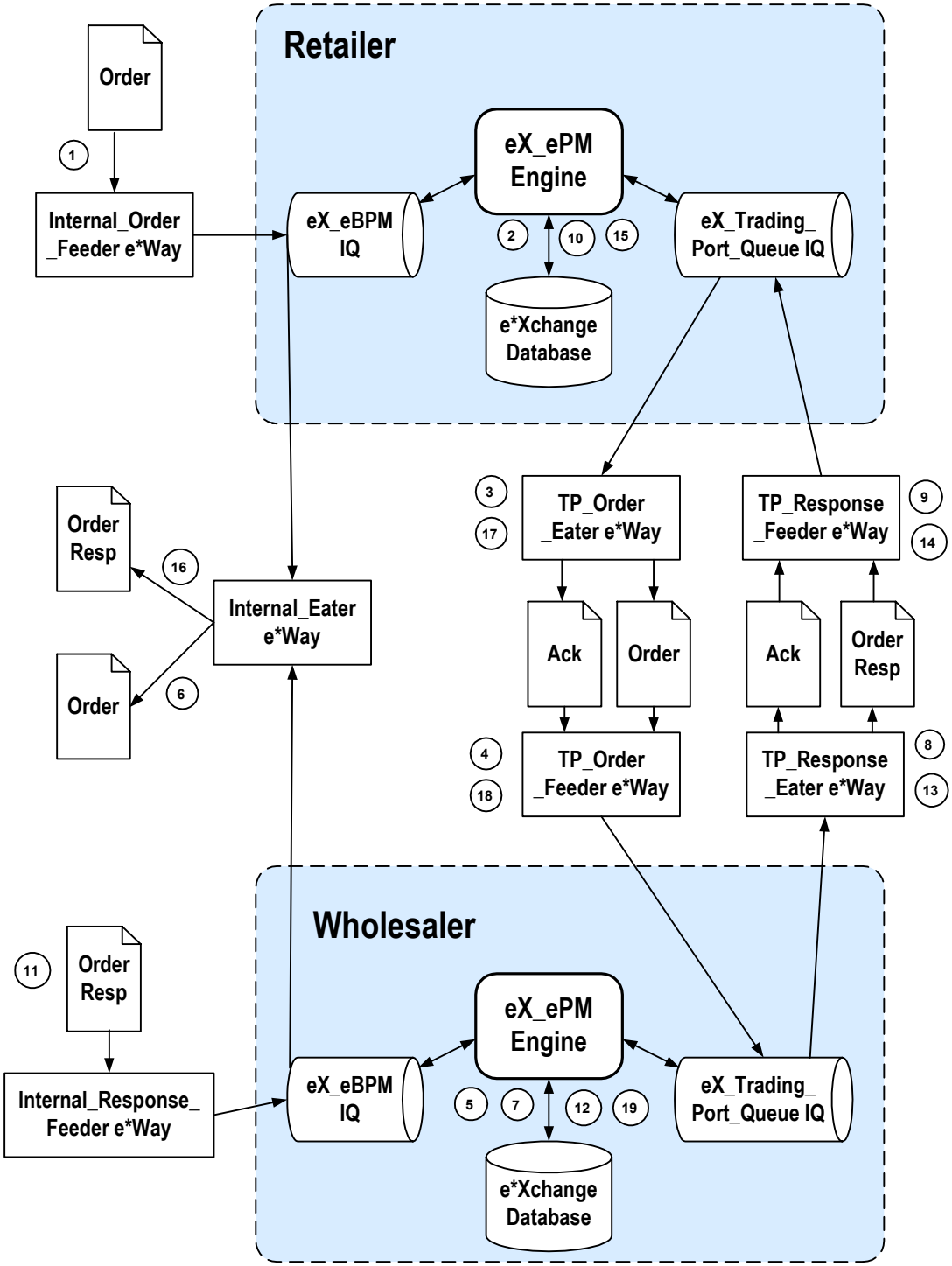


Figure 66 data flow description

1. The **Internal_Order_Feeder** e*Way picks up the purchase order message and publishes it to the Buyer **eX_eBPM** IQ.
2. The Retailer e*Xchange engine picks the purchase order message up from the IQ, validates it, saves it to the database, and publishes the purchase order message to the Retailer **eX_Trading_Port_Queue** IQ.
3. The **TP_Order_Eater** e*Way sends out the purchase order message to the Wholesaler trading partner by writing the purchase order message to file.
4. The **TP_Order_Feeder** e*Way picks up the message from the file and publishes it to the Wholesaler **eX_Trading_Port_Queue** IQ.
5. The Wholesaler e*Xchange engine picks the purchase order message up from the IQ, validates it, saves it to the database, and publishes the purchase order message to the Wholesaler **eX_eBPM** IQ.
6. The **Internal_Eater** e*Way sends out the purchase order message to the internal system by writing the purchase order message to file.
7. The Wholesaler e*Xchange engine publishes a purchase order acknowledgment message to the Wholesaler **eX_Trading_Port_Queue** IQ.
8. The **TP_Response_Eater** e*Way sends out the purchase order acknowledgment message to the Retailer trading partner by writing the purchase order acknowledgment message to file.
9. The **TP_Response_Feeder** e*Way picks up the purchase order acknowledgment message from the file and publishes it to the Retailer **eX_Trading_Port_Queue** IQ.
10. The Retailer e*Xchange engine picks the purchase order acknowledgment message up from the IQ, validates it, saves it to the database.
11. The **Internal_Response_Feeder** e*Way picks up the purchase order response message and publishes it to the Wholesaler **eX_eBPM** IQ.
12. The Wholesaler e*Xchange engine picks the purchase order message up from the IQ, validates it, saves it to the database, and publishes the purchase order message to the Wholesaler **eX_Trading_Port_Queue** IQ.
13. The **TP_Response_Eater** e*Way sends out the purchase order response message to the Retailer trading partner by writing the purchase order response message to file.
14. The **TP_Response_Feeder** e*Way picks up the purchase order response message from the file and publishes it to the Retailer **eX_Trading_Port_Queue** IQ.
15. The Retailer e*Xchange engine picks the purchase order response message up from the IQ, validates it, saves it to the database, and publishes the purchase order response message to the Retailer **eX_eBPM** IQ. The Retailer e*Xchange engine also publishes the purchase order response acknowledgment message to the Retailer **eX_Trading_Port_Queue** IQ.
16. The **Internal_Eater** e*Way sends out the purchase order response message to the internal system by writing the purchase order message to file.

17 The **TP_Order_Eater** e*Way sends out the purchase order response acknowledgment message to the Wholesaler trading partner by writing the purchase order message to file.

18 The **TP_Order_Feeder** e*Way picks up the message from the file and publishes it to the Wholesaler **eX_Trading_Port_Queue** IQ.

19 The Wholesaler e*Xchange engine picks the purchase order response acknowledgment message up from the IQ, validates it and updates database.

11.2 Using the Implementation Sample

The components for this implementation are provided on your installation CD, and are located in **\setup\eXchange\sample\ROSETTANET_SAMPLE_IMPLEMENTATION.zip**.

To install the components

1 Unzip the file to a local directory.

2 Install the e*Gate schema using one of the following commands. The schema name is user defined.

Note: *The default registry port number is 23001.*

A For UNIX:

```
sh install_rosettanet_po.sh <egate_registry_host_name>
<schema_name> <user_name> <password> <egate_registry_port_num>
```

B For Windows:

```
install_rosettanet_po.bat <egate_registry_host_name> <schema_name>
<user_name> <password> <egate_registry_port_num>
```

3 Use the e*Xchange Import function to import **ROSETTANET.exp** into e*Xchange Partner Manager.

4 Copy the **demos** folder to the **<egate>** directory.

5 Configure the **eX_ePM** e*Way as described in **"Configure the eX_ePM e*Way" on page 192**.

The steps on the following pages describe how the components for this implementation were created. See **"Running the Scenario" on page 193** for instructions to run the implementation.

11.3 Create the Trading Partner Profiles

The trading partner profiles in e*Xchange act as the repositories for the information necessary to send messages back and forth between the entities. They contain all of the information to properly envelope an Event and forward it to its correct destination.

When creating trading partner profiles, check your values carefully before saving or leaving a section/screen, because many values cannot be changed once they are committed to the database due to auditing restrictions. You can inactivate erroneous information and add the correct information under a different company, outer envelope, and so on.

Refer to the *e*Xchange Partner Manager User's Guide* for detailed assistance with the process of creating trading partner profiles.

Trading Partner Information Hierarchy

e*Xchange stores trading partner information at various levels. The process of creating a trading partner profile proceeds from the most general inclusive level, that of a company with which you do business, to the most specific information regarding an message that you wish to send (the message profile).

11.3.1 The Retailer Company

The Retailer Company uses the RosettaNet format to exchange business data with its customers. In our example we send a purchase order from the Retailer Company to the Wholesaler Company.

On the Retailer, you configure the trading partner profile for the Wholesaler company. Figure 67 shows an overview of the components that you need to create for this example, including:

- Company
- Trading Partner
- B2B Protocol Information
- Message Profiles

Figure 67 Wholesaler Trading Partner Configuration on the Retailer Company

To configure the Wholesaler trading partner profile follow the steps listed below:

- **Step 1: Create the Wholesaler Company** on page 149
- **Step 2: Create the Wholesaler Trading Partner** on page 150
- **Step 3: Set Up Inbound B2B Protocol Information (Wholesaler TP)** on page 150
- **Step 4: Create the Inbound Message Profiles (Wholesaler TP)** on page 151
- **Step 5: Set Up Outbound B2B Protocol Information (Wholesaler TP)** on page 153
- **Step 6: Create the Outbound Message Profiles (Wholesaler TP)** on page 153
- **Step 7: Configure Return Messages for Inbound (Wholesaler TP)** on page 155

The following procedure and accompanying tables were used to create the Wholesaler Company trading partner for this example.

Step 1: Create the Wholesaler Company

1. Log in to the e*Xchange Web interface.
2. From the **Main** page, click **Profile Management**.
3. From the **Company** page, click **New**.
4. In the **Company - adding** page, enter the **Company** name, “Wholesaler Company”.
5. Click **Next**.

This saves your changes and returns to the **Company** page.

Note: *The security information is automatically configured for the current user.*

Step 2: Create the Wholesaler Trading Partner

1. From the **Company** page, ensure that the "Wholesaler Company" is selected, and click **Continue: Trading Partner**.
2. From the **Trading Partner** page, click **New** to access the **Trading Partner - adding** page.
3. Enter the **Trading Partner Name**, "Wholesaler TP".
4. Click **Next**.

   This saves your changes and returns to the **Trading Partner** page.

The required security information defaults from the company level.

Step 3: Set Up Inbound B2B Protocol Information (Wholesaler TP)

To set up the inbound B2B Protocol Information

1. From the **Trading Partner** page, ensure that the "Wholesaler TP" is selected, and click **Continue: B2B Protocol**.
2. From the **B2B Protocol** page, click **New** to access the **B2B Protocol - adding** page.
3. Enter the information listed in Table 45.

   In an actual implementation, your local administrator can provide you with the B2B Protocol information. For an explanation of the B2B Protocol parameters, see the *e*Xchange Partner Manager User's Guide*.

Table 45 B2B Protocol Information

Parameter	Value
eBusiness Protocol	RosettaNet
Version	2.0
Direction	Inbound

4. Click **Next** to save your changes and access the **General** section.
5. Enter the information listed in Table 46.

Table 46 B2B Protocol Information, General Page

Parameter	Value
Logical Name	Wholesaler
Status	Active
Communication Protocol	HTTP

6 Click **Next** to save your changes and access the **Transport Component** section.

7 No changes are required. Click **Next** to access the **Message Security** section.

8 No changes are required. Click **Finish** to save the information and return to the **B2B Protocol** page.

Step 4: Create the Inbound Message Profiles (Wholesaler TP)

For the purposes of this scenario, you must set up the following inbound message profiles:

- Purchase Order Response Message (3A4 Response - Manage Purchase Order)
- Control (Business Signal - Receipt Acknowledge)

To set up the 3A4 Response - Manage Purchase Order inbound message profile

1 From the **B2B Protocol** page, click **Continue: Message Profile**.

2 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

3 In the **Name** window, type **3A4 Response - Manage Purchase Order**, and leave all other parameters with their default values.

4 Click **Next** to access the **Delivery Header** section. Enter the information listed in Table 47.

Table 47 Delivery Header (3A4 Response - Manage Purchase Order)

Name	Value
From Global Partner Business Identification	6264712002
To Global Partner Business Identification	6264716002

5 Click **Next** to access the **Service Header** section. Enter the information listed in Table 48.

Note: *This table only lists the attributes required to make this scenario work.*

Table 48 Service Header (3A4 Response - Manage Purchase Order)

Name	Value
Activity Identifier	1
From Global Partner Role Classification	Seller
From Global Business Service Code	Seller Service
Global Business Action/Signal Code	Purchase Order Acceptance Action
Global Business Action/Signal Version Identifier	01.02
Global Process Code(PIP)	3A4
PIP Version Identifier	01.02
To Global Partner Role Classification	Buyer

Name	Value
To Global Business Service Code	Buyer Service
Usage Code	Test

6 Click **Next** to access the **Return Messages** section.

7 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

Note: *Setup of the return inner envelope is done later, after the Outbound inner envelopes have been set up.*

To set up the Business Signal - Receipt Acknowledge inbound inner envelope

1 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

2 In the **Name** window, type **Business Signal - Receipt Acknowledge**, and leave all other parameters with their default values.

3 Click **Next** to access the **Delivery Header** section. The information should appear as listed in Table 49.

Table 49 Delivery Header (Business Signal - Receipt Acknowledge)

Name	Value
From Global Partner Business Identification	6264712002
To Global Partner Business Identification	6264716002

4 Click **Next** to access the **Service Header** section. Enter the information listed in Table 50.

Note: *This table only lists the attributes required to make this scenario work.*

Table 50 Service Header (Business Signal - Receipt Acknowledge)

Name	Value
Activity Identifier	1
From Global Partner Role Classification	Seller
From Global Business Service Code	Seller Service
Global Business Action/signal Code	Receipt Acknowledge
Global Business Action/Signal Version Identifier	01.02
To Global Partner Role Classification	Buyer
To Global Business Service Code	Buyer Service
Usage Code	Test

5 Click **Next** to access the **Return Messages** section.

6 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

Step 5: Set Up Outbound B2B Protocol Information (Wholesaler TP)

To set up the outbound B2B Protocol Information

As a shortcut, you can copy the Inbound B2B protocol information as a model for the Outbound B2B protocol information.

1 On the **B2B Protocol** page, select the RosettaNet-2.0-Inbound protocol that you created in **"To set up the inbound B2B Protocol Information" on page 150**.

2 Click **Copy**.

The **B2B Protocol - copying** page appears.

3 In the **Direction** field, ensure that **Outbound** is selected.

4 Click **Next**.

The **B2B Protocol - copying**, **General** page appears.

5 No changes are needed: click **Next** to accept the values and access the **Transport Component** page.

6 No changes are needed: click **Next** to accept the values and access the **Message Security** page.

7 No changes are required. Click **Finish** to save the information and return to the **B2B Protocol** page.

Step 6: Create the Outbound Message Profiles (Wholesaler TP)

For the purposes of this scenario, you must set up the following outbound message profiles:

- Purchase Order Message (3A4 Request - Manage Purchase Order)
- Control (Business Signal - Receipt Acknowledge)

To set up the 3A4 Request - Manage Purchase Order outbound message profile

1 From the **B2B Protocol** page, click **Continue: Message Profile**.

2 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

3 In the **Name** window, type **3A4 Request - Manage Purchase Order**. Leave all other parameters with their default values.

4 Click **Next** to access the **Delivery Header** section. Enter the information listed in Table 51.

Table 51 Delivery Header (3A4 Request - Manage Purchase Order)

Name	Value
From Global Partner Business Identification	6264716002

Name	Value
To Global Partner Business Identification	6264712002

5 Click **Next** to access the **Service Header** section. Enter the information listed in Table 52.

Table 52 Service Header (3A4 Request - Manage Purchase Order)

Name	Value
Activity Identifier	1
From Global Partner Role Classification	Buyer
From Global Business Service Code	Buyer Service
Global Business Action/signal Code	Purchase Order Request Action
Global Business Action/Signal Version Identifier	01.02
Global Process Code(PIP)	3A4
PIP Version Identifier	01.02
To Global Partner Role Classification	Seller
To Global Business Service Code	Seller Service
Usage Code	Test

6 Click **Next** to access the **Return Messages** section.

7 Select the return message (select the check box), and enter the values, as shown in Table 53.

Table 53 Return Message Values: Outbound

Name	Response Time	Period	# Retries
Business Signal - Receipt Acknowledge	2	Minutes	1
3A4 Response - Manage Purchase Order	5	Minutes	1

8 Click **Finish** to save the information and return to the **Message Profile** page.

To set up the Business Signal - Receipt Acknowledge outbound inner envelope

1 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

2 In the **Name** window, type **Business Signal - Receipt Acknowledge**, leave all other parameters with their default values.

3 Click **Next** to access the **Delivery Header** section. The information should appear as listed in Table 54.

Table 54 Delivery Header (Business Signal - Receipt Acknowledge)

Name	Value
From Global Partner Business Identification	6264716002
To Global Partner Business Identification	6264712002

4 Click **Next** to access the **Service Header** section. Enter the information listed in Table 55.

Note: *This table only lists the attributes required to make this scenario work.*

Table 55 Service Header (Business Signal - Receipt Acknowledge)

Name	Value
Activity Identifier	1
From Global Partner Role Classification	Buyer
From Global Business Service Code	Buyer Service
Global Business Action/Signal Code	Receipt Acknowledge
Global Business Action/Signal Version Identifier	01.02
To Global Partner Role Classification	Seller
To Global Business Service Code	Seller Service
Usage Code	Test

5 Click **Next** to access the **Return Messages** section.

6 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

Step 7: Configure Return Messages for Inbound (Wholesaler TP)

To set up the Return Inner Envelope values for Inbound

Once you have set up inbound and outbound message profiles, you can specify return messages.

1 From the **B2B Protocol** page, select **RosettaNet-2.0-Inbound**.

2 Click **Continue: Message Profile**.

3 From the **Message Profile** page, select **3A4 Response - Manage Purchase Order** from the drop-down list.

4 Click the **Return Messages** link to access the **Return Messages** section.

5 Click **Edit**.

6 Select the return messages (select the check boxes), and enter the values, as shown in Table 56.

Table 56 Return Message Values: Inbound

Name	Response Time	Period	# Retries
Business Signal - Receipt Acknowledge	10	Minutes	0

7 Click **Apply** to save the information and return to the **Message Profile** page.

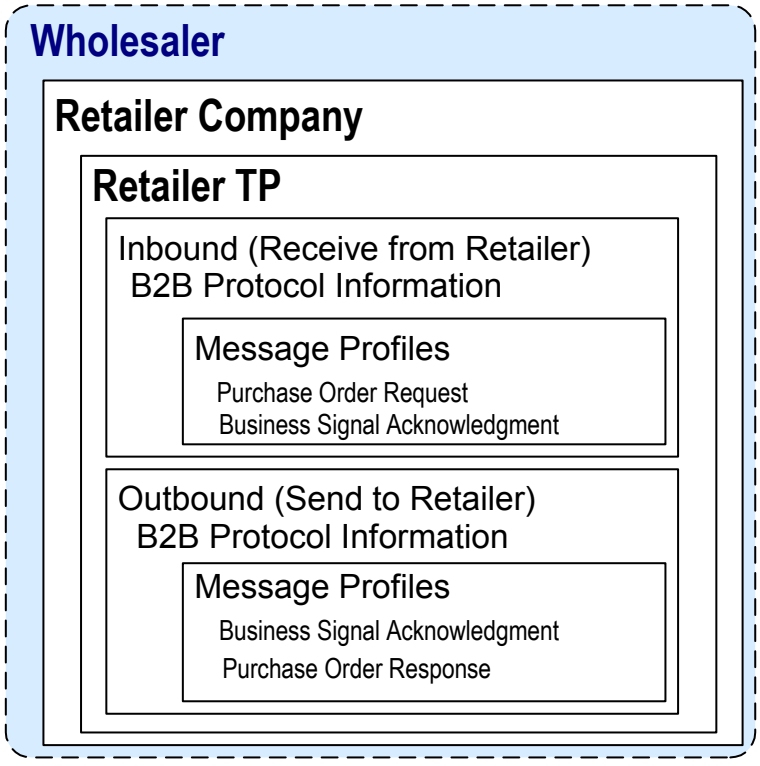
11.4 The Wholesaler

The Wholesaler is the supplier that uses the RosettaNet format to exchange business data with its customers. In our example the Wholesalers receive a purchase order from the Retailer and sends a purchase order response back.

On the Wholesaler, you configure the Trading Partner Profile for the Retailer company. Figure 68 shows an overview of the components that you need to create for this example, including:

- Company
- Trading Partner
- B2B Protocol Information
- Message Profiles

Figure 68 Retailer Trading Partner Configuration on the Wholesaler Company



To configure the Retailer Trading Partner Profile you must follow the steps listed below:

- **Step 1: Create the Retailer Company** on page 158
- **Step 2: Create the Retailer Trading Partner** on page 158
- **Step 3: Set Up Inbound B2B Protocol Information (Retailer TP)** on page 158
- **Step 4: Create the Inbound Message Profiles (Retailer TP)** on page 159

- **Step 5: Set Up the Outbound B2B Protocol Information (Retailer TP)** on page 161
- **Step 6: Set Up the Outbound Message Profiles (Retailer TP)** on page 162
- **Step 7: Configure Return Messages for Inbound (Retailer TP)** on page 164

The following procedure and accompanying tables were used to create the Retailer Company trading partner for this example.

Step 1: Create the Retailer Company

1. Log in to the e*Xchange Web Interface.
2. From the **Main** page, click **Profile Management**.
3. From the **Company** page, click **New**.
4. In the **Company - adding** page, enter the **Company** name, "Retailer Company".
5. Click **Next**.

   This saves your changes and returns to the **Company** page.

Note: *The security information is automatically configured for the current user.*

Step 2: Create the Retailer Trading Partner

1. From the **Company** page, ensure that the "Retailer Company" is selected, and then click **Continue: Trading Partner**.
2. From the **Trading Partner** page, click **New** to access the **Trading Partner - adding** page.
3. Enter the **Trading Partner Name**, "Retailer TP".
4. Click **Next**.

   This saves your changes and returns to the **Trading Partner** page.

The required security information defaults from the company level.

Step 3: Set Up Inbound B2B Protocol Information (Retailer TP)

To set up the inbound B2B Protocol Information

1. From the **Trading Partner** page, ensure that the "Retailer TP" is selected, and click **Continue: B2B Protocol**.
2. From the **B2B Protocol** page, click **New** to access the **B2B Protocol - adding** page.
3. Enter the information listed in Table 57.

   In an actual implementation, your local administrator can provide you with the B2B Protocol information. For an explanation of the B2B Protocol parameters, see the *e*Xchange Partner Manager User's Guide*.

Table 57 B2B Protocol Information

Parameter	Value
eBusiness Protocol	RosettaNet
Version	2.0
Direction	Inbound

4 Click **Next** to save your changes and access the **General** section.

5 Enter the information listed in Table 58.

Table 58 B2B Protocol Information, General Page

Parameter	Value
Logical Name	Retailer
Status	Active
Communication Protocol	HTTP

6 Click **Next** to save your changes and access the **Transport Component** section.

7 No changes are required. Click **Next** to access the **Message Security** section.

8 No changes are required. Click **Finish** to save the information and return to the **B2B Protocol** page.

Step 4: Create the Inbound Message Profiles (Retailer TP)

For the purposes of this scenario, you must set up the following inbound message profiles:

- Purchase Order Message (3A4 Request - Manage Purchase Order)
- Control (Business Signal - Receipt Acknowledge)

To set up the 3A4 Request - Manage Purchase Order inbound inner envelope

1 From the **B2B Protocol** page, click **Continue: Message Profile**.

2 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

3 In the **Name** window, type **3A4 Request - Manage Purchase Order**, and leave all other parameters with their default values.

4 Click **Next** to access the **Delivery Header** section. Enter the information listed in Table 59.

Table 59 Delivery Header (3A4 Request - Manage Purchase Order)

Name	Value
From Global Partner Business Identification	6264716002
To Global Partner Business Identification	6264712002

5 Click **Next** to access the **Service Header** section. Enter the information listed in Table 60.

Table 60 Service Header (3A4 Request - Manage Purchase Order)

Name	Value
Activity Identifier	1
From Global Partner Role Classification	Buyer
From Global Business Service Code	Buyer Service
Global Business Action/signal Code	Purchase Order Request Action
Global Business Action/Signal Version Identifier	01.02
Global Process Code(PIP)	3A4
PIP Version Identifier	01.02
To Global Partner Role Classification	Seller
To Global Business Service Code	Seller Service
Usage Code	Test

6 Click **Next** to access the **Return Messages** section.

7 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

Note: *Setup of the return inner envelope is done later, after the Outbound inner envelopes have been set up.*

To set up the Business Signal - Receipt Acknowledge inbound inner envelope

1 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

2 In the **Name** window, type **Business Signal - Receipt Acknowledge**, and leave all other parameters with their default values.

3 Click **Next** to access the **Delivery Header** section. The information should appear as listed in Table 61.

Table 61 Delivery Header (Business Signal - Receipt Acknowledge)

Name	Value
From Global Partner Business Identification	6264716002
To Global Partner Business Identification	6264712002

4 Click **Next** to access the **Service Header** section. Enter the information listed in Table 62.

Note: *This table only lists the attributes required to make this scenario work.*

Table 62 Service Header (Business Signal - Receipt Acknowledge)

Name	Value
Activity Identifier	1
From Global Partner Role Classification	Buyer
From Global Business Service Code	Buyer Service
Global Business Action/signal Code	Receipt Acknowledge
Global Business Action/Signal Version Identifier	01.02
To Global Partner Role Classification	Seller
To Global Business Service Code	Seller Service
Usage Code	Test

5 Click **Next** to access the **Return Messages** section.

6 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

Step 5: Set Up the Outbound B2B Protocol Information (Retailer TP)

To set up the outbound B2B Protocol information

As a shortcut, you can copy the inbound B2B Protocol information as a model for the Inbound B2B Protocol information.

1 On the **B2B Protocol** page, select the RosettaNet-2.0-Inbound protocol that you created in **"To set up the inbound B2B Protocol Information" on page 158**. Click **Copy**.

The **Copy Type** page appears.

2 Clear the **Include Sub-components** check box and then click **OK**.

The **Copy Type** page appears.

3 Clear the **Include Sub-components** check box and then click **OK**.

The **B2B Protocol - Copying** page appears.

4 In the **Direction** field, ensure that **Outbound** is selected.

5 Click **Next**.

The **B2B Protocol - copying**, **General** page appears.

6 No changes are needed: click **Next** to accept the values and access the **Transport Component** page.

7 No changes are needed: click **Next** to accept the values and access the **Message Security** page.

8 No changes are required. Click **Finish** to save the information and return to the **B2B Protocol** page.

Step 6: Set Up the Outbound Message Profiles (Retailer TP)

To set up the 3A4 Response - Manage Purchase Order outbound Message Profile

1. From the **B2B Protocol** page, click **Continue: Message Profile**.
2. From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.
3. In the **Name** window, type **3A4 Response - Manage Purchase Order**, and leave all other parameters with their default values.
4. Click **Next** to access the **Delivery Header** section. Enter the information listed in Table 63.

Table 63 Delivery Header (3A4 Response - Manage Purchase Order)

Name	Value
From Global Partner Business Identification	6264712002
To Global Partner Business Identification	6264716002

5. Click **Next** to access the **Service Header** section. Enter the information listed in Table 64.

Note: *This table only lists the extended attributes required to make this scenario work.*

Table 64 Service Header (3A4 Response - Manage Purchase Order)

Name	Value
Activity Identifier	1
From Global Partner Role Classification	Seller
From Global Business Service Code	Seller Service
Global Business Action/signal Code	Purchase Order Acceptance Action
Global Business Action/Signal Version Identifier	01.02
Global Process Code(PIP)	3A4
PIP Version Identifier	01.02
To Global Partner Role Classification	Buyer
To Global Business Service Code	Buyer Service
Usage Code	Test

6. Click **Next** to access the **Return Messages** section.
7. Select the return message (select the check box), and enter the values, as shown in Table 65.

Table 65 Return Message Values: Outbound

Name	Response Time	Period	# Retries
Business Signal - Receipt Acknowledge	2	Minutes	1

8 Click **Finish** to save the information and return to the **Message Profile** page.

To set up the Business Signal - Receipt Acknowledge outbound message profile

1 From the **Message Profile** page, click the **New** button to access the **Message Profile - adding** page.

2 In the **Name** window, type **Business Signal - Receipt Acknowledge**, and leave all other parameters with their default values.

3 Click **Next** to access the **Delivery Header** section. The information should appear as listed in Table 66.

Table 66 Delivery Header (Business Signal - Receipt Acknowledge)

Name	Value
From Global Partner Business Identification	6264712002
To Global Partner Business Identification	6264716002

4 Click **Next** to access the **Service Header** section. Enter the information listed in Table 67.

Note: *This table only lists the extended attributes required to make this scenario work.*

Table 67 Service Header (Business Signal - Receipt Acknowledge)

Name	Value
Activity Identifier	1
From Global Partner Role Classification	Seller
From Global Business Service Code	Seller Service
Global Business Action/signal Code	Receipt Acknowledge
Global Business Action/Signal Version Identifier	01.02
To Global Partner Role Classification	Buyer
To Global Business Service Code	Buyer Service
Usage Code	Test

5 Click **Next** to access the **Return Messages** section.

6 No changes are required. Click **Finish** to save the information and return to the **Message Profile** page.

Step 7: Configure Return Messages for Inbound (Retailer TP)

To set up the Return Inner Envelope values for Inbound

Once you have set up inbound and outbound message profiles, you can specify return messages.

1. From the **B2B Profile** page, select **RosettaNet-2.0-Inbound**.
2. Click **Continue: Message Profile**.
3. From the **Message Profile** page, select **3A4 Request - Manage Purchase Order** from the drop-down list.
4. Click the **Return Messages** link to access the **Return Messages** section.
5. Click **Edit**.
6. Select the return messages (select the check boxes), and enter the values, as shown in Table 68.

Table 68 Return Message Values: Inbound

Name	Response Time	Period	# Retries
3A4 Response - Manage Purchase Order	10	Minutes	0
Business Signal - Receipt Acknowledge	5	Minutes	0

7. Click **Apply** to save the information and return to the **Message Profile** page.

11.5 Clone the eXSchema

The supplied schema named eXSchema contains the components required to run e*Xchange. Make a copy of this schema and then configure the copy for this implementation.

To make a copy of eXSchema

1. Open eXSchema in the e*Gate Enterprise Manager GUI.
2. Export eXSchema.
3. Create a new schema named **RosettaNet** using the exported file.

11.6 Configure the Internal_Order_Feeder e*Way

The component (e*Way or BOB) that feeds data into e*Xchange must put the data into the appropriate business protocol format. It must also populate the required fields in the e*Xchange Event that is processed by e*Xchange.

This component is entirely user-defined and must be added to the RosettaNet schema. The type of component to use depends on whether a connection to a system outside

e*Gate must be made, and if so, what type of system. Typically, this component is an e*Way that connects to a business application such as SAP that sends out electronic messages. These messages may or may not be in the format required by the trading partner to which they are being sent. If the data is not in the correct format, the e*Way must translate the data into the required format before it is sent to the e*Xchange system for enveloping and forwarding to the trading partner.

The e*Xchange Internal_Order_Feeder e*Way

This example simulates the publication of an electronic purchase order from a trading partner. This file, which is already in RosettaNet format, is picked up by a file e*Way and moved into the e*Xchange system.

Configuration Steps

Follow these steps to configure the **Internal_Order_Feeder** e*Way.

1. Create and configure the e*Way.
2. Create the ETDs.
3. Create the Collaboration.

11.6.1 Step 1: Create and configure the Internal_Order_Feeder e*Way

1. Create an e*Way called **Internal_Order_Feeder**.
2. In the **e*Way Properties** dialog box, in the **Executable file** area of the **General** tab, browse for **stcewfile.exe.**
3. In the **e*Way Properties** dialog box, in the **Configuration file** area of the **General** tab, click **New**.
4. Use the following table to set the e*Way parameters for the **Internal_Order_Feeder** e*Way:

Table 69 Internal_Order_Feeder e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\Demos\RosettaNet\input\order
	MultipleRecordsPerFile	NO
	(All others)	(Default)
Performance Testing	(All)	(Default)

5. When finished editing the e*Way configuration file, save your work and close the e*Way editor.
6. Click **OK** to close the **e*Way Properties** dialog box.

11.6.2 Step 2: Create the Internal_Order_Feeder ETDs

In the present example, since the data is already in standard RosettaNet format for a purchase order, you can bring in the Event without parsing it. To do this, all that is required is an ETD with a root node.

*Note: If **root.ssc** already exists, you do not need to create the ETD.*

To create the root ETD

1. Create a new ETD called **root.ssc**. In the **Type** box, select Delimited, and select **Other** from the drop-down list.
2. Add a single node to the structure. The ETD is shown in Figure 69.

Figure 69 root.ssc Event Type Definition

3. Save the ETD.

11.6.3 Step 3: Create the Internal_Order_Feeder Collaboration

The **Internal_Order_Feeder** Collaboration must prepare the data coming into the e*Xchange system. How complicated this task is depends on the state of the data before the **Internal_Order_Feeder** Collaboration processes it.

The **Internal_Order_Feeder** Collaboration must do the following:

- convert the data to base 64 encoding
- populate the required nodes in the e*Xchange Event sent to e*Xchange for processing

Convert the Event to Base 64 Encoding

The **Internal_Order_Feeder** Collaboration must ensure that the data going into e*Xchange doesn't include any characters that cause problems for the XML structure of the standard e*Xchange Event (for example, characters that are the same as the XML control characters). This is done by converting the entire message to base 64 encoding using the Monk function **raw->base64**, and then copying it to the payload node of the **eX_Standard_Event** ETD.

Populate the Required e*Xchange Nodes

In addition to copying the base 64 encoded message to the payload node of the TP_EVENT portion of the e*Xchange standard Event, you must provide e*Xchange with tracking information about the message.

e*Xchange Tracking Information

e*Xchange needs to know certain things about an message before processing. The **Internal_Order_Feeder** Collaboration must supply this information by populating certain required nodes in the Event that is sent to e*Xchange. At a minimum you must tell e*Xchange:

- Direction (inbound or outbound)
- Partner Name (logical name from the outer envelope in e*Xchange)

All of these requirements can be met by copying the appropriate information to the corresponding nodes in the TP section of the e*Xchange ETD (**eX_Standard_Event.ssc**).

The **TP_EVENT.CT.DSN.DS.Direction.CT.DSN.DS.Data** node must contain the direction of the Event: "O" for outbound to the trading partner or "I" for inbound from a trading partner.

The **TP_EVENT.CT.DSN.DS.PartnerName.CT.DSN.DS.Data** node must contain the name (case-sensitive) of the trading partner as defined in the **B2B Protocol Information**, **General** page.

The e*Xchange Payload

In addition to the tracking information, the **TP_EVENT.CT.DSN.DS.Payload.CT.DSN.DS.Data** node must be filled with the entire base 64 encoded message.

The e*Xchange Internal_Order_Feeder CRS

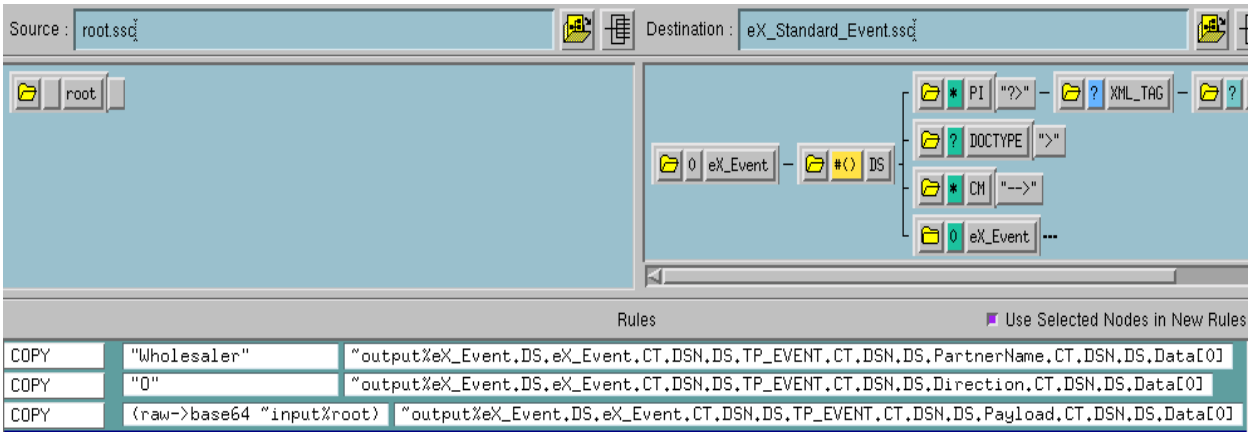
The CRS, **Internal_Order_Feeder.tsc**, used in the present example is shown in Figure 70. It does the following:

- Converts the RosettaNet message to base 64 encoding, and copies it to the Payload node of the TP_EVENT section of the e*Xchange standard Event.
- Copies "O" for outbound to the direction node of the TP_EVENT section.
- Copies the trading partner logical name "Wholesaler" to the PartnerName node of the TP_EVENT section.

To create and configure the Internal_Order_Feeder Collaboration Rule

1. Open the Collaboration Editor.
2. Create a new Collaboration Rules script named **Internal_Order_Feeder.tsc**. The Source Event Type Definition is **root.ssc**. The Destination Event Type Definition is **eX_Standard_Event.ssc**.
3. Add the rules shown in Figure 70.

Figure 70 Internal_Order_Feeder.tsc



Internal_Order_Feeder Collaboration Properties Setup

Once the CRS has been created, you must set up the Collaboration and Collaboration Rules Properties for the **Internal_Order_Feeder** Component in the Enterprise Manager GUI.

To create and configure the Internal_Order_Feeder Collaboration Rule

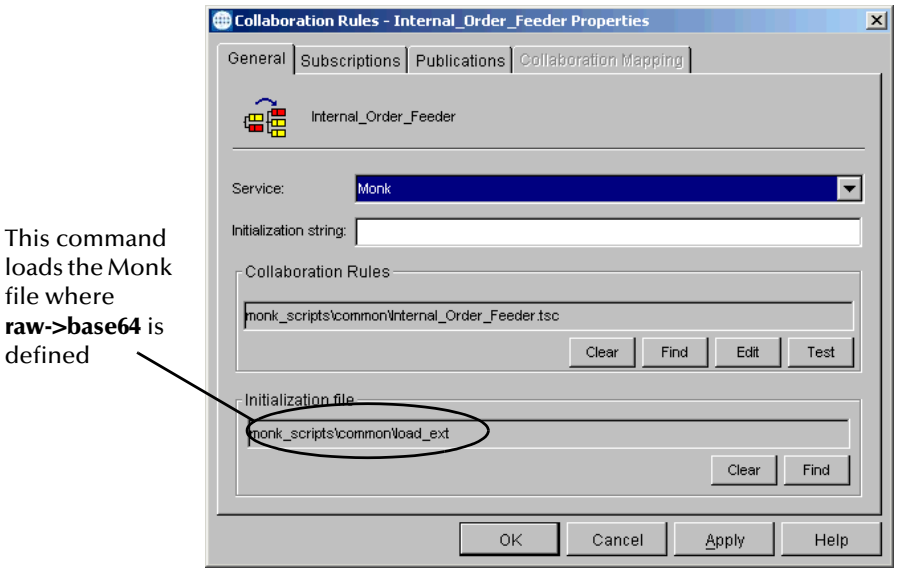
1. Create a new Collaboration Rule named **Internal_Order_Feeder**.
2. From **Internal_Order_Feeder** Collaboration Rule properties, select the **General** tab. Configure as shown in Table 70.

Table 70 Internal_Order_Feeder CR configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	Internal_Order_Feeder
Initialization File	monk_scripts\common\load_ext

Important: *To use the Monk function* ***raw->base64****, you must make sure the file containing this function has been loaded.*

Figure 71 Internal_Order_Feeder Collaboration Rules Properties Dialog Box



3 Select the **Subscriptions** tab. Select **eX_External_Evt** and move it to the right pane.

4 Select the **Publications** tab. Select **eX_to_ePM** and move it to the right pane.

To create and configure the Internal_Order_Feeder Collaboration

1 Select the **Internal_Order_Feeder** e*Way.

2 Create a new Collaboration named **Internal_Order_Feeder**.

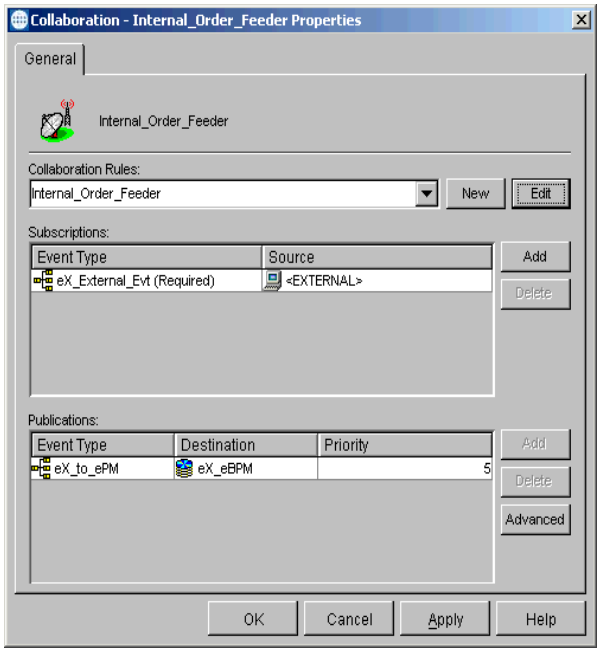
3 Configure the Internal_Order_Feeder Collaboration properties using Table 71.

Table 71 Internal_Order_Feeder Collaboration configuration

Section	Value
Collaboration Rules	Internal_Order_Feeder
Subscriptions	Event Type: eX_External_Evt Source: <EXTERNAL>
Publications	Event Type: eX_to_ePM Destination: eX_eBPM

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 72.

Figure 72 Internal_Order_Feeder Collaboration Properties



11.7 Configure the TP_Order_Eater e*Way

The component (e*Way or BOB) sends the message to the external system.

The e*Xchange TP_Order_Eater e*Way

The e*Xchange example simulates the publication of the message to the external system.

Configuration Steps

Follow these steps to configure the TP_Order_Eater e*Way.

1. Create the configuration file.
2. Create the ETDs.
3. Create the Collaboration.

11.7.1 Step 1: Create and configure the TP_Order_Eater e*Way

1. Create an e*Way called **TP_Order_Eater**.
2. In the **e*Way Properties** dialog box **General** tab, in the **Executable file** area browse for **stcewfile.exe**.

3 In the **e*Way Properties** dialog box **General** tab, in the **Configuration file** area, click **New**.

4 Use Table 72 to set the e*Way parameters for the **TP_Order_Eater** e*Way.

Table 72 TP_Order_Eater e*Way Parameters

Screen	Parameter	Setting
General Settings	AllowIncoming	NO
	AllowOutgoing	YES
Outbound (send) settings	OutputDirectory	<eGate>\Demos\RosettaNet\output\Order_Out\TP
	OutputFileName	order%d.dat
	MultipleRecordsPerFile	NO
	(All others)	(Default)
Poller (inbound) settings	(All)	(Default)
Performance Testing	(All)	(Default)

5 When finished editing the e*Way configuration file, save your work and close the e*Way editor.

6 Click **OK** to close the **e*Way Properties** dialog box.

11.7.2 Step 2: Create the TP_Order_Eater Collaboration

The **TP_Order_Eater** Collaboration must prepare the data leaving the e*Xchange system. How complicated this task is depends on the state of the data before the **TP_Order_Eater** Collaboration processes it.

The **TP_Order_Eater** Collaboration must do the following:

- put the data into the appropriate format
- convert the data to raw data

The e*Xchange TP_Order_Eater CRS

The CRS, **TP_Order_Eater.tsc**, checks that the message is for the Wholesaler Trading Partner (Wholesaler). If it is, it converts the RosettaNet message to raw data, and then copies it from the Payload node of the TP_EVENT section of the e*Xchange standard Event to the output ETD.

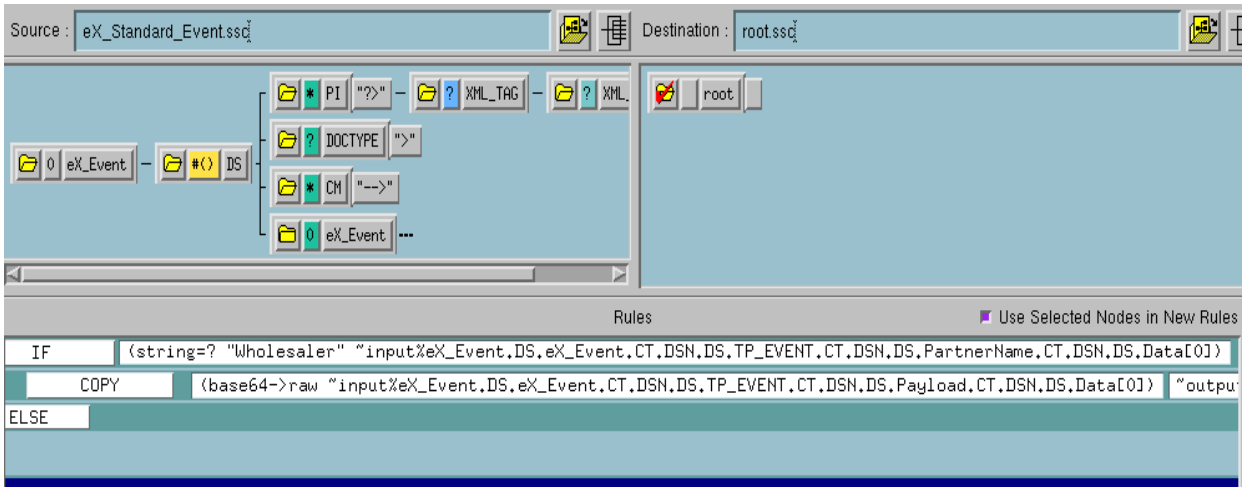
To create and configure the TP_Order_Eater Collaboration Rule

1 Open the Collaboration Editor.

2 Create a new Collaboration Rules script named **TP_Order_Eater.tsc**. The Source Event Type Definition is **eX_Standard_Event.ssc**. The Destination Event Type Definition is **root.ssc**.

3 Add the rule shown in Figure 73.

Figure 73 TP_Order_Eater.tsc



TP_Order_Eater Collaboration Properties Setup

Once the CRS has been created, you must set up the Collaboration and Collaboration Rules Properties for the **TP_Order_Eater** Component in the Enterprise Manager GUI.

To create and configure the TP_Order_Eater Collaboration Rule

1. Create a new Collaboration Rule named **TP_Order_Eater**.
2. From the **Internal_Order_Eater** Collaboration Rule properties, select the **General** tab. Configure as shown in Table 73.

Table 73 TP_Order_Eater CR Configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	TP_Order_Eater
Initialization File	monk_scripts\common\load_ext

Important: *To use the Monk function* ***base64->raw****, you must make sure the file containing this function has been loaded.*

3. Select the **Subscriptions** tab. Select **eX_HTTP** and move to the right pane.
4. Select the **Publications** tab. Select **eX_External_Evt** and move to the right pane.

To create and configure the TP_Order_Eater Collaboration

1. Select the **TP_Order_Eater** e*Way.
2. Create a new Collaboration named **TP_Order_Eater**.
3. Configure the Internal_Order_Eater Collaboration properties using Table 74.

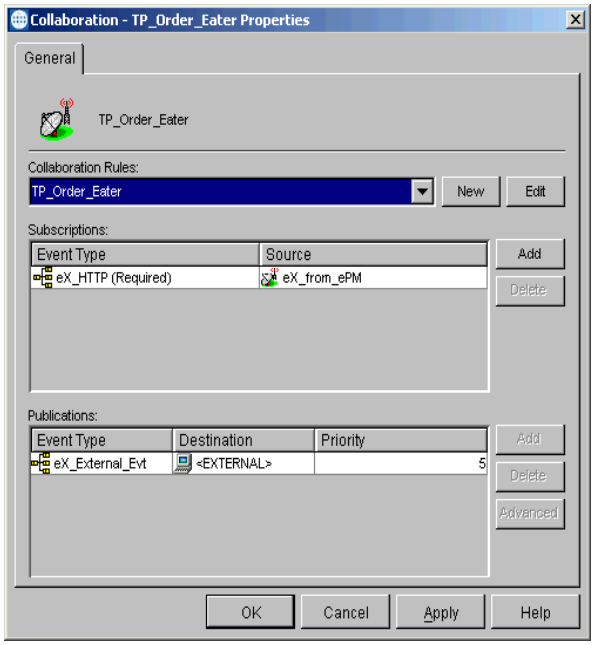
Table 74 TP_Order_Eater Collaboration configuration

Section	Value
Collaboration Rule	TP_Order_Eater

Section	Value
Subscriptions	Event Type: eX_HTTP Source: eX_from_ePM
Publications	Event Type: eX_External_Evt Destination: <EXTERNAL>

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 74.

Figure 74 TP_Order_Eater Collaboration Properties



11.8 Configure the TP_Order_Feeder e*Way

This component feeds the data that was sent to the Wholesaler Trading Partner into e*Xchange.

The e*Xchange TP_Order_Feeder e*Way

The e*Xchange example simulates the publication of an electronic purchase order from a trading partner. This file, which is already in RosettaNet format, is picked up by a file e*Way and moved into the e*Xchange system.

Configuration Steps

Follow these steps to configure the Internal_Order_Feeder e*Way.

1. Create and configure the e*Way.
2. Create the Collaboration.

11.8.1 Step 1: Create and configure the TP_Order_Feeder e*Way

1. Create an e*Way called **TP_Order_Feeder**.
2. In the **e*Way Properties** dialog box, in the **Executable file** area of the **General** tab, browse for **stcewfile.exe**.
3. In the **e*Way Properties** dialog box, in the **Configuration file** area of the **General** tab, click **New**.
4. Use Table 75 to set the e*Way parameters for the **TP_Order_Feeder** e*Way.

Table 75 TP_Order_Feeder e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\Demos\RosettaNet\output\order_out\TP
	MultipleRecordsPerFile	NO
	(All others)	(Default)
Performance Testing	(All)	(Default)

5. When finished editing the e*Way configuration file, save your work and close the e*Way editor.
6. Click **OK** to close the **e*Way Properties** dialog box.

11.8.2 Step 2: Create the TP_Order_Feeder Collaboration

The **TP_Order_Feeder** Collaboration must prepare the data coming into the e*Xchange system. How complicated this task is depends on the state of the data before the **TP_Order_Feeder** Collaboration processes it.

The **TP_Order_Feeder** Collaboration must do the following:

- convert the data to base 64 encoding
- populate the required nodes in the e*Xchange Event sent to e*Xchange for processing

Convert the Event to Base 64 Encoding

The **TP_Order_Feeder** Collaboration must ensure that the data going into e*Xchange doesn't include any characters that cause problems for the XML structure of the standard e*Xchange Event (for example, characters that are the same as the XML control characters). This is done by converting the entire message to base 64 encoding using the Monk function **raw->base64**, before copying it to the payload node of the **eX_Standard_Event** ETD.

Populate the Required e*Xchange Nodes

In addition to copying the base 64 encoded message to the payload node of the TP_EVENT portion of the e*Xchange standard Event, you must provide e*Xchange with tracking information about the message.

The e*Xchange TP_Order_Feeder CRS

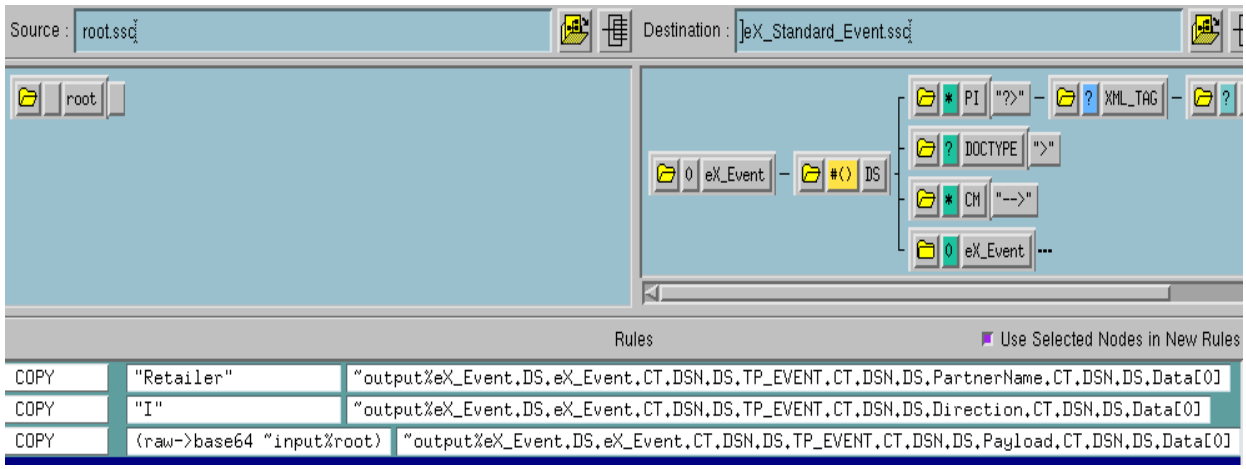
The CRS, **TP_Order_Feeder.tsc** does the following:

- Converts the RosettaNet message to base 64 encoding, and copies it to the Payload node of the TP_EVENT section of the e*Xchange standard Event.
- Copies "I" for outbound to the direction node of the TP_EVENT section.
- Copies the trading partner logical name "Retailer" to the PartnerName node of the TP_EVENT section.

To create and configure the TP_Order_Feeder Collaboration Rule

1. Open the Collaboration Editor.
2. Create a new Collaboration Rules script named **TP_Order_Feeder.tsc**. The Source Event Type Definition is **root.ssc**. The Destination Event Type Definition is **eX_Standard_Event.ssc**.
3. Add the rules shown in Figure 75.

Figure 75 TP_Order_Feeder.tsc



TP_Order_Feeder Collaboration Properties Setup

Once the CRS has been created, you must set up the Collaboration and Collaboration Rules Properties for the **TP_Order_Feeder** Component in the Enterprise Manager GUI.

To create and configure the TP_Order_Feeder Collaboration Rule

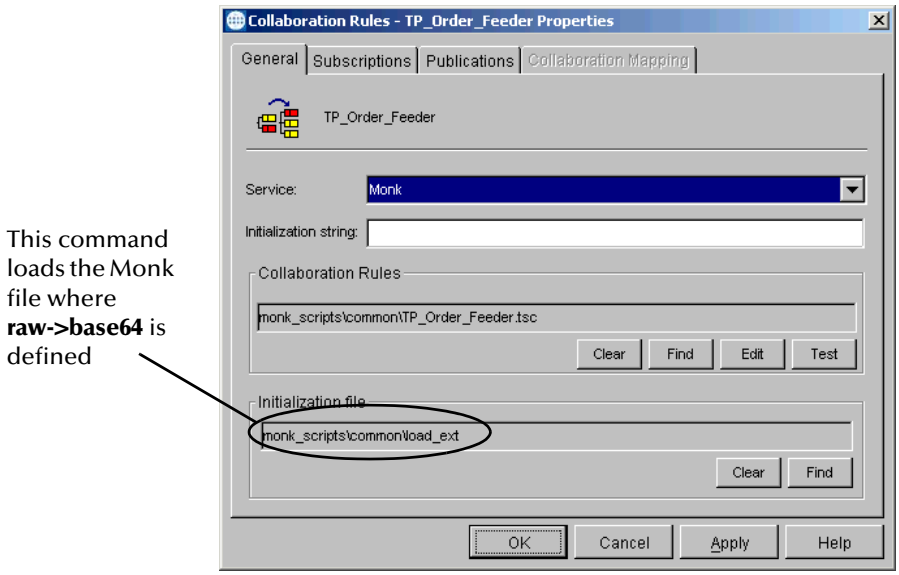
1. Create a new Collaboration Rule named **TP_Order_Feeder**.
2. From **TP_Order_Feeder** Collaboration Rule properties, select the **General** tab. Configure as shown in Table 76.

Table 76 TP_Order_Feeder CR Configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	TP_Order_Feeder
Initialization File	monk_scripts\common\load_ext

Important: *To use the Monk function* ***raw->base64****, you must make sure the file containing this function has been loaded.*

Figure 76 TP_Order_Feeder Collaboration Rules Properties Dialog Box



3 Select the **Subscriptions** tab. Select **eX_External_Evt** and move it to the right pane.

4 Select the **Publications** tab. Select **eX_to_ePM** and move it to the right pane.

To create and configure the TP_Order_Feeder Collaboration

1 Select the **TP_Order_Feeder** e*Way.

2 Create a new Collaboration named **TP_Order_Feeder**.

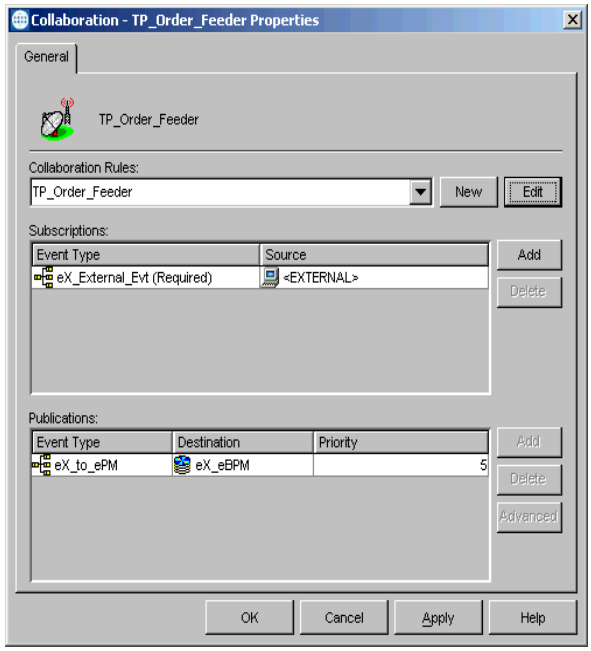
3 Configure the TP_Order_Feeder Collaboration properties as shown in Table 77.

Table 77 TP_Order_Feeder Collaboration Configuration

Section	Value
Collaboration Rules	TP_Order_Feeder
Subscriptions	Event Type: eX_External_Evt Source: <EXTERNAL>
Publications	Event Type: eX_to_ePM Destination: eX_eBPM

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 77.

Figure 77 TP_Order_Feeder Collaboration Properties



11.9 Configure the Internal_Eater e*Way

This component eats messages sent to the internal system. It is used for both purchase orders and purchase order responses.

The e*Xchange Internal_Eater e*Way

The e*Xchange example simulates the publication of the message to the internal system.

Configuration Steps

Follow these steps to configure the Internal_Eater e*Way.

1. Create the configuration file.
2. Create the ETDs.
3. Create the Collaboration.

11.9.1 Step 1: Create and configure the Internal_Eater e*Way

1. Create an e*Way called **Internal_Eater**.
2. In the **e*Way Properties** dialog box **General** tab, in the **Executable file** area browse for **stcewfile.exe**.

3 In the **e*Way Properties** dialog box, **General** tab, in the **Configuration file** area, click **New**.

4 Use the following table to set the e*Way parameters for the **Internal_Eater** e*Way.

Table 78 Internal_Eater e*Way Parameters

Screen	Parameter	Setting
General Settings	AllowIncoming	NO
	AllowOutgoing	YES
Outbound (send) settings	OutputDirectory	<eGate>\Demos\RosettaNet\output\Order_Out
	OutputFileName	order%d.dat
	MultipleRecordsPerFile	NO
	(All others)	(Default)
Poller (inbound) settings	(All)	(Default)
Performance Testing	(All)	(Default)

5 When finished editing the e*Way configuration file, save your work and close the e*Way editor.

6 Click **OK** to close the **e*Way Properties** dialog box.

11.9.2 Step 2: Create the Internal_Eater Collaboration

The **Internal_Eater** Collaboration routes the data without changing its format.

Internal_Eater Collaboration Properties Setup

You must set up the Collaboration and Collaboration Rules Properties for the **Internal_Eater** Component in the Enterprise Manager GUI.

To create and configure the Internal_Eater Collaboration Rule

1 Create a new Collaboration Rule named **Internal_Eater**.

2 From Internal_Eater Collaboration Rule properties, select the **General** tab. Configure as shown in Table 79.

Table 79 Internal_Order_Eater CR configuration - General Tab

Section	Value
Service	PassThrough

3 Select the **Subscriptions** tab. Select **eX_to_eBPM** and move to the right pane.

4 Select the **Publications** tab. Select **eX_External_Evt** and move to the right pane.

To create and configure the Internal_Order_Eater Collaboration

1 Select the **Internal_Eater** e*Way.

2 Create a new Collaboration named **Internal_Eater**.

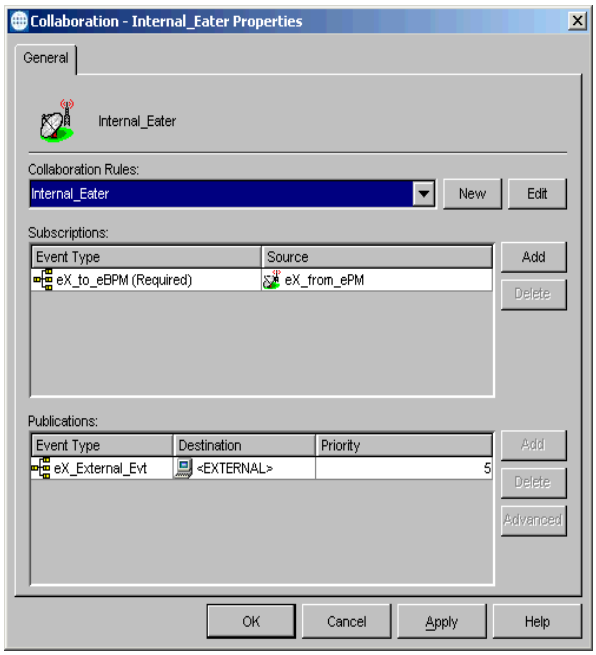
3 Configure the Internal_Eater Collaboration properties using Table 80.

Table 80 Internal_Eater Collaboration configuration

Section	Value
Collaboration Rule	Internal_Eater
Subscriptions	Event Type: eX_to_eBPM Source: eX_from_ePM
Publications	Event Type: eX_External_Evt Destination: <EXTERNAL>

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 78.

Figure 78 Internal_Eater Collaboration Properties



11.10 Configure the Internal_Response_Feeder e*Way

This component feeds the purchase order response to e*Xchange to be sent to the Retailer Trading Partner.

The e*Xchange Internal_Response_Feeder e*Way

The e*Xchange example simulates the publication of an electronic purchase order response to a trading partner. This file, which is already in RosettaNet format, is picked up by a file e*Way and moved into the e*Xchange system.

Configuration Steps

Follow these steps to configure the Internal_Response_Feeder e*Way.

1 Create and configure the e*Way.

2 Create the Collaboration.

11.10.1 Step 1: Create and configure the Internal_Response_Feeder e*Way

1 Create an e*Way called **Internal_Response_Feeder**.

2 In the **e*Way Properties** dialog box, in the **Executable file** area of the **General** tab, browse for **stcewfile.exe.**

3 In the **e*Way Properties** dialog box, in the **Configuration file** area of the **General** tab, click **New**.

4 Configure the **Internal_Response_Feeder** e*Way parameters using Table 81.

Table 81 Internal_Response_Feeder e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\Demos\RosettaNet\input\response
	MultipleRecordsPerFile	NO
	(All others)	(Default)
Performance Testing	(All)	(Default)

5 When finished editing the e*Way configuration file, save your work and close the e*Way editor.

6 Click **OK** to close the **e*Way Properties** dialog box.

11.10.2 Step 2: Create the Internal_Response_Feeder Collaboration

The **Internal_Response_Feeder** Collaboration must prepare the data coming into e*Xchange. How complicated this task is depends on the state of the data before the **Internal_Response_Feeder** Collaboration processes it.

The **Internal_Response_Feeder** Collaboration must do the following:

- convert the data to base 64 encoding
- populate the required nodes in the e*Xchange Event sent to e*Xchange for processing

Convert the Event to Base 64 Encoding

The **Internal_Response_Feeder** Collaboration must ensure that the data going into e*Xchange doesn't include any characters that cause problems for the XML structure of the standard e*Xchange Event (for example, characters that are the same as the XML control characters). This is done by converting the entire message to base 64 encoding using the Monk function **raw->base64**, before copying it to the payload node of the **eX_Standard_Event** ETD.

Populate the Required e*Xchange Nodes

In addition to copying the base 64 encoded message to the payload node of the TP_EVENT portion of the e*Xchange standard Event, you must provide e*Xchange with tracking information about the message.

The e*Xchange Internal_Response_Feeder CRS

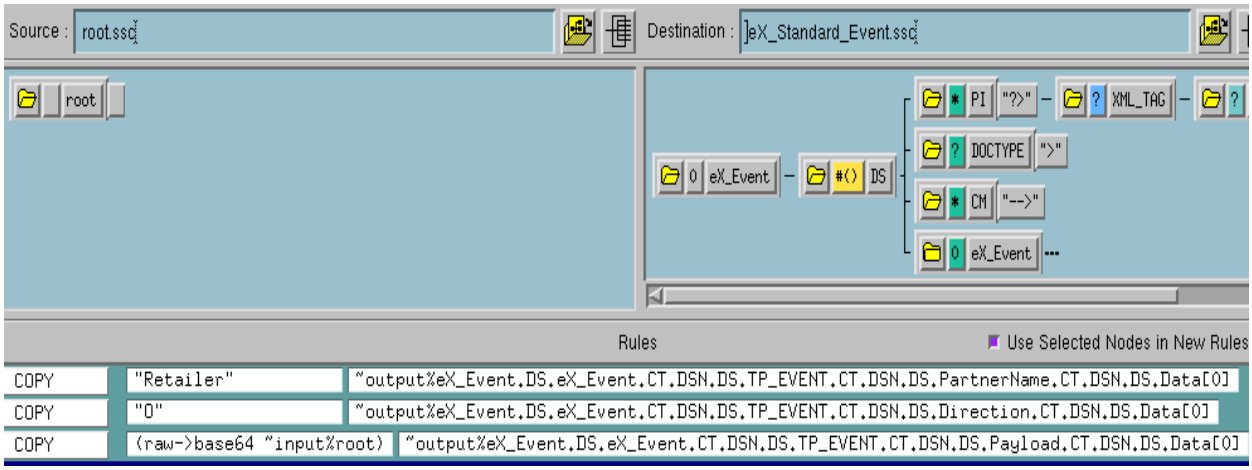
The **Internal_Response_Feeder.tsc** CRS does the following:

- Converts the RosettaNet message to base 64 encoding, and copies it to the Payload node of the TP_EVENT section of the e*Xchange standard Event.
- Copies "O" for outbound to the direction node of the TP_EVENT section.
- Copies the trading partner logical name "Retailer" to the PartnerName node of the TP_EVENT section.

To create and configure the Internal_Response_Feeder Collaboration Rule Script

1. Open the Collaboration Editor.
2. Create a new Collaboration Rules script named **Internal_Response_Feeder.tsc**. The Source Event Type Definition is **root.ssc**. The Destination Event Type Definition is **eX_Standard_Event.ssc**.
3. Add the rules shown in Figure 79.

Figure 79 Internal_Response_Feeder.tsc



Internal_Response_Feeder Collaboration Properties Setup

Once the CRS has been created, you must set up the Collaboration and Collaboration Rules Properties for the **Internal_Response_Feeder** Component in the Enterprise Manager GUI.

To create and configure the Internal_Response_Feeder Collaboration Rule

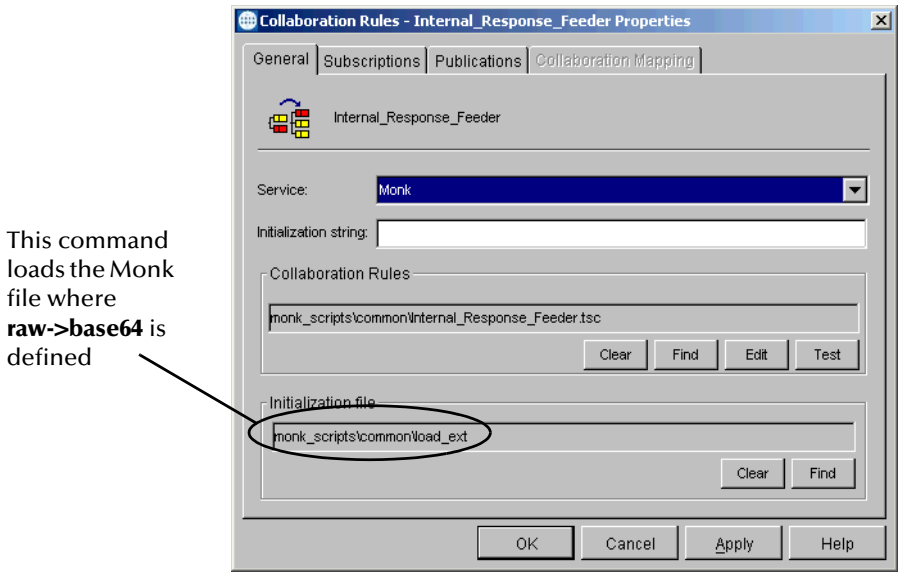
1. Create a new Collaboration Rule named **Internal_Response_Feeder**.
2. From **Internal_Response_Feeder** Collaboration Rule properties, select the **General** tab. Configure as shown in Table 82.

Table 82 Internal_Response_Feeder CR Configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	Internal_Response_Feeder
Initialization File	monk_scripts\common\load_ext

Important: *To use the Monk function* ***raw->base64****, you must make sure the file containing this function has been loaded.*

Figure 80 Internal_Response_Feeder Collaboration Rules Properties Dialog Box



3 Select the **Subscriptions** tab. Select **eX_External_Evt** and move it to the right pane.

4 Select the **Publications** tab. Select **eX_to_ePM** and move it to the right pane.

To create and configure the Internal_Response_Feeder Collaboration

1 Select the **Internal_Response_Feeder** e*Way.

2 Create a new Collaboration named **Internal_Response_Feeder**.

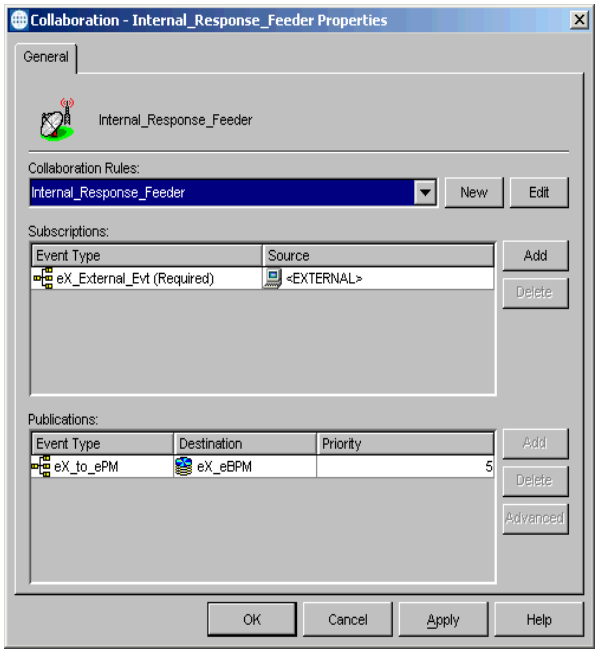
3 Configure the Internal_Response_Feeder Collaboration properties using Table 83.

Table 83 Internal_Response_Feeder Collaboration Configuration

Section	Value
Collaboration Rules	Internal_Response_Feeder
Subscriptions	Event Type: eX_External_Evt Source: <EXTERNAL>
Publications	Event Type: eX_to_ePM Destination: eX_eBPM

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 81.

Figure 81 Internal_Response_Feeder Collaboration Properties



11.11 Configure the TP_Response_Eater e*Way

The component (e*Way or BOB) sends the message to the external system.

The e*Xchange TP_Response_Eater e*Way

This example simulates the publication of the message to the external system.

Configuration Steps

Follow these steps to configure the TP_Response_Eater e*Way.

1 Create the configuration file.

2 Create the Collaboration.

11.11.1 Step 1: Create and configure the TP_Response_Eater e*Way

1 Create an e*Way called **TP_Response_Eater**.

2 In the **e*Way Properties** dialog box, **General** tab, in the **Executable file** area, browse for **stcewfile.exe**.

3 In the **e*Way Properties** dialog box, **General** tab, in the **Configuration file** area, click **New**.

4 Configure the **TP_Response_Eater** e*Way parameters using Table 84.

Table 84 TP_Response_Eater e*Way Parameters

Screen	Parameter	Setting
General Settings	AllowIncoming	NO
	AllowOutgoing	YES
Outbound (send) settings	OutputDirectory	<eGate>\Demos\RosettaNet\output\Response_Out\TP
	OutputFileName	order%d.dat
	MultipleRecordsPerFile	NO
	(All others)	(Default)
Poller (inbound) settings	(All)	(Default)
Performance Testing	(All)	(Default)

5 When finished editing the e*Way configuration file, save your work and close the e*Way editor.

6 Click **OK** to close the **e*Way Properties** dialog box.

11.11.2 Step 2: Create the TP_Response_Eater Collaboration

The **TP_Response_Eater** Collaboration must prepare the data leaving the e*Xchange system. How complicated this task is depends on the state of the data before the **TP_Response_Eater** Collaboration processes it.

The **TP_Response_Eater** Collaboration must do the following:

- put the data into the appropriate format
- convert the data to raw data

The e*Xchange TP_Response_Eater CRS

The CRS, **TP_Response_Eater.tsc** checks that the message is for the Wholesaler Trading Partner (Retailer). If it is, it converts the RosettaNet message to raw data, and copies it from the Payload node of the TP_EVENT section of the e*Xchange standard Event to the output ETD.

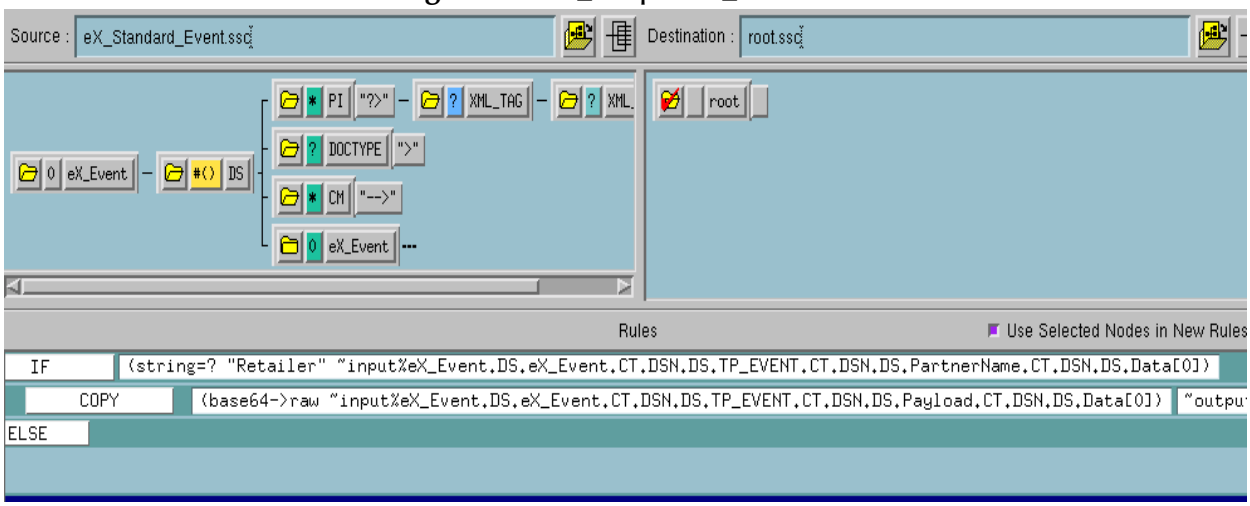
To create and configure the TP_Response_Eater Collaboration Rule Script

1 Open the Collaboration Editor.

2 Create a new Collaboration Rules script named **TP_Response_Eater.tsc**. The Source Event Type Definition is **eX_Standard_Event.ssc**. The Destination Event Type Definition is **root.ssc**.

3 Add the rule shown in Figure 82.

Figure 82 TP_Response_Eater.tsc



TP_Response_Eater Collaboration Properties Setup

Once the CRS has been created, you must set up the Collaboration and Collaboration Rules Properties for the **TP_Response_Eater** Component in the Enterprise Manager GUI.

To create and configure the TP_Response_Eater Collaboration Rule

1. Create a new Collaboration Rule named **TP_Response_Eater**.
2. From Internal_Order_Eater Collaboration Rule properties, select the **General** tab. Configure as shown in Table 85.

Table 85 TP_Response_Eater CR configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	TP_Response_Eater
Initialization File	monk_scripts\common\load_ext

Important: *To use the Monk function* ***base64->raw****, you must make sure the file containing this function has been loaded.*

3. Select the **Subscriptions** tab. Select **eX_HTTP** and move to the right pane.
4. Select the **Publications** tab. Select **eX_External_Evt** and move to the right pane.

To create and configure the TP_Response_Eater Collaboration

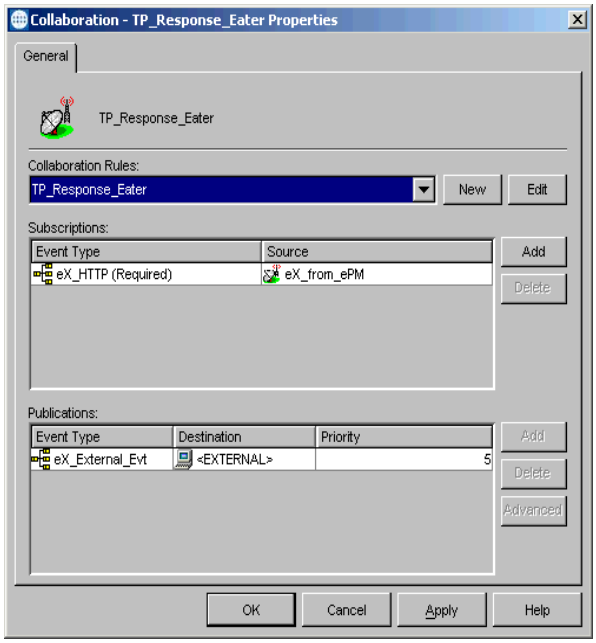
1. Select the **TP_Response_Eater** e*Way.
2. Create a new Collaboration named **TP_Response_Eater**.
3. Configure the TP_Response_Eater Collaboration properties using Table 86.

Table 86 TP_Response_Eater Collaboration configuration

Section	Value
Collaboration Rules	TP_Response_Eater
Subscriptions	Event Type: eX_HTTP Source: eX_from_ePM
Publications	Event Type: eX_External_Evt Destination: <EXTERNAL>

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 83.

Figure 83 TP_Response_Eater Collaboration Properties



11.12 Configure the TP_Response_Feeder e*Way

This component feeds the data that was sent to the Retailer Trading Partner into e*Xchange.

The e*Xchange TP_Response_Feeder e*Way

The e*Xchange example simulates the publication of an electronic purchase order from a trading partner. This file, which is already in RosettaNet format, is picked up by a file e*Way and moved into the e*Xchange system.

Configuration Steps

Follow these steps to configure the TP_Response_Feeder e*Way.

1 Create and configure the e*Way.

2 Create the Collaboration.

11.12.1 Step 1: Create and Configure the TP_Response_Feeder e*Way

1 Create an e*Way called **TP_Response_Feeder**.

2 In the **e*Way Properties** dialog box, in the **Executable file** area of the **General** tab, browse for **stcewfile.exe**.

3 In the **e*Way Properties** dialog box, in the **Configuration file** area of the **General** tab, click **New**.

4 Configure the **TP_Response_Feeder** e*Way parameter using Table 87.

Table 87 TP_Response_Feeder e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\Demos\RosettaNet\Output\Response_Out\TP
	InputFileMask	*.dat
	MultipleRecordsPerFile	NO
	(All others)	(Default)
Performance Testing	(All)	(Default)

5 When finished editing the e*Way configuration file, save your work and close the e*Way editor.

6 Click **OK** to close the **e*Way Properties** dialog box.

11.12.2 Step 2: Create the TP_Response_Feeder Collaboration

The **TP_Response_Feeder** Collaboration must prepare the data coming into e*Xchange. How complicated this task is depends on the state of the data before the **TP_Response_Feeder** Collaboration processes it.

The **TP_Response_Feeder** Collaboration must do the following:

- convert the data to base 64 encoding
- populate the required nodes in the e*Xchange Event sent to e*Xchange for processing

Convert the Event to Base 64 Encoding

The **TP_Response_Feeder** Collaboration must ensure that the data going into e*Xchange doesn't include any characters that cause problems for the XML structure of the standard e*Xchange Event (for example, characters that are the same as the XML control characters). This is done by converting the entire message to base 64 encoding using the Monk function **raw->base64**, before copying it to the payload node of the **eX_Standard_Event** ETD.

Populate the Required e*Xchange Nodes

In addition to copying the base 64 encoded message to the payload node of the TP_EVENT portion of the e*Xchange standard Event, you must provide e*Xchange with tracking information about the message.

The e*Xchange TP_Response_Feeder CRS

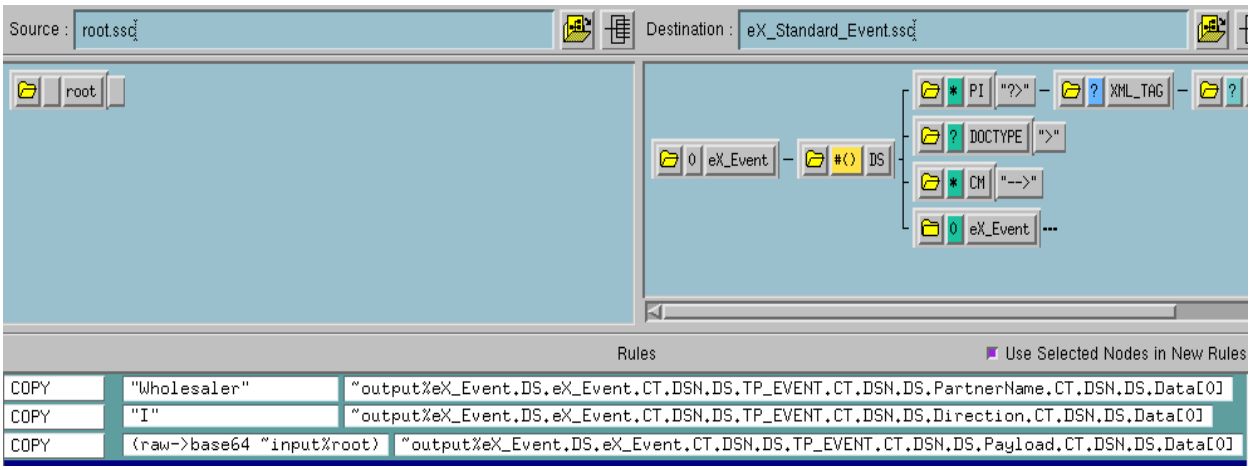
The **TP_Response_Feeder.tsc** CRS does the following:

- Converts the RosettaNet message to base 64 encoding, and copies it to the Payload node of the TP_EVENT section of the e*Xchange standard Event.
- Copies "I" for outbound to the direction node of the TP_EVENT section.
- Copies the trading partner logical name "Wholesaler" to the PartnerName node of the TP_EVENT section.

To create and configure the TP_Response_Feeder Collaboration Rule Script

1. Open the Collaboration Editor.
2. Create a new Collaboration Rules script named **TP_Response_Feeder.tsc**. The Source Event Type Definition is **root.ssc**. The Destination Event Type Definition is **eX_Standard_Event.ssc**.
3. Add the rules shown in Figure 84.

Figure 84 TP_Response_Feeder.tsc



TP_Response_Feeder Collaboration Properties Setup

Once the CRS has been created, you must set up the Collaboration and Collaboration Rules Properties for the **TP_Response_Feeder** Component in the Enterprise Manager GUI.

To create and configure the TP_Response_Feeder Collaboration Rule

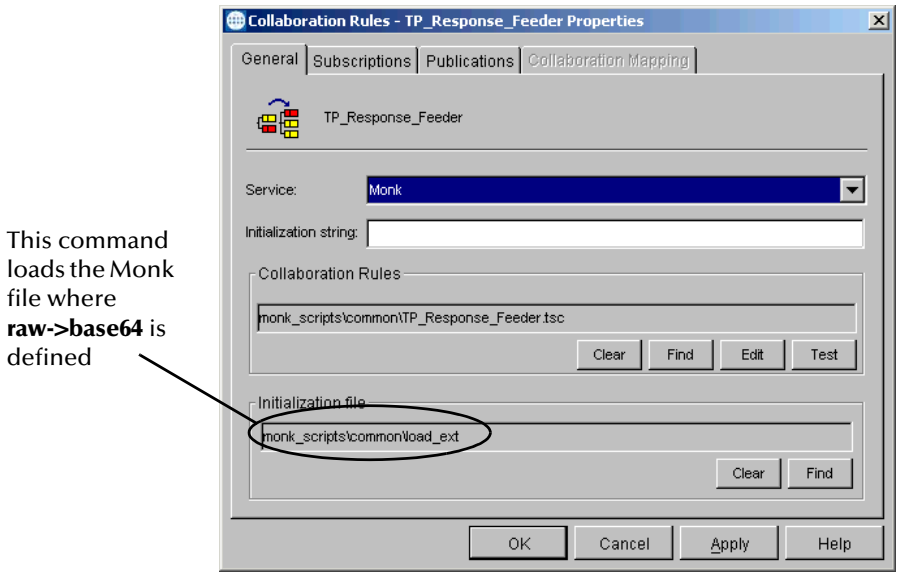
1. Create a new Collaboration Rule named **TP_Response_Feeder**.
2. From TP_Response_Feeder Collaboration Rule properties, select the **General** tab. Configure as shown in Table 88.

Table 88 TP_Response_Feeder CR configuration - General Tab

Section	Value
Service	Monk
Collaboration Rule	TP_Response_Feeder
Initialization File	monk_scripts\common\load_ext

Important: *To use the Monk function raw->base64, you must make sure the file containing this function has been loaded.*

Figure 85 TP_Response_Feeder Collaboration Rules Properties Dialog Box



3 Select the **Subscriptions** tab. Select **eX_External_Evt** and move it to the right pane.

4 Select the **Publications** tab. Select **eX_to_ePM** and move it to the right pane.

To create and configure the Internal_Order_Feeder Collaboration

1 Select the **TP_Response_Feeder** e*Way.

2 Create a new Collaboration named **TP_Response_Feeder**.

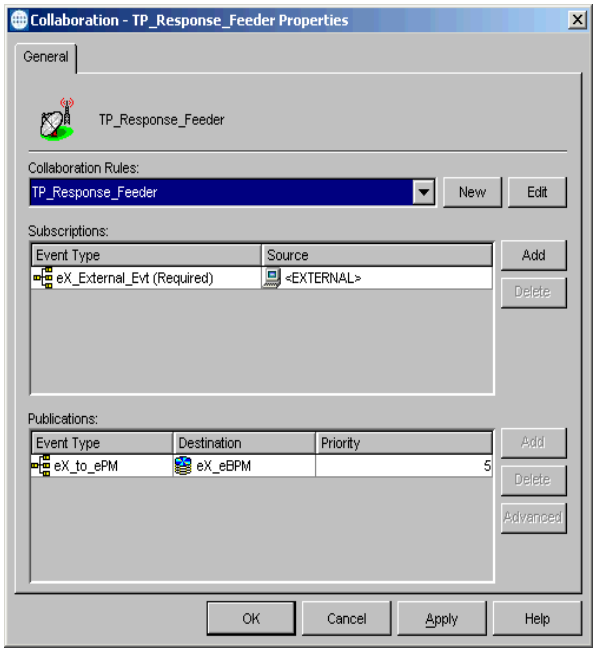
3 Configure the TP_Response_Feeder Collaboration properties using Table 89.

Table 89 TP_Response_Feeder Collaboration configuration

Section	Value
Collaboration Rules	TP_Response_Feeder
Subscriptions	Event Type: eX_External_Evt Source: <EXTERNAL>
Publications	Event Type: eX_to_ePM Destination: eX_eBPM

Verify the information in the **Collaboration Properties** dialog box as shown in Figure 86.

Figure 86 TP_Response_Feeder Collaboration Properties



11.13 Configure the eX_ePM e*Way

The **eX_ePM** e*Way requires only minimal configuration. You must give it the logon information for the e*Xchange database.

To configure the eX_ePM configuration file

1. In the **eX_ePM** e*Way properties, select the **General** tab.
2. In the **Configuration File** area, click **Edit**.
3. Configure the parameters as shown in Table 90.

Table 90 eX_ePM e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Communication Setup	(All)	(Default)
Monk Configuration	(All)	(Default)

Table 90 eX_ePM e*Way Parameters

Screen	Parameter	Setting
Database Setup	Database Name	(service name of the e*Xchange database)
	User name	ex_admin
	Password	ex_admin
	(All others)	(Default)

To set the file names correctly

1 In **<egate>\Demos\RosettaNet\input\order**, change the name of the **orders.~in** file to **orders.fin**.

2 In **<egate>\Demos\RosettaNet\input\order_response**, change the name of the **order_response.~in** file to **order_response.fin**.

That completes the data setup. The next step is to run the scenario.

11.14 Running the Scenario

There are five parts to running the scenario:

A The Retailer trading partner sends the purchase order to the Wholesaler trading partner.

B The Wholesaler trading partner processes the purchase order message received from the Retailer trading partner

C The Wholesaler trading partner sends the acknowledgment back to the Retailer trading partner

D The Wholesaler trading partner sends the response message back to the Retailer trading partner

E The Retailer trading partner sends the acknowledgment back to the Wholesaler trading partner

Parts A, B, and C are performed in **"To process the purchase order message"**. Parts D and E are performed in **"To send the response message" on page 197**.

To process the purchase order message

1 Rename the file **<egate>\Demos\RosettaNet\input\Orders.~in** to **Orders.fin**.

Once your data file is in place, start the following e*Gate components:

2 Start the Control Broker. At the command line, enter:

```
stccb.exe -rh localhost -rs RosettaNet -ln localhost_cb -un
Administrator -up STC
```

3 Open the e*Gate Monitor and select the RosettaNet schema.

4 Start the **eX_ePM** e*Way

This starts the e*Xchange engine.

5 Start the **Internal_Order_Feeder** e*Way

This e*Way retrieves the purchase order from the internal system and sends it to the e*Xchange Partner Manager.

6 Look in the **<egate>\Demos\RosettaNet\Input\Order** folder. The file name changes from **Order.fin** to **Order.~in** as the file is picked up.

7 Start the **TP_Order_Eater** e*Way.

This e*Way sends the purchase order to a file which is then retrieved and sent to the Wholesaler trading partner.

8 Look in the **<egate>\Demos\RosettaNet\Output\Order_Out\TP** folder. The file **Order1.dat** is created.

9 Start the **TP_Order_Feeder** e*Way.

This e*Way sends the message to the Wholesaler trading partner.

10 Look in the **<egate>\Demos\RosettaNet\Output\Order_Out\TP** folder. The file name changes from **Order1.dat** to **Order1.~in** as the file is picked up.

11 Start the **Internal_Eater** e*Way.

This e*Way sends the message to a file (simulating sending to an internal system).

12 Look in the **<egate>\Demos\RosettaNet\Output\Order_Out** folder. The file **Order1.dat** is created.

13 Start the **TP_Response_Eater** e*Way.

This e*Way sends the purchase order acknowledgment to a file which is then retrieved and sent to the Retailer trading partner.

14 Look in the **<egate>\Demos\RosettaNet\Output\Response_Out\TP** folder. The file **Order1.dat** is created.

15 Start the **TP_Response_Feeder** e*Way

This e*Way sends the purchase order acknowledgment to the Retailer trading partner.

16 Look in the **<egate>\Demos\RosettaNet\Output\Response_Out\TP** folder. The file name changes from **Order1.dat** to **Order1.~in** as the file is picked up.

The message is processed by **Internal_Eater** e*Way. This e*Way sends the message to a file (simulating sending to an internal system).

Note: *Look in the **<egate>\Demos\RosettaNet\Output\Order_Out** folder. The file **Order1.dat** is created.*

That completes sending the purchase order. You can view the results in Message Tracking, in e*Xchange Web interface.

Viewing the Results in Message Tracking

You can view the results of the message processing by using the Message Tracking feature of the e*Xchange Partner Manager.

Message Tracking shows two entries for the incoming message. This is because a control message is sent out immediately, and a response message is sent out later. These two responses to the trading partner are tracked separately.

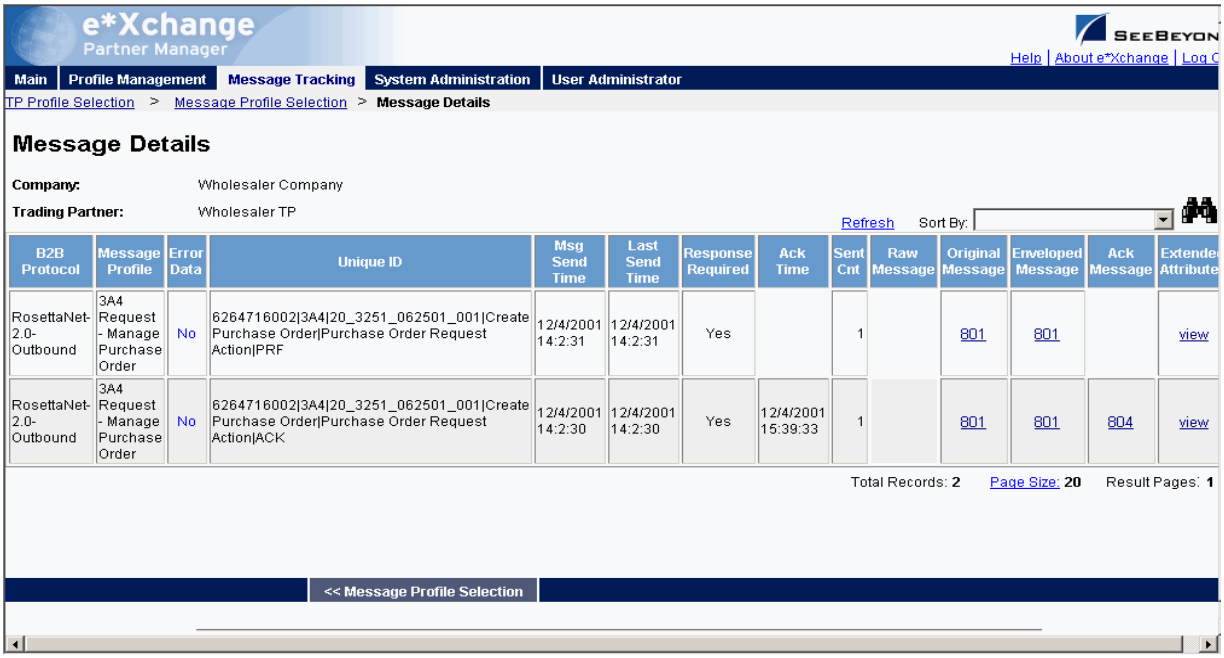
To view the outbound message in Message Tracking for the Wholesaler Trading Partner

1. From the e*Xchange Web interface, **Main** page, select **Message Tracking**.

   The **TP Profile Selection** page appears.
2. In the **Company Profile** field, select **Wholesaler Company**.
3. In the **Trading Partner Profile** field, select **Wholesaler TP**.
4. In the **eBusiness Protocol** field, select **RosettaNet**.
5. In the **Direction** field, select **Outbound**.
6. Click the **Message Profile Selection**.
7. Select the **3A4 Request - Manage Purchase Order** message.
8. Click the **Message Details** link to view the resulting list.

   The results are shown in Figure 87.

Figure 87 Message Tracking: Outbound



As shown in Figure 87, e*Xchange records two entries for the message. One entry is for the original message, for which a response message is sent. The other entry is for the acknowledgment message.

For one entry, the **Ack Message** column has a link to the message information. Click it to view the acknowledgment message.

Later, when the response message is sent out, you are able to view it in Message Tracking. For the moment, the **Ack Message** column is not showing a link for the other message, since the response has not been sent out yet.

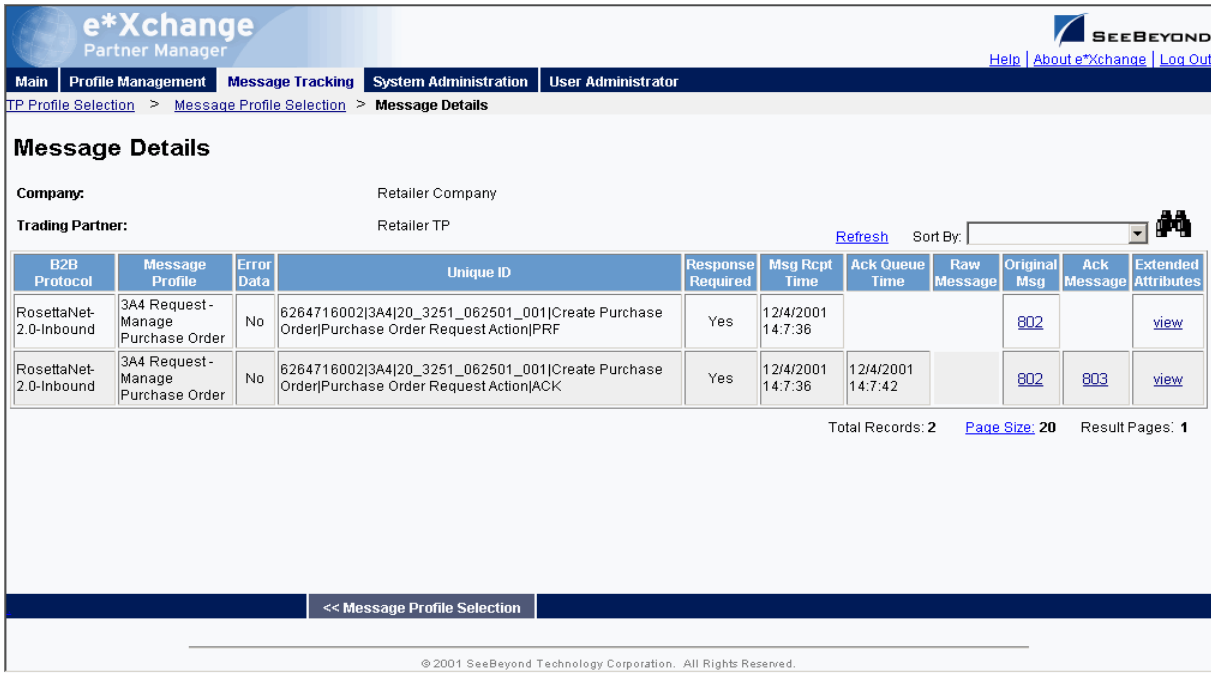
To view the inbound message in Message Tracking for the Retailer Trading Partner

1. From the e*Xchange Web interface, **Main** page, select **Message Tracking**.

   The **TP Profile Selection** page appears.
2. In the **Company Profile** field, select **Retailer Company**.
3. In the **Trading Partner Profile** field, select **Retailer TP**.
4. In the **eBusiness Protocol** field, select **RosettaNet**.
5. In the **Direction** field, select **Inbound**.
6. Click the **Message Profile Selection**.
7. Select the **3A4 Request - Manage Purchase Order** message.
8. Click the **Message Details** link to view the resulting list.

   The results are shown in Figure 88.

Figure 88 Message Tracking: Inbound



As shown in Figure 88, e*Xchange records two entries for the message. One entry is for the original message, for which a response message is sent later. The other entry is for the acknowledgment message.

11.15 Sending the Response

The next step is to send the response message.

To send the response message

1. Rename the file **<egate>\Demos\RosettaNet\Input\Order_Response\order_response.~in** to **order_response.fin**.
2. In the e*Gate Monitor, start the **Internal_Response_Feeder** e*Way.
3. Look in the **<egate>\Demos\RosettaNet\input\order_response** folder. The file name changes from **order_response.fin** to **order_response.~in** as the file is picked up.

   The message is processed by **TP_Response_Eater** e*Way. This e*Way sends the purchase order response to a file which is then retrieved and sent to the Retailer trading partner.
4. Look in the **<egate>\Demos\RosettaNet\Output\Response_Out\TP** folder. The file **Order2.dat** is created. This is immediately renamed in the following step.

   The message is processed by **TP_Response_Feeder** e*Way. This e*Way sends the purchase order response to the Retailer trading partner.
5. Look in the **<egate>\Demos\RosettaNet\Output\Response_Out\TP** folder. The file name changes from **Order2.dat** to **Order2.~in** as the file is picked up.

   The message is process by **Internal_Eater** e*Way. This e*Way sends the message to a file (simulating sending to an internal system).
6. Look in the **<egate>\Demos\RosettaNet\Output\Order_Out** folder. The file **Order2.dat** is created.

   The message is processed by **TP_Order_Eater** e*Way. This e*Way sends the purchase order response acknowledgment to a file which is then retrieved and sent to the Wholesaler trading partner.
7. Look in the **<egate>\Demos\RosettaNet\Output\Order_Out\TP** folder. The file **Order2.dat** is created.

   The message is processed by **TP_Order_Feeder** e*Way. This e*Way sends the purchase order response acknowledgment to the Wholesaler trading partner.
8. Look in the **<egate>\Demos\RosettaNet\Output\Order_Out\TP** folder. The file name changes from **Order2.dat** to **Order2.~in** as the file is picked up.

That completes the second part of the exercise. You can view the results in Message Tracking.

Viewing the Results in Message Tracking

You can view the results of the message processing in Message Tracking.

To view the association of the response message to the original outbound message in Message Tracking for the Retailer Trading Partner

1 From e*Xchange Web Interface, **Main** page, select **Message Tracking**.

The **TP Profile Selection** page appears.

2 In the **Company Profile** field, select **Retailer Company**.

3 In the **Trading Partner Profile** field, select **Retailer TP**.

4 In the **eBusiness Protocol** field, select **RosettaNet**.

5 In the **Direction** field, select **Inbound**.

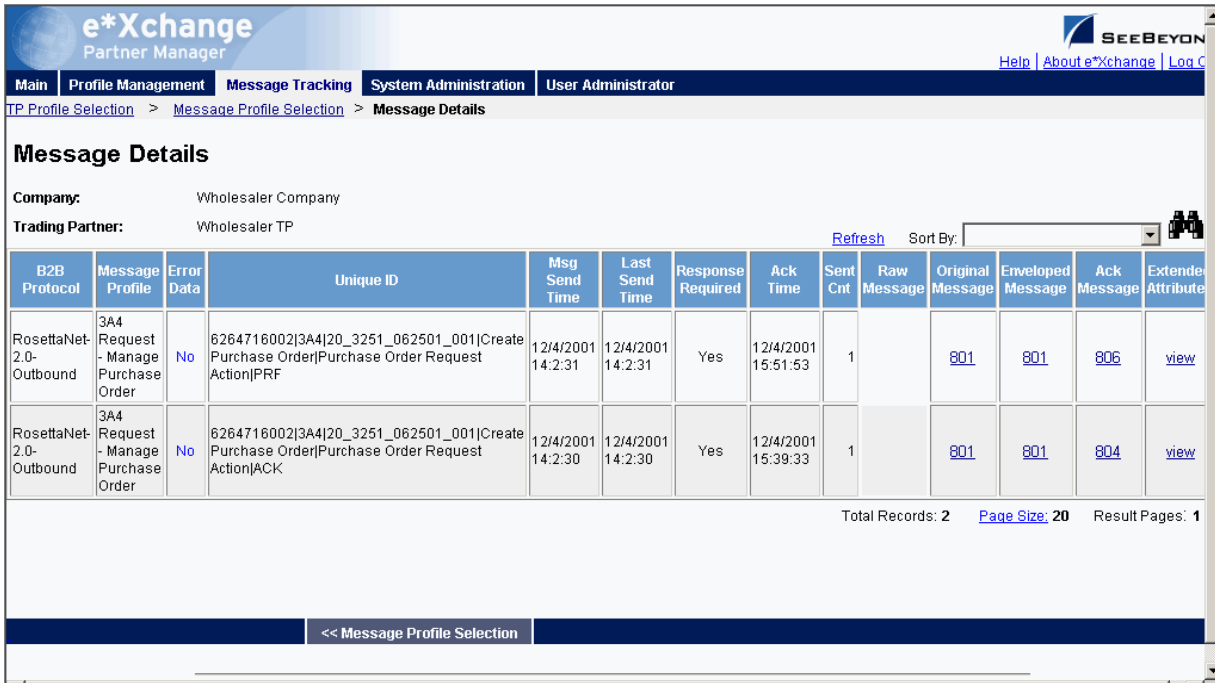
6 Click the **Message Profile Selection**.

7 Select the **3A4 Request - Manage Purchase Order** message.

8 Click the **Message Details** link to view the resulting list.

The results are shown in Figure 89.

Figure 89 Message Tracking: Outbound Completed



Notice that both entries now have responses available for viewing: one is the acknowledgment message, the other is the full response message.

To view the association of the response message to the original inbound message in Message Tracking for the Retailer trading partner

1 From the e*Xchange Web interface, **Main** page, select **Message Tracking**.

The **TP Profile Selection** page appears.

2 In the **Company Profile** field, select **Retailer Company**.

3 In the **Trading Partner Profile** field, select **Retailer TP**.

4 In the **eBusiness Protocol** field, select **RosettaNet**.

5 In the **Direction** field, select **Inbound**.

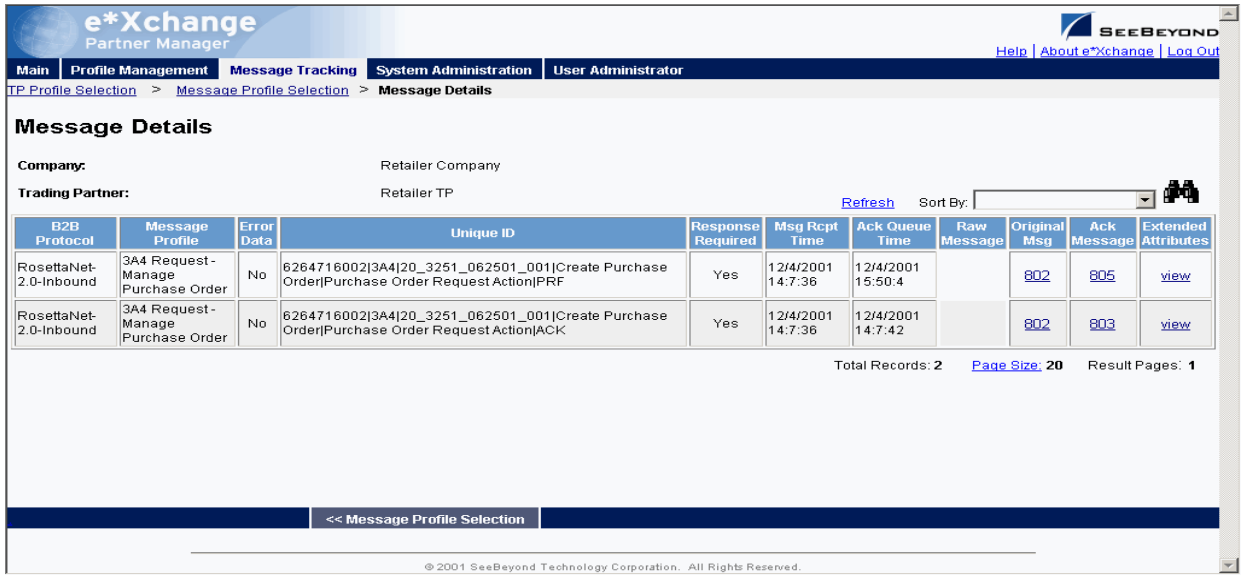
6 Click the **Message Profile Selection**.

7 Select the **3A4 Request - Manage Purchase Order** message.

8 Click the **Message Details** link to view the resulting list.

The results are shown in Figure 90.

Figure 90 Message Tracking: Inbound Completed



Notice that both entries now have responses associated with them: one is the acknowledgment message, the other is the full response message.

11.16 Editing the Data Files

Before rerunning the scenario, you must make sure that the unique ID in the order file matches that in the response file, and that both files have the expected filename and extension.

Knowing how to set these values also gives you the capability to reset the unique ID to an appropriate new value so that you can run the scenario multiple times.

To ensure the unique ID in both files matches

1 Open up the file **order.~in** (in the **<egate>\Demos\RosettaNet\input\order** folder) in a text editor such as Notepad or Wordpad.

2 Search for the following string, which is the unique ID in the files provided:

```
20_3251_062501_001
```

3 Replace that string with the following string:

```
20_3251_062501_002
```

4 Save and close.

5 Open up the file order_response.~in (in the **<egate>\Demos\RosettaNet\input\order_response** folder) in a text editor such as Notepad or Wordpad.

6 Repeat steps 2 through 4 for this file. Make sure that the string is updated throughout the file.

Note: *The last three digits of the unique ID indicate that this is the first instance for this date. For a second and subsequent running of this scenario, increment the last three digits: 002, 003, and so forth. In each case, make sure that the value is the same in both files.*

Chapter 12

Advanced Configuration

This chapter provides a information on manually creating a Validation Collaboration and adding a custom communication protocol.

12.1 Manually Creating a Validation Rules Collaboration

Validation Collaborations can be created for X12 and UN/EDIFACT using the Validation Collaborations Rules Builder. However, it is possible to build the Validation Collaboration manually.

Validation Collaborations for RosettaNet must be created manually.

12.1.1 Creating a Validation Rules Collaboration for X12 or UN/EDIFACT

The steps required to create the Validation Collaboration are as follows:

1. Create the validation ETD.
2. Create the validation collaboration rules script.

Creating the Validation ETD

The Event Type Definition used for the Validation Collaboration is based on the ETD for the message type. For example, if you were working with X12 4010, then base your validation ETD on X12_4010_100.ssc.

To create a validation ETD

1. Open the generic ETD (for example, X12_4010_100.ssc).
2. Save as a new ETD with a different name (for example, X12_4010_valid_100.ssc).
3. The nodes required for this ETD are those between **ST** and **SE** (ST and SE are required in the structure) for X12 and between **UNH** and **UNT** (UNH and UNT are required in the structure) for UN/EDIFACT. Other nodes should be deleted.
4. Add a node above ST or UNH called **Delimiter**.
5. From the **File** menu, select **Default Delimiters**. Set the following:

Table 91 X12 below Version 4020

Level	Delimiter
1	[2]
2	[0]
3	[1]

Table 92 X12 Version 4020 and above

Level	Delimiter
1	[3]
2	[0]
3	[2]
4	[1]

Table 93 UN/EDIFACT

Level	Delimiter
1	[3]
2	[0]
3	[2]
4	[1]

6 Save the ETD.

Creating the Validation Collaboration

The validation collaboration rule script is used to validate the incoming or outgoing message. The minimum requirement for this script is to create a unique identifier. Additional functionality that can be used as required, for example, to add user defined tests on the data and to send a 997 response.

There is a variable named **error** that is used to determine whether the message has been validated. The default value of error is #f, and this value should be set to #t if an error has occurred and the message should not be processed.

To create the Validation Collaboration

1 Create a new Collaboration Rule Script using the structure created in **"To create a validation ETD" on page 201** as the source ETD. Leave the destination ETD blank.

2 Add a line of code to create a unique identifier for the message. For example:

```
(define unique_id (strftime "%Y%m%d%H$M%S" (time)))
```

Creating a unique identifier for the message is the minimum requirement for the Validation Collaboration.

To reject the message

Set the error variable value to #t if an error has occurred and the message should not be processed using the following line of code.

```
(set! error #t)
```

To send a 997 response

Add the following line of code if you want to send a 997 response, or if you want to reject the message:

```
(define add_997 (ux-track-997-error (list "AK2" (get <ST01 path>) (get <ST02 path>)))
```

where

- <ST01 path> is the path to the ST01 node
- <ST02 path> is the path to the ST02 node

To send a 997 response for an error

When the message is not successfully validated the 997 response can contain AK3 and AK4 information. AK3 contains information about the segment and AK4 contains information about the element in the segment that caused a problem.

To set AK3 information

Add the following line of code:

```
(define add_997 (ux-track-997-error (list "AK3" <SegIDCode> <SegPos> <LoopID>
<SegmentErrorCode>)))
```

where

- <SegIdCode> is the segment ID code, for example "BGN".
- <SegPos> is the segment position. For example, the first segment has position "1".
- <LoopID> is the loop identifier.
- <SegmentErrorCode> is a user defined error identifier.

To set AK4 information

Add the following line of code:

```
(define add_997 (ux-track-997-error (list "AK4" <PosInSegment> <DataElementRefNumber>
<DataElementErrorCode> <CopyOfDataElement>)))
```

where

- <PosInSegment> is the element position within the segment.
- <DataElementRefNumber> is the data element reference number as defined for X12.
- <DataElementErrorCode> is a user defined error identifier.
- <CopyOfDataElement> is an optional parameter allowing you to send the data from the element with the error message.

To send information about multiple errors

Error information can be defined in a variable named **error_data**. This needs to be in the format "num^description", using the ^ character as a delimiter. To send information about multiple errors each number/description pair needs to be delimited by the ~ character. For example:

```
"1^description1~2^description2~3^description3"
```

12.1.2 Creating a Validation Rules Collaboration for RosettaNet

The inbound event for a RosettaNet validation collaboration must represent the format of the Service Content. The Monk ETD files for each RosettaNet PIP Service Content are found in the monk_scripts/templates directory, once you have installed them from the add-on section of the installation CD.

The validation Collaboration Rules Script can be given any name, however it is recommended that it represents what is being validated. For example, eX_ROS_Validate_3A4Request_11_SC.tsc is a RosettaNet 1.1validation script that checks the 3A4 Request service content. The extension for the validation script must be "tsc", and the script must be located in **monk_scripts/common/ROS/etc**.

To create the Validation Collaboration

1. Create a new Collaboration Rule Script using the required RosettaNet structure as the source ETD. Leave the destination ETD blank.
2. Add the required lines of code.

To reject the message

When creating a validation collaboration, the variable string, **error_data**, should be used to capture all the errors. This variable is defined globally by the calling script, (eX_ROS_main.dsc for RosettaNet 1.1,eX-ROS20-Outb-Main.dsc for RosettaNet 2.0 outbound messages, and eX-ROS20-Inb-Main.dsc for RosettaNet 2.0 inbound messages), so **set!** function should be used each time error_data is reset.

Setting the **error_data** variable to a value other than an empty string causes the message to be rejected. For example:

```
(set! error_data “5101^Missing City Name”)
```

If the variable error_data contains some error information, then the processing takes place with the assumption that there is at least one invalid entry in the Service Content and then the RosettaNet message is rejected. For RosettaNet 1.1, an Inbound message is rejected by sending out a negative Receipt Acknowledgment, and an Outbound message is rejected by sending out an internal failure.

For RosettaNet 2.0 Inbound messages, the validation collaboration should have the following set, in addition to error_data, if an invalid entry is found in the service content:

```
(set! error_code “UNP.SCON.VALERR”)
(set! error_comp “ServiceContent”)
```

If an invalid entry is found for a RosettaNet 2.0 Inbound message, then a Receipt Acknowledgment Exception is sent out if a Receipt Acknowledgment is expected. Also, the invalid message is stored in the e*Xchange database with the associated errors.

For RosettaNet 2.0 Outbound messages, the validation collaboration should set the error_data variable to contain any errors for invalid entries found in the service content. The invalid message is stored in the e*Xchange database with the associated errors. An internal failure message is sent out to the internal application.

To send information about multiple errors

A new error string should be appended to error_data if error_data already contains some errors. Each error must be separated by ~, and within each error, the code and description are separated by ^. For example, error_data may contain "5101^Missing City Name", and then another error is encountered, such as "5118^Invalid Revision Number". A string-append including a ~ separator should be used to be sure both errors are included in error_data. The resulting error_data string would then be

"5101^Missing City Name~5118^Invalid Revision Number"

Using the util-add-to-error function

The **util-add-to-error** function can be used to generate the error string. This function is described below.

Syntax

```
(util-add-to-error existing_error_str new_error_component)
```

Description

util-add-to-error appends the new error component to the existing error string and returns the new error string.

Parameters

Name	Type	Description
existing_error_str	string	The existing error string.
new_error_component	string	The new error component to be appended to the error string.

Return Values

string
Returns the new error string.

Example

The following example first test whether the city name is missing, and then tests if the revision number is invalid. This code assumes that two user defined functions (city-name-missing? and revision-number-invalid?) have been created to test the data.

```
(if (city-name-missing?)
    (set! error_data "5101^Missing City Name")
)
(if (revision-number-invalid?)
    (util-add-to-error (error_data "5118^Invalid Revision Number"))
)

=> sets error_data to "5101^Missing City Name~5118^Invalid Revision Number"
if both errors are found
```

Predefined Validation Scripts

There are some validation scripts included in the installation of e*Gate Schema for e*Xchange scripts. The validation scripts located in monk_scripts/common/ROS/etc for RosettaNet 1.1 are:

- eX_ROS_Validate_3A4Accept_11_SC.tsc
- eX_ROS_Validate_3A4Cancel_11_SC.tsc
- eX_ROS_Validate_3A4Change_11_SC.tsc
- eX_ROS_Validate_3A4Request_11_SC.tsc
- eX_ROS_Validate_PriceAndAvailabilityQuery.tsc
- eX_ROS_Validate_PriceAndAvailabilityResponse.tsc

These validation scripts refer to code files stored in the same location. The file, eX-validation-codes.monk, located in monk_library/eXchange contains reference variables to all the code files used. An example of a variable defined in this file is GLOBAL_COUNTRY_CODES_FILE, which corresponds to the codes file monk_scripts/common/ROS/etc/Global_Country_Codes. Additional code file references can be added to eX-validation-codes.monk, if necessary, for new validation script references. This monk file gets loaded on startup of e*Xchange.

There are no validation scripts provided for RosettaNet 2.0.

12.2 Adding a Custom Protocol

This section describes how you can define additional protocols to use with e*Xchange.

12.2.1 Adding a Custom Protocol for X12 or UN/EDIFACT

The steps required to create an additional protocol are as follows:

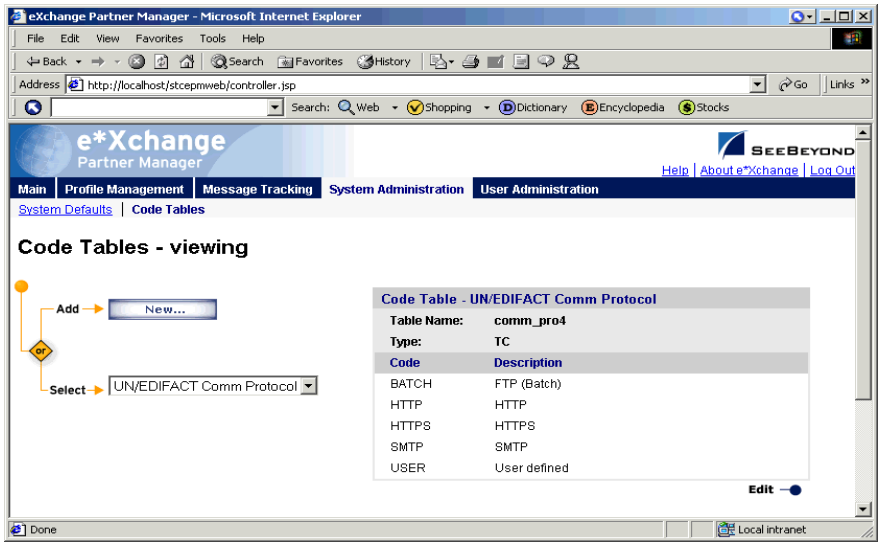
1. Add a Comm Protocol to the Code Table.
2. Add an Event Type for the protocol.
3. Update eX_from_ePM Collaboration Rule to publish the new Event Type.
4. Update eX_from_ePM Collaboration to publish the new Event Type to the eX_Trading_Port_Queue IQ.
5. Edit monk_library\eXchange\eX_ePM_Send_To_External.monk to set the output event.
6. Update eX_from_ePM.tsc to perform an iq-put with the new Event Type, if the new output event is received.

Step 1: Add a Comm Protocol to the Code Table

The UN/EDIFACT and X12 Comm Protocol in the Code table list the protocols that are currently available. Add an additional protocol and assign a name and description.

Figure 91 shows a protocol named USER.

Figure 91 Example Code Table for UN/EDIFACT



Step 2: Add an Event Type for the Protocol

Use the e*Gate Enterprise Manager GUI to create a new Event Type in eXSchema. For example, **eX_User**.

Step 3: Update eX_from_ePM Collaboration Rule

Update the eX_from_ePM Collaboration Rule to publish the Event Type created in Step 2.

Step 4: Update eX_from_ePM Collaboration

Update eX_from_ePM Collaboration to publish the new Event Type to the eX_Trading_Port_Queue IQ.

Figure 92 shows an example configuration using eX_User.

Figure 92 Example eX_from_ePM using eX_User

Step 5: Update eX_ePM_Send_To_External.monk

Edit monk_libray\eXchange\eX_ePM_Send_To_External.monk to set the output event. Add the following code within the case statement:

```
((<Comm Protocol>)
        (set! out_event "<Comm Protocol Ref>")
  )
```

where

- <Comm Protocol> defines the name given in the code table
- <Comm Protocol Ref> is a used defined name with exactly five characters

Example code for the USER protocol:

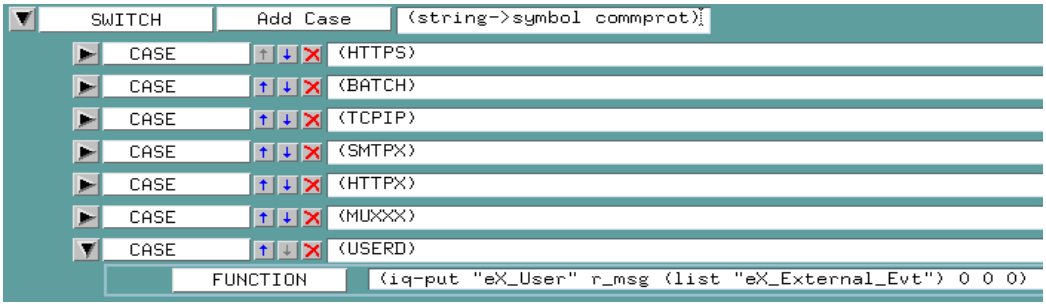
```
((USER)
        (set! out_event "USERD")
  )
```

Step 6: Update eX_from_ePM.tsc

Update eX_from_ePM.tsc to perform an iq-put with the new Event Type, if the new output event is received. This should be added within the case statement that checks for the comm protocol.

Figure 93 shows an example script.

Figure 93 Example eX_from_ePM.tsc



12.2.2 Adding a Customer Protocol for RosettaNet 1.1

The steps required to create an additional protocol are as follows:

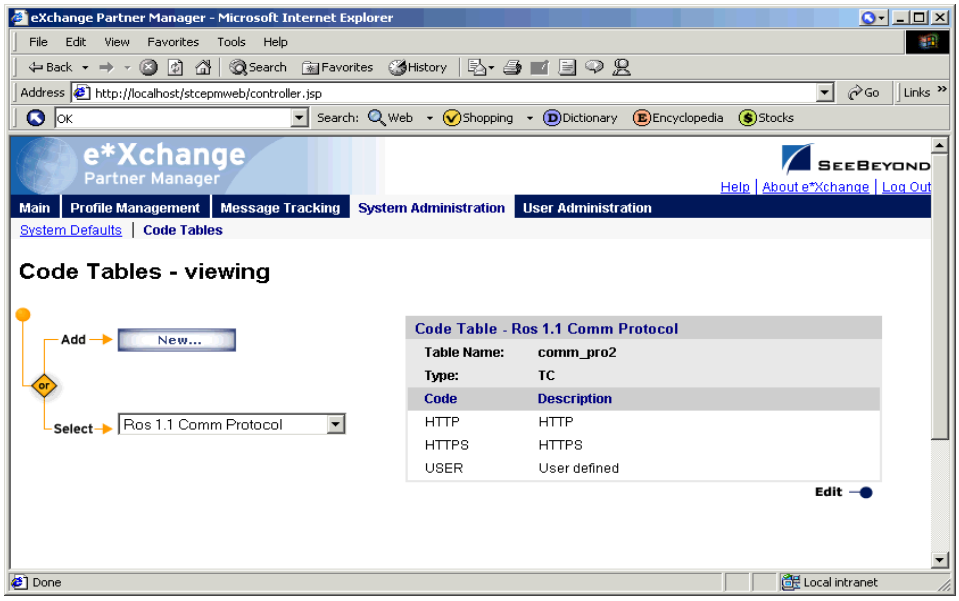
1. Add a Comm Protocol to the Code Table.
2. Add an Event Type for the protocol.
3. Update eX_from_ePM Collaboration Rule to publish the new Event Type.
4. Update eX_from_ePM Collaboration to publish the new Event Type to the eX_Trading_Port_Queue IQ.
5. Edit monk_scripts\common\ROS\eX_ROS_main.dsc to set the output event.
6. Update eX_from_ePM.tsc to perform an iq-put with the new Event Type, if the new output event is received.
7. (Optional, for use with Ack Monitor) Update ack_mon.dsc to support resends for RosettaNet.

Step 1: Add a Comm Protocol to the Code Table

The ROS 1.1 Comm Protocol in the Code table lists the protocols that are currently available. Add an addition protocol and assign a name and description.

Figure 94 shows a protocol named USER.

Figure 94 Example Code Table for RosettaNet 1.1



Step 2: Add an Event Type for the Protocol

Use the e*Gate Enterprise Manager GUI to create a new Event Type in eXSchema. For example, **eX_User**.

Step 3: Update eX_from_ePM Collaboration Rule

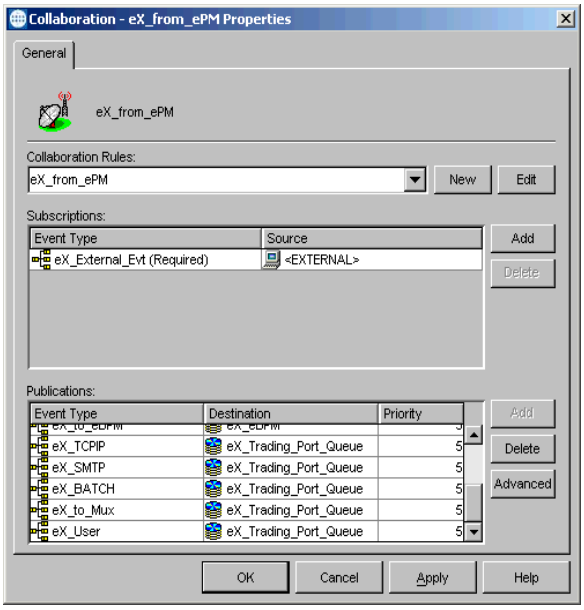
Update the eX_from_ePM Collaboration Rule to publish the Event Type created in Step 2.

Step 4: Update eX_from_ePM Collaboration

Update eX_from_ePM Collaboration to publish the new Event Type to the eX_Trading_Port_Queue IQ.

Figure 95 shows an example configuration using eX_User.

Figure 95 Example eX_from_ePM using eX_User



Step 5: Update eX_ROS_main.dsc

Modify **monk_scripts/common/ROS/eX_ROS_main.dsc**. Search for lines that specify "HTTPS". Replace all incidences of HTTPS with g_commport, so the new communication protocol just added is included in the outgoing message, and does not default to HTTPS.

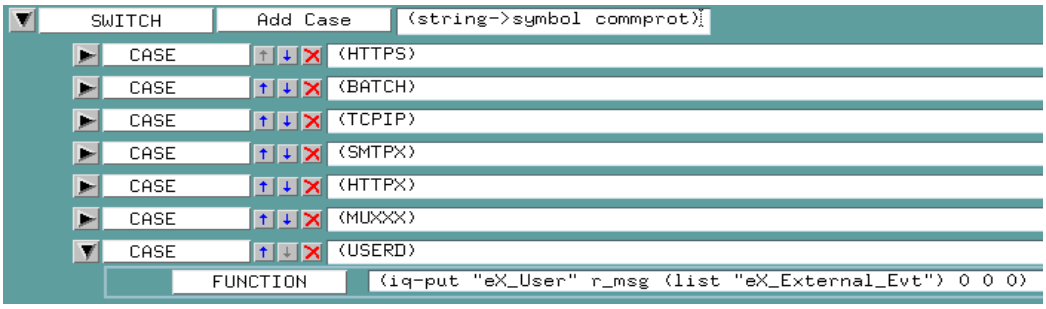
It is necessary to ensure that g_commport has exactly 5 characters. If the new protocol name is not exactly five characters then reset g_commport to a five character string in this script. For example, reset g_commport from “USER” to “USERD”.

Step 6: Update eX_from_ePM.tsc

Update eX_from_ePM.tsc to perform an iq-put with the new Event Type, if the new output event is received. This should be added within the case statement that checks for the comm protocol.

Figure 96 shows an example script.

Figure 96 Example eX_from_ePM.tsc



Step 7: Modify ack_mon.dsc

Modify the event-send-to-egate function in ack_mon.dsc to support RosettaNet resends from Ack Monitor. You need to add support for the new comm protocol since the default is HTTPS in the file.

Find the following section in the code:

```
(event-send-to-egate (string-append "R|O|HTTPS" (get ~output%eX_Event)))
```

Replace HTTPS with g_commport, so the new communication protocol just added is included in the outgoing message, and does not default to HTTPS.

It is necessary to ensure that g_commport has exactly 5 characters. If the protocol name is not exactly five characters then reset g_commport to a five character string in this script. For example, reset g_commport from "USER" to "USERD", or from "HTTP" to "HTTPS".

12.2.3 Adding a Customer Protocol for RosettaNet 2.0

The steps required to create an additional protocol are as follows:

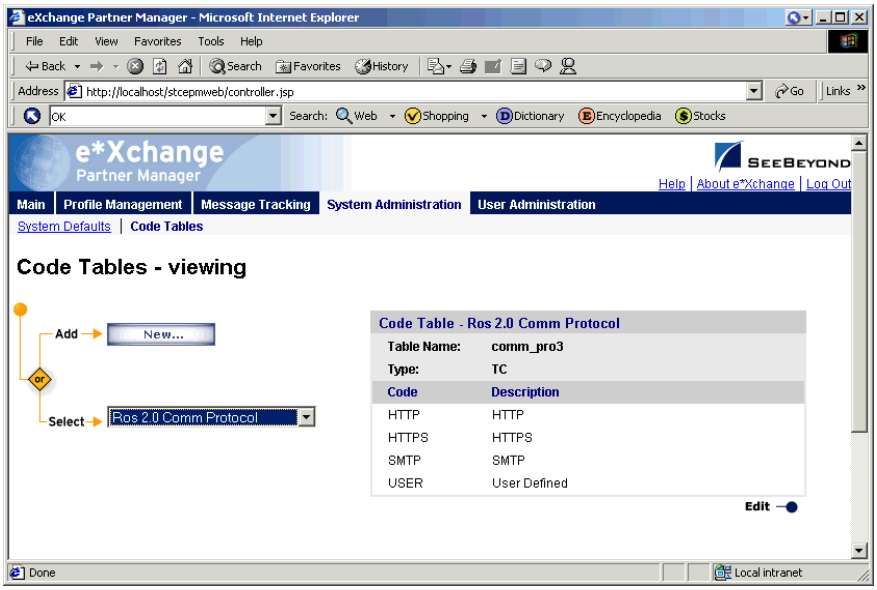
1. Add a Comm Protocol to the Code Table.
2. Add an Event Type for the protocol.
3. Update eX_from_ePM Collaboration Rule to publish the new Event Type.
4. Update eX_from_ePM Collaboration to publish the new Event Type to the eX_Trading_Port_Queue IQ.
5. Edit monk_scripts/common/ROS/eX-ROS20-Send-To-Egate.monk to set the output event.
6. Update eX_from_ePM.tsc to perform an iq-put with the new Event Type, if the new output event is received.

Step 1: Add a Comm Protocol to the Code Table

The Comm Protocol in the Code table lists the protocols that are currently available. Add an addition protocol and assign a name and description.

Figure 97 shows a protocol named USER.

Figure 97 Example Code Table for RosettaNet 2.0



Step 2: Add an Event Type for the Protocol

Use the e*Gate Enterprise Manager GUI to create a new Event Type in eXSchema. For example, **eX_User**.

Step 3: Update eX_from_ePM Collaboration Rule

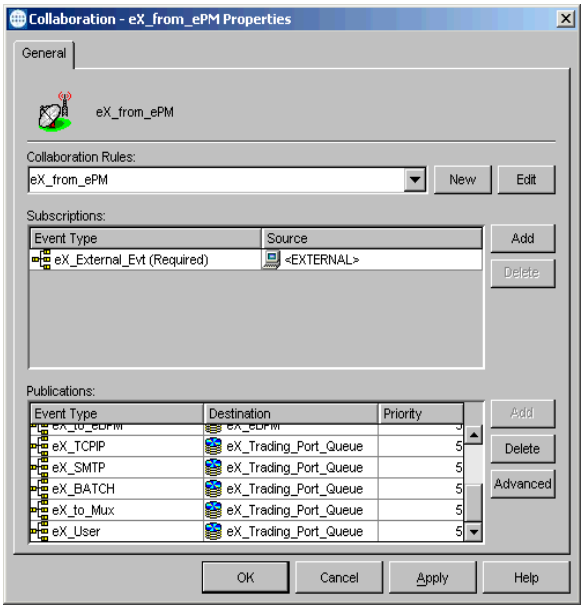
Update the eX_from_ePM Collaboration Rule to publish the Event Type created in Step 2.

Step 4: Update eX_from_ePM Collaboration

Update eX_from_ePM Collaboration to publish the new Event Type to the eX_Trading_Port_Queue IQ.

Figure 98 shows an example configuration using eX_User.

Figure 98 Example eX_from_ePM using eX_User



Step 5: Update eX_ROS_Send_To_Egate.monk

Edit monk_scripts/common/ROS/eX_ROS20_Send_To_Egate.monk to set the output event. This step also enables Ack Monitor to handle the new protocol. Find eX-ROS20-Forward-To-TP function and add the following if statement:

```
(if (string-ci=? comm_port "<Comm Protocol>")
    (begin
      (set! comm_port "<Comm Protocol Ref>")
    )
    (begin
    )
  )
```

where

- <Comm Protocol> defines the name given in the code table
- <Comm Protocol Ref> is a used defined name with exactly five characters

Example code for the USER protocol:

```
(if (string-ci=? comm_port "USER")
    (begin
      (set! comm_port "USERD")
    )
    (begin
    )
  )
```

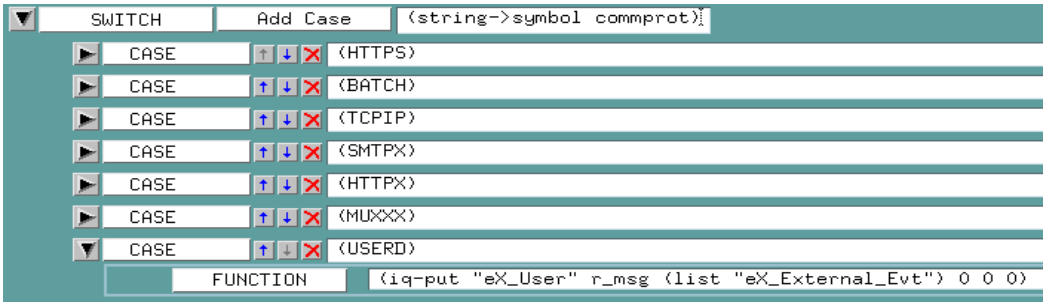
In addition to setting the comm_port, rules for copying and setting values in the outgoing message should be added within the if statement for this new protocol.

Step 6: Update eX_from_ePM.tsc

Update eX_from_ePM.tsc to perform an iq-put with the new Event Type, if the new output event is received. This should be added within the case statement that checks for the comm protocol.

Figure 99 shows an example script.

Figure 99 Example eX_from_ePM.tsc



Chapter 13

e*Xchange Partner Manager Functions

This chapter provides information on the e*Xchange APIs. These APIs are divided into three groups based on their use within e*Xchange. These groups are:

- e*Xchange helper functions (used when working with the e*Xchange ETD) see **"e*Xchange Helper Monk Functions" on page 217**
- e*Xchange Partner Manager functions (used by the e*Xchange) see **"e*Xchange Functions" on page 224**
- Validation Rules Builder functions (used by the validation Collaborations created by the VRB) see **"Monk Functions Used by the Validation Rules Builder" on page 322**
- Mime functions, see **"e*Xchange MIME Functions" on page 333**
- RosettaNet 2.0 functions, see **"e*Xchange RosettaNet 2.0 Functions" on page 341**
- Security functions, see **"e*Xchange Security Functions" on page 380**

13.1 e*Xchange Helper Monk Functions

A number of Monk functions have been added to make it easier to set information in the e*Xchange Event (eX_Standard_Event.ssc ETD) and to get information from it. These functions are contained in two files:

- **eX-ePM-utils.monk**

Important: *Make sure that the Monk file **eX-ePM-utils.monk**, containing the e*Xchange helper functions, are loaded before calling them in a Collaboration Rules Script. You can do this in several ways, by putting them in the root of the **monk_library** directory, loading them explicitly in your CRS, or using the **eX-init-eXchange** bootstrap file to load them via the Collaboration Rule. See* **"Convert the Event to Base 64 Encoding" on page 73** *for an example of how to use the **eX-init-eXchange** bootstrap file in a Collaboration Rule.*

These functions are discribed in detail on the following pages:

eX-set-TP_EVENT

Syntax

```
(eX-set-TP_EVENT root-path event-type value)
```

Description

eX-set-TP_EVENT sets the value of the specified event type.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	Either PARTNERNAME, MESSAGEID, ORIGEVENTCLASS, USAGEINDICATOR, COMMPROT, URL, INTERNALNAME, or DIRECTION
value	string	The value to which you want to set the event type. For event-type "DIRECTION" value must be I or O. For event-type "USAGEINDICATOR" value must be P or T.

Return Values

Boolean

Returns **#t** (true) except when an invalid parameter is passed, then **#f** (false) is returned.

Throws

None.

Example

```
(eX-set-TP_EVENT ~input%eX_Event "PARTNERNAME" "Acme Inc.")
=> sets the trading partner name to "Acme Inc."
```

eX-get-TP_EVENT

Syntax

```
(eX-get-TP_EVENT root-path event-type)
```

Description

eX-get-TP_EVENT finds the path to the value of the specified event type in the e*Xchange Event named in the root-path.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	Either PARTNERNAME, MESSAGEID, ORIGEVENTCLASS, USAGEINDICATOR, COMMPROT, URL, INTERNALNAME, DIRECTION, or PAYLOAD

Return Values

Returns one of the following values:

Boolean

Returns **#f** (false) if the value for TP_EVENT is not found.

path

Returns the path to the value found at the TP_EVENT node location. Use **get** to return the actual value.

Throws

None.

Example

For an Event with a partner name of "Acme Inc.":

```
(get (eX-get-TP_EVENT ~input%eX_Event "PARTNERNAME"))
=> Acme Inc.
```

eX-set-Payload

Syntax

```
(eX-set-Payload root-path encrypt-type value)
```

Description

eX-set-Payload sets the value of the payload.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
encrypt-type	string	RAW, PROCESSED, ENCRYPTED.
value	string	The value to which you want to set the payload.

Return Values

Returns **#t** (true) except when an invalid parameter is passed, then **#f** (false) is returned.

Throws

None.

Example

```
(eX-set-Payload ~input%eX_Event "RAW" "Have a nice day!")
=> sets the payload to "Have a nice day!"
```

eX-count-TP-attribute

Syntax

```
(eX-count-TP-attribute root-node)
```

Description

eX-count-TP-attribute searches the e*Xchange Event for the number of trading partner attributes using the name/value pair format stored in the repeating **TPAttribute** node in the TP_EVENT portion of the e*Xchange Event.

Parameters

Name	Type	Description
root-node	path	Either ~input%eX_Event or ~output%eX_Event

Return Values

Returns the following:

integer
Number of TPAttribute name/value pairs.

Throws

None.

Example

For a e*Xchange Event that has 3 TPAttributes:

```
(eX-count-TP-attribute ~input%eX_Event)
=> 3
```

eX-get-TP-attribute

Syntax

```
(eX-get-TP-attribute root-node name)
```

Description

eX-get-TP-attribute finds the attribute value in the e*Xchange Event named in the root-node.

Parameters

Name	Type	Description
root-node	path	Either ~input%eX_Event or ~output%eX_Event
name	string	Name of the trading partner attribute you want to get.

Return Values

Returns the following:

string
Returns the value associated with the TPAttribute name.

Throws

None.

Example

```
(eX-get-TP-attribute ~input%eX_Event "COMM_PROT")
=> "X12"
```

eX-set-TP-attribute

Syntax

```
(eX-set-TP-attribute root-node name value)
```

Description

eX-set-TP-attribute creates an entry in the e*Xchange Event under the TPAttribute repeating node for the specified name/value pair.

Parameters

Name	Type	Description
root-node	path	Either ~input%eX_Event or ~output%eX_Event
name	string	The name of the TP attribute whose value you want to set.
attribute	string	The value to which you want to set the TP attribute.

Return Values

None.

Throws

None.

Example

```
(eX-set-TP-attribute ~output%eX_Event "COMM_PROT" "X12")

=> creates an entry in the e*Xchange Event under TPAttribute for the
name/value pair COMM_PROT/X12.
```

13.2 e*Xchange Functions

The specialized e*Xchange Monk functions used by e*Xchange are:

ux-ack-handler

Syntax

```
(ux-ack-handler connection-handle ack-stat)
```

Description

ux-ack-handler performs message association for an inbound or outbound business message.

Note: *If the acknowledgment is to be stored in the database, then* ***ux-store-msg*** *should be called before* ***ux-ack-handler*** *to store the ack.*

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
ack-stat	list: ack_tm ack_type level mtrk_id unique_id error_data direction out_queue resp_id sub-list	Required. Information about the acknowledgment. All list arguments must be strings, except for the sub-lists which are lists containing strings. All elements before the sub-list section are required, but can be empty strings (""), with the exception of ack_type, level, unique_id and direction, which return an error if no value is provided. The first sub-list is required.

Name	Type	Description
ack-stat	**List member**	**Description**
	ack_tm	The date and time (yyyymmddhhmmss format).
	ack_type	Identifies the kind of acknowledgment (positive or negative): P—Positive acknowledgment N—Negative acknowledgment
	level	Required. Specifies the level of the original message: I—B2B Protocol level information T—Message Profile level information
	mtrk_id	Optional—future versions may use this value to store messages.
	unique_id	The unique identifier for the original message.
	error_data	Error information— code^description~code^description (^ separates the values for an error and ~ separates the errors).
	direction	Required. Indicates the direction of the message: I—Inbound O—Outbound
	out_queue	Indicates whether the message is placed in es_out_queue: Y—Yes N—No
	resp_id	Optional—tpts_id for Message Profile or tpic_id for B2B Protocol This needed when the message received by e*Xchange is not known to be an original message or response.

Return Values

Returns one of the following values:

string

Returns **mtrk_id**, if found and the row is updated.

Boolean

Returns **#t** (true) if the acknowledgment processed successfully; otherwise, returns **#f** (false). Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

Exception-InvalidArg, Exception-Mapping, Exception-Catastrophic (can't be caught).

Additional Information

ux-ack-handler updates es_mtrk_outb or es_mtrk_inb, es_waiting_ack, and es_mtrk_error database tables based on the global transact info associated with the acknowledgment just received.

ux-ack-handler does the following:

- Receives a database connection handle and a list of information about the acknowledgment from the calling process.
- Updates es_mtrk_inb.ack_que_tm and es_mtrk_inb.ack_msg_id if direction is "I". For direction = "O", updates es_mtrk_outb.ack_tm and es_mtrk_outb.ack_msg_id , and deletes a row from es_waiting_ack table corresponding to the original mtrk_outb_id.
- If the acknowledgment tran_set_id does not match the expected tran_set_id and the ack_type = "P", then the acknowledgment is ignored.
- Returns a value to the calling process that indicates whether or not it was successful.

Based on values of mtrk_id and resp_id in ack-stat, **ux-ack-handler** performs the following:

- If mtrk_id is provided, but resp_id is not provided, then **ux-ack-handler** uses mtrk_id to update the correct mtrk row in the database
- If resp_id is provided, but mtrk_id is not provided, then **ux-ack-handler** tries to find the corresponding request_ids (es_ids) by looking to see if the resp_id is part of RTN_TS_ID values. Then it uses any found request_ids (es_ids), unique_id, and extended list after ID part to find mtrk_id. If no corresponding request_ids (es_ids) are found or no mtrk_id is found, then the resp_id is treated as a request_id (es_id).
- If mtrk_id is provided and resp_id is provided, then **ux-ack-handler** ignores resp_id and uses only mtrk_id to update the correct row in the mtrk table.
- If mtrk_id is not provided and resp_id is not provided, then **ux-ack-handler** uses the given criteria (unique_id, global structures, and extended data) to find the correct mtrk row to update.

Example

The following Monk script example calls **ux-ack-handler**. This script makes two assumptions:

- That **ux-init-trans** was executed successfully for the given acknowledgment.
- That a connection to the database, **conn-handle**, has been established before **ux-ack-handler** is called.

ux-ack-handler uses the unique_id "TESTVAL129" to find the appropriate row to update in es_mtrk_outb.

If successful, then ack-msg is placed in es_mtrk_outb.ack_msg_id in the same row as the original message. Ack-code "Negative" and ack-tm set as the current time are also stored in es_mtrk_outb.

If **ux-ack-handler** fails, then the error, **ux-get-error-str**, is displayed.

```
(define ack-stat (list "" ; ack_tm
          "N" ; ack_type
          "T"; level
          ""  ; mtrk_id
          "TESTVAL129" ; unique_id
          "12345^Bad dept code~56789^Invalid bed" ; error_data
          "O" ; direction
          "N" ; out_queue
          ""  ; resp_id
          ""
         ))
(if (not (ux-ack-handler connection-handle ack-stat))
(begin
    (display "Ack Handler failed!\n")
    (display (ux-get-error-str))
    (newline)
)
    (display "Ack Handler succeeded!\n")
)
```

ux-ack-monitor

Syntax

```
(ux-ack-monitor connection-handle wa-id)
```

Description

ux-ack-monitor processes messages that have overdue acknowledgments.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
wa-id	string	Required. The ID of an es_waiting_ack record.

ux-ack-monitor processes messages as described below:

Return Values

Returns one of the following values:

Boolean

Returns #t (true) if the API is successful and no records exceeded the retry max; otherwise, returns #f (false) if an error occurs and the API is not successful. Use ux-get-error-str to retrieve the corresponding error message.

Vector

Returns a vector of mtrk_outb_ids and associated original msgs for all mtrk_outb_ids associated with the waiting_ack_id that achieved the retry max.

This vector should not contain duplicate msgs. Therefore it is possible that one mtrk_outb_id represents all the mtrk_outb_ids that have the same orig_msg_id.

This vector takes the following form: (mtrk_outb_id1 msg1 mtrk_outb_id2 msg2 ...)

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Additional Information

ux-ack-monitor receives, as a parameter, the record ID of a record in the es_waiting_ack table that has an expired acknowledgment time. This occurs when e*Xchange Partner Manager did not receive an acknowledgment message within the time allotted by the trading partner profile. The API determines the transaction type (X12 or RosettaNet), the transfer mode (Interactive or Batch), and the message send count as compared to the maximum resend count allowed.

For interactive messages:

If the passed in waiting_ack_id has not hit the maximum allowable retries then the following occurs:

1. The next send time is updated for this waiting_ack_id and all es_waiting_ack rows that refer to the same original message (es_mtrk_outb.orig_msg_id).

2 **ux-ack-monitor** increments send count for this row all rows with same es_mtrk_outb.orig_msg_id.

3 The send count is updated in the es_waiting_ack and es_mtrk_outb tables, and an entry is inserted into the es_out_queue so that the message is resent to the trading partner by the e*Xchange Transaction Polling e*Way.

If the waiting_ack_id that was passed in has exceeded the maximum allowable retries then:

1 **ux-ack-monitor** stores an error for each es_mtrk_outb row that has the same es_mtrk_outb.env_msg_id as the waiting_ack_id, and the es_mtrk_outb.ack_msg_id is NULL (have not received an ack).

2 All rows in es_waiting_ack that have the same es_mtrk_outb.orig_msg_id as the passed in waiting_ack_id are deleted.

For batch messages:

If the passed in waiting_ack_id has not hit the maximum allowable retries then:

1 X12 (TS level) - Removes all env_msg_id rows associated with passed in waiting_ack_id and nulls out control numbers, so batch process resends.

X12 (IC level) - All TS records that are subsets of IC es_mtrk_outb record referred to by waiting_ack_id passed in, and those records that have rtn_rcpt set to 'N' have all similar env_msg_id rows removed, IC_CONTROL_NUM is set to the value held in IC_BATCH, and FGI_CONTROL_NUM is set to the value held in FGI_BATCH. This allows the batch process to perform a resend. **ux-ack-monitor** stores an error for the IC level es_mtrk_outb record that it has timed out.

EDF - Removes all env_msg_id rows associated with passed in waiting_ack_id, IC_CONTROL_REF is set to the value held in IC_BATCH, and FGI_CONTROL_REF is set to the value held in FGI_BATCH, so batch process resends.

2 all rows in es_waiting_ack that have the same es_mtrk_outb.orig_msg_id as the passed in waiting_ack_id are deleted.

If the passed in waiting_ack_id has exceeded the maximum allowable retries then:

1 **ux-ack-monitor** stores an error for each es_mtrk_outb row that has the same es_mtrk_outb.env_msg_id as the waiting_ack_id, and the es_mtrk_outb.ack_msg_id is NULL (have not received an ack).

2 all rows in es_waiting_ack that have the same es_mtrk_outb.orig_msg_id as the passed in waiting_ack_id are deleted.

Example

The following example passes a record ID of 75 from the es_waiting_ack table. The function assumes that the record has already been identified as having timed out. If a vector is returned then information about each mtrk_outb_id is written to the log and the database changes made are committed. If the function returns **#t** (true), the database changes made are committed. If **#f** (false) is returned (indicating an error) a rollback is committed to roll back any database changes that may have occurred before the error was encountered.

```
(define mtrk_outb_id_msgs (ux-ack-monitor connection-handle "75"))
(cond ((not (boolean? mtrk_outb_id_msgs))
 (do ((i 0 (+ i 1)) (value-count (vector-length mtrk_outb_id_msgs)))
     ((= i value-count))
            (display "mtrk_outb_id <")
            (display i)
            (display "> = ")
            (display (vector-ref mtrk_outb_id_msgs i))
            (newline)
            (set! i (+ i 1))
            (display "msg = ")
            (display (vector-ref mtrk_outb_id_msgs i))
            (newline)
  )
  (db-commit connection-handle)
 )
 (else
       (begin
        (if (eq? #t mtrk_outb_id_msgs)
          (begin
           (display "ux-ack-monitor succeeded - no mtrk_outb_ids hit
retry max")
           (db-commit connection-handle)
          )
          (begin
           (display (ux-get-error-str))
           (db-rollback connection-handle)
          )
        )
      )
 )
)
```

ux-check-shutdown-uid

Syntax

```
(ux-check-shutdown-uid connection-handle id level unique_id)
```

Description

ux-check-shutdown-uid compares unique_id provided with ones in es_sd_data. If there is a match, then it returns a full unique_id and deletes the row from es_sd_data table.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
id	string	Required. Either tpic_id or tpts_id.
level	string	Required. The level of the control number. Valid values: I—Indicates ID is tpic_id T—Indicates ID is tpts_id
unique_id	string	Required. The string that uniquely identifies the transaction.

Return Values

Returns one of the following values:

string

Returns a string containing the unique_id from es_sd_data table if the combination of tpts_id, or tpic_id, level, and unique_id is found.

Boolean

Returns **#t** (true) if the combination of tpts_id, or tpic_id, level, and unique_id is not found in the es_sd_data table.

Returns **#f** (false) if a problem occurs.

Throws

None.

Example

```
(define unique_id "AAAAA")
(define orig_tpts_id "1")
(define check-result (ux-check-shutdown-uid connection-handle
                          orig_tpts_id "T" unique_id)
)
```

ux-control-check

Syntax

```
(ux-control-check control-num level type)
```

Description

ux-control-check determines whether the control number provided in an inbound Message Profile or B2B Protocol is valid.

It does the following:

- Checks the es_ext_detail and es_ext_data tables for the control numbers.
- Determines whether the control number in the message is valid; that is, whether message control num is greater than database control num.

Parameters

Name	Type	Description
control-num	string	Required. The control number to validate.
level	string	Required. The level of the control number. Valid values: I—Interchange control number G—Functional group control number T—Transaction set control number
type	string	O—Original A—Ack

Return Values

Returns one of the following values:

string

Returns "Y" if the control number is valid; otherwise returns "N" if the control number is not valid.

Boolean

Returns **#f** (false) if the API fails. Use **ux-get-error-str** to retrieve the corresponding error message.

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Additional Information

ux-control-check compares the control numbers that are found to the values in the global structures, which represent the values in the database. The values should be as follows:

Level	Global Structure Control Number
T	g_ts->ext_data.col_value and g_ts->ext_data.col_name = "T_CONTROL_NUM"

Level	Global Structure Control Number
G	g_ic->ext_data.col_value and g_ic->ext_data.col_name = "G_CONTROL_NUM"
I	g_ic->ext_data.col_value and g_ic->ext_data.col_name = "I_CONTROL_NUM"

The message's control number is valid if either of the following two conditions is true:

- The message's control number is greater than the database control number, or
- The message's control number is 1 or 0 and the database control number is the maximum (9999999999).

If the database control number is at the maximum, the next time it is incremented it starts over.

The control number must contain only numeric characters, and it must be greater than the number stored in the database/global structures. The control number can have leading zeros.

Example

The following Monk script example calls **ux-control-check** with the assumption that **ux-init-trans** was executed successfully for the given message. **ux-control-check** compares the given control-num "1005" and level "G" with the g_control_num stored in the database, g_ic->ext_data.col_value where g_ic->ext_data.col_name = "G_CONTROL_NUM". If "1005" is greater than g_ic->ext_data.col_value where g_ic->ext_data.col_name = "G_CONTROL_NUM", then con-res = "Y", otherwise con-res = "N". If an error occurs, then **#f** is returned and the error string is printed using the display of **ux-get-error-str**.

```
(define control-num "1005")
(define level "G")
(define type "O")

 (define con-res (ux-control-check control-num level type))
   (cond ((not (boolean? con-res))
      (cond ((string-ci=? "Y" con-res)
               (display "Control Number is valid\n")
             )
             (else
               (display "Control Number is NOT valid\n")
             )
           )
    )
    (else
        (display (ux-get-error-str))
        (newline)
    )
 )
```

ux-dbproc-ros-inb

Syntax

```
(ux-dbproc-ros-inb connection-handle msg list_of_rn_pars)
```

Description

ux-dbproc-ros-inb handles message tracking and acknowledgment for an inbound RosettaNet message that e*Xchange receives from a trading partner.

Parameters

Name	Type	Description
connection-handle	connection-handle Required.	The previously established connection to the database.
msg	String	The raw message received from the trading partner.
List_of_rn_pars	list: global_proc_ind_code global_proc_id global_trans_code global_trans_id global_sigact_code global_sigact_id inrespto_sigact_code inrespto_sigact_id rec_ack_time acc_ack_time perform_time ext_data_col_name ext_data_col_value …	Required. RosettaNet transaction information. All list arguments must be strings. Any number of ext_data_col_name---ext_data_col_value pairs can be specified as long as they are specified in pairs. All elements are required, but can be empty strings ("").
	List member	**Description**
	global_proc_ind_code	The RosettaNet global process indicator code.
	global_proc_id	The RosettaNet global process ID.
	global_trans_code	The RosettaNet global transaction code.
	global_trans_id	The RosettaNet global transaction ID.
	global_sigact_code	The RosettaNet global action code or signal code. This depends on whether the message is a RosettaNet business action or business signal.
	global_sigact_id	The RosettaNet global action code or signal ID. This depends on whether the message is a RosettaNet business action or business signal.

Name	Type	Description
	inrespto_sigact_code	If the message is a RosettaNet business signal or an action that is a response to another action, this field is the global action code for the original action.
	inrespto_sigact_id	If the message is a RosettaNet business signal or an action that is a response to another action, this field is the global action ID for the original action.
	rec_ack_time	Time allowed to acknowledge the receipt of the message.
	acc_ack_time	Time allowed to acknowledge the acceptance of the message.
	perform_time	Time to carry out the action specified in the message and provide a response.
	ext_data_col_name	Optional. Field name for any external data to be saved with the message.
	ext_data_col_value	Optional, but must appear in pair with ext_data_col_value. This value is assigned to the external data field with the corresponding ext_data_col_name as the column name. Any external data, if specified, are saved with the message.
	…	More ext_data_col_name --- ext_data_col_value pairs.

Return Values

Returns one of the following values:

string

Returns a string indicating what action to take, when the function executes successfully.

Return String	Action to Take
SEND_BPFAILURE_TO_EGATE	Send a standard event to the eX_eBPM queue indicating failure of the process.
SEND_REC_ACK_TO_TP	Send a receipt acknowledgment to the eX_Trading_Port_Queue.
SEND_ACC_ACK_TO_TP	Send an acceptance acknowledgment to the eX_Trading_Port_Queue.
SEND_MSG_TO_EGATE	Send the original message to the eX_eBPM queue.
SEND_MSG_TO_TP	Send the original message to the eX_Trading_Port_Queue.

Boolean

Returns **#f** (false) when the function fails to complete successfully.

Note: *Before sending an acknowledgment to the eX_Trading_Port_Queue, it is the caller's responsibility to save the acknowledgment message using ux_store_msg and to register it as a response message to the original message using ux_ack_handler.*

Throws

Exception-InvalidArg, Exception-Mapping, Exception-Catastrophic (can't be caught).

Additional Information

ux-dbproc-ros-inb does the following:

- Receives a database connection handle and a list of information about the RosettaNet message from the calling process.
- Store the original message with any external data using **ux-store-msg**.
- If the message is a General Exception in response to an original message, handles all expected acknowledgments and responses for the original message as if a negative ack was received for each. Adds the command "SEND_BPFAILURE_TO_EGATE" to the returned list.
- If the message is a Receipt acknowledgment, handles the expected receipt acknowledgment using ux_ack_handler. If it is a Receipt Acknowledge Exception, adds the command "SEND_BPFAILURE_TO_EGATE" to the returned list.
- If the message is an Acceptance acknowledgment, handles any expected Receipt acknowledgment as if positive ack were received, then handles the expected Acceptance acknowledgment. If it is a Acceptance Acknowledge Exception, adds the command "SEND_BPFAILURE_TO_EGATE" to the returned list.
- If the message is a business action in response to an original action, handle the expected response for the original action message using **ux-ack-handler**.
- For any business action message, sets up message tracking for each response expected by the message.
- If a Receipt acknowledgment is expected by this message, handles the acknowledgment as if a positive ack was received from e*Gate and adds the command "SEND_REC_ACK_TO_TP" to the command list to be returned.
- If a Acceptance acknowledgment is expected by this message, handles the acknowledgment as if a positive ack was received from e*Gate and adds the command "SEND_ACC_ACK_TO_TP" to the command list to be returned.
- For a business action message, adds the command "SEND_MSG_TO_EGATE" to the command list to be returned.
- Commits or rolls back the database depending on the result of the process. Returns to the caller.

Examples

The following Monk script example calls **ux-dbproc-ros-inb**. This script makes three assumptions:

- That **ux-init-trans** was executed successfully for the given message.
- That a connection to the database, **conn-handle**, has been established before **ux-dbproc-ros-inb** is called.
- All variables in the first two statements below have been properly defined with values either from the message itself or from the trading partner profile in the database.

If **ux-dbproc-ros-inb** fails, then a user defined function **SendFailureNotification** is called.

```
(set! dbproc_info
    (list g_ros_proc_ind_code g_ros_proc_id g_ros_trans_code
          g_ros_trans_id g_ros_sigact_code g_ros_sigact_id
          g_ros_inrespto_code g_ros_inrespto_id g_ros_rec_ack_time
          g_ros_acc_ack_time g_ros_perform_time error_data))
(set! dbproc_info
    (append dbproc_info
        (list "ACTIVITY_TYPE" activity_type "ACT_INST_ID" event_id)))
(set! dbproc_info
    (ux-dbproc-ros-inb g_connection_handle
        (get ~input%RosettaNetGeneric) dbproc_info))
(if (boolean? dbproc_info)
    (begin
        (SendFailureNotification g_direction)
        (throw Exception-Monk-Usage
            (string-append "ux-dbproc-ros-inb() failed: <"
                           (ux-get-error-str)
                           ">\n")))
    (begin)
)
(do ((i 0 (+ 1 i)))
    ((>= i (vector-length dbproc_info)))
    (let ((element (vector-ref dbproc_info i)))
    (if (string=? element "SEND_BPFAILURE_TO_EGATE")
        (begin
         …
        )
        (begin)
    )
    (if (string=? element "SEND_MSG_TO_EGATE")
        (begin
         …
        )
        (begin)
    )
… ;Take action for other commands.
)
```

ux-dbproc-ros-outb

Syntax

```
(ux-dbproc-ros-outb connection-handle msg list_of_rn_pars)
```

Description

ux-dbproc-ros-outb handles message tracking and acknowledgment for an outbound RosettaNet message that the e*Xchange receives from e*Gate.

Parameters

Name	Type	Description
connection-handle	connection-handle Required.	The previously established connection to the database.
msg	String	The raw message received from e*Gate
List_of_rn_pars	list: global_proc_ind_code global_proc_id global_trans_code global_trans_id global_sigact_code global_sigact_id inrespto_sigact_code inrespto_sigact_id rec_ack_time acc_ack_time perform_time ext_data_col_name ext_data_col_value …	Required. RosettaNet transaction information. All list arguments must be strings. Any number of ext_data_col_name---ext_data_col_value pairs can be specified as long as they are specified in pairs. All elements are required, but can be empty strings ("").
	List member	**Description**
	global_proc_ind_code	The RosettaNet global process indicator code.
	global_proc_id	The RosettaNet global process ID.
	global_trans_code	The RosettaNet global transaction code.
	global_trans_id	The RosettaNet global transaction ID.
	global_sigact_code	The RosettaNet global action code or signal code, depend on if the message is a RosettaNet business action or business signal.
	global_sigact_id	The RosettaNet global action code or signal ID, depend on if the message is a RosettaNet business action or business signal.

Name	Type	Description
	inrespto_sigact_code	If the message is a RosettaNet business signal or an action that is a response to another action, this field is the global action code for the original action.
	inrespto_sigact_id	If the message is a RosettaNet business signal or an action that is a response to another action, this field is the global action ID for the original action.
	rec_ack_time	Time to acknowledge the receipt of the message.
	acc_ack_time	Time to acknowledge the acceptance of the message.
	perform_time	Time to carry out the action specified in the message and provide a response.
	ext_data_col_name	Optional. Field name for any external data to be saved with the message.
	ext_data_col_value	Optional, but must appear in pair with ext_data_col_value. This value is assigned to the external data field with the corresponding ext_data_col_name as the column name. Any external data, if specified, are saved with the message.
	…	More ext_data_col_name --- ext_data_col_value pairs.

Return Values

Returns one of the following values:

string

Returns a string indicating what action to take when the function executes successfully.

Return String	Action to Take
SEND_MSG_TO_TP	end the original message to the eX_Trading_Port_Queue.

Boolean

Returns **#f** (false) when the function fails to complete successfully.

Throws

Exception-InvalidArg, Exception-Mapping, Exception-Catastrophic (can't be caught).

Additional Information

ux-dbproc-ros-outb does the following:

- Receives a database connection handle and a list of information about the RosettaNet message from the calling process.

- Stores the original message with any external data using **ux-store-msg**.
- If the message is a General Exception, handles all expected acknowledgments and responses for the original message as if a negative ack were received for each.
- If the message is a business action in response to an original action, handles the expected response for the original action message using **ux-ack-handler**.
- For any business action message, sets up message tracking for each response expected by this message.
- Adds the command "SEND_MSG_TO_TP" to the command list to be returned.
- Commits or rolls back the database depending on the result of the process, then returns to the caller.

Examples

The following Monk script example calls **ux-dbproc-ros-outb**. This script makes three assumptions:

- That **ux-init-trans** was executed successfully for the given message.
- That a connection to the database, conn-handle, has been established before **ux-dbproc-ros-outb** is called.
- All variables in the first two statements below have been properly defined with values either from the message itself or from the partner profile in the database.

If **ux-dbproc-ros-outb** fails, then the error, a user defined function **SendFailureNotification** is called.

```
(set! dbproc_info (list g_ros_proc_ind_code g_ros_proc_id
                        g_ros_trans_code g_ros_trans_id
                        g_ros_sigact_code g_ros_sigact_id
                        g_ros_inrespto_code g_ros_inrespto_id
                        g_ros_rec_ack_time g_ros_acc_ack_time
                        g_ros_perform_time error_data))
(set! dbproc_info
    (append dbproc_info
        (list "ACTIVITY_TYPE" activity_type "ACT_INST_ID" event_id)))
(set! dbproc_info
    (ux-dbproc-ros-outb g_connection_handle
                        (get ~input%RosettaNetGeneric)
                        dbproc_info))
(if (boolean? dbproc_info)
    (begin
        (SendFailureNotification g_direction)
        (throw Exception-Monk-Usage
            (string-append "ux-dbproc-ros-outb() failed: <"
                           (ux-get-error-str) ">\n")))
    (begin)
)
(do ((i 0 (+ 1 i))) ((>= i (vector-length dbproc_info)))
(let ((element (vector-ref dbproc_info i)))
    (if (string=? element "SEND_BPFAILURE_TO_EGATE")
        (begin
        …
        )
        (begin)
    )
    (if (string=? element "SEND_MSG_TO_EGATE")
        (begin
        …
        )
        (begin)
    )
… ;Take action for other commands.
)
```

ux-dequeue

Syntax

```
(ux-dequeue connection-handle)
```

Description

ux-dequeue retrieves enveloped, outbound messages that are ready to be routed to a trading partner.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.

Return Values

Returns one of the following values:

string
If an outbound message was found and no errors were encountered, the message is returned as a string.

Boolean
Returns **#t** (true) if an outbound message was not found and no errors were encountered; otherwise returns **#f** (false) if the process was unsuccessful. Use **ux-get-error-str** to retrieve the corresponding error message.

Throws

Exception-InvalidArg.

Additional Information

ux-dequeue is called by the e*Xchange Transaction Polling e*Way.

It does the following:

- Receives a database connection handle from the e*Xchange Transaction Polling e*Way.
- Searches the Transaction Queue for the identity of the oldest record in the Stored Message table that needs to be sent to a trading partner.
- Decompresses the message before adding it to the Stored Message table.
- Deletes the corresponding identifier in the Transaction Queue.
- Returns a value to the e*Xchange Transaction Polling e*Way to indicate whether or not the process was successful and whether or not an outbound message was found. If successful, and if a message was found, the message is returned as a string to the e*Xchange Transaction Polling e*Way.

Internally within the API, a call is made to **ux-init-trans** to load the trading partner information into global memory. This is useful so that the e*Xchange Transaction Polling e*Way Collaboration has the information available to determine whether any

encryption or digital signatures are required on the message before it is sent to the trading partner.

Example

The following sample Monk script illustrates how the e*Xchange Transaction Polling e*Way:

1. Creates an outbound message structure named tran_poll.
2. Calls **ux-dequeue**, which returns a value stored in the parameter named result.
3. Determines whether the value of the result parameter is a Boolean value or a string, and then performs one of the following actions:
   - If the value is a string, then the message contained in the result parameter is inserted into the output message structure, and then forwarded to the server.
   - If the value is Boolean **#f** (false), **ux-get-error-str** is called to display the error message that was encountered.
   - If the value is Boolean #t (true), a message is displayed indicating that no message was returned from the database.

```
(define input-message-format-file-name "")
 (define output-message-format-file-name "tran_poll.ssc")
 (load "tran_poll.ssc")
 (define  result ( ux-dequeue  connection-handle))
 ( if (boolean? result )
      (begin
         (if (eq? result #f)
           (begin
              (display (ux-get-error-str))
              (newline)
           )
           (begin
              (display "There are no more msgs to retrieve
                  from DB\n")
           )
          )
 )
```

ux-duplicate-check

Syntax

```
(ux-duplicate-check connection-handle unique_id level direction)
```

Description

ux-duplicate-check checks whether the current Message Profile or B2B Protocol is a duplicate.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
unique_id	string	Required. The string that uniquely identifies the transaction. For an incoming message, this is the unique ID created by the Validation Rules Builder tool. For an outgoing message, it is the message ID taken from the message XML.
level	string	Required. The envelope level from which to obtain header segment data to check whether the current message is a duplicate. Valid values: I—Interchange T—Transaction
direction	string	Required. Indicates the direction of the message: I—Inbound O—Outbound

Return Values

Returns one of the following values:

string
Returns "Y" if the message is a duplicate; otherwise returns "N" if the message is not a duplicate.

Boolean
Returns **#f** (false) if the API fails and the message cannot be verified as unique or duplicated. Use **ux-get-error-str** to retrieve the error message.

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Additional Information

You can call **ux-duplicate-check** from a Monk script that handles inbound or outbound messages.

The Inbound Message Tracking table (es_mtrk_inb) contains one row of information for each inbound message and acknowledgment that has been processed and stored in the Message Storage table. The Outbound Message Tracking table (es_mtrk_outb) contains one row of information for each outbound message and acknowledgment that has been processed and stored.

Each row of the Inbound and Outbound Message Tracking tables contains a unique identifier for each message or acknowledgment. This identifier may consist of any combination of the following:

- Interchange control, functional group, or (depending on the messaging protocol used) message control numbers. X12 uses control numbers, RosettaNet does not.
- Message identifier code
- Version code
- Sending application or company name; for example, SAP or Sears

ux-duplicate-check looks at the es_mtrk_outb or es_mtrk_inb table to determine if the data just received is a duplicate. It takes a unique_id, direction ("I" or "O"), and level ("T" or "I"), and determines whether the combination of the unique_id, es_id (tpts_id if level = "T" or tpic_id if level = "I"), and es_opt ("TS" if level = "T" or "IC" if level = "I") already exist in the es_mtrk_outb (if direction "O"). If that combination already exists, then the data just received is considered a duplicate.

Example

```
(define unique_id "LA LA LA LA FA")
(define direction "O")
(define level "T")
(display "calling ux-duplicate-check\n")
(define res (ux-duplicate-check connection-handle unique_id level
direction))
(cond ((not (boolean? res))
       (cond ((string-ci=? "Y" res)
(display "It is a duplicate\n")
                )
               (else
(display "It is not a duplicate\n")
                )
         )
       )
       (else
            (display (ux-get-error-str))
            (newline)
        )
)
```

ux-func-ack-handler

Syntax

```
(ux-func-ack-handler connection-handle ext-list mtrk-ext-list
error_data)
```

Description

ux-func-ack-handler associates an inbound functional acknowledgment such as an X12 TA1 or 997 or a UN/EDIFACT CNTRL message with the appropriate outbound message or messages. If there are errors, it adds the error data to the database.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
ext-list	list of sub-lists	Required if the global Message Profile structure has not been previously loaded by ux-init-trans or ux-init-ts. Otherwise, this can be an empty list. Each sub-list being a pair of es_ext_detail.col_name and es_ext_data.ext_data_value. An example of each is given below: ▪ (list) ▪ (list (list "FUNC_ID_CODE" "FA"))
mtrk-ext-list	list of sub-lists	Required. Cannot be an empty list. Each sub-list is a pair of es_mtrk_ext_det.col_name and es_mtrk_ext_data.mtrk_data_value. Example: ▪ (list (list "I_CONTROL_NUM" "000000009"))
error_data	string	Required. Two possible formats: ▪ If there is error information—code^description~code^description (^ separates the values for an error and ~ separates the errors). ▪ If there is no error information—empty string ("").

The valid combination of values for mtrk-ext-list are listed below:

Possible combinations for X12

T_CONTROL_NUM, G_CONTROL_NUM, RESP_ID

T_CONTROL_NUM, G_CONTROL_NUM

G_CONTROL_NUM, RESP_ID

G_CONTROL_NUM

I_CONTROL_NUM

Possible combinations for UN/EDIFACT

IC_CONTROL_REF, TS_CONTROL_REF, RESP_ID

IC_CONTROL_REF, TS_CONTROL_REF

IC_CONTROL_REF, FG_CONTROL_REF, TS_CONTROL_REF, RESP_ID

IC_CONTROL_REF, FG_CONTROL_REF, TS_CONTROL_REF

IC_CONTROL_REF, FG_CONTROL_REF

IC_CONTROL_REF

Return values

returns one of the following values:

vector

Vector of mtrk_outb_ids (which are strings). For example:

```
#("8440" "8725")
```

Boolean

Returns **#t** (true) if nothing is retrieved; otherwise, returns **#f** (false) if an error is encountered.

Throws

None.

Additional information

ux-func-ack-handler updates the es_mtrk_outb, es_waiting_ack, and es_mtrk_error (if error data is included) database tables based on the message storage info associated with the acknowledgment just received.

The B2B Protocol global structure must be loaded, however the Message Profile global structure is not required. If only the B2B Protocol global structure is loaded, e*Xchange finds the appropriate tpts_ids in the es_tpts table using the ext-list.

ux-func-ack-handler updates es_mtrk_outb.ack_msg_id with the global message storage ID for the rows associated with the mtrk-ext-list, unique_id, and tpts_ids. It deletes any rows in the es_waiting_ack table associated with the es_mtrk_outb.mtrk_outb_id that has been updated with the global message storage ID.

It returns #t if no es_mtrk_outb.mtrk_outb_ids were updated. Otherwise it returns a vector of es_mtrk_outb.mtrk_outb_ids that were updated (using distinct es_mtrk_outb.orig_msg_ids).

If there is error data, ux-func-ack-handler inserts the data into the es_mtrk_error table.

Examples

In the following example, 8440 and 8725 are returned from the es_mtrk_outb table. 8440 and 8551 have the same orig_msg_id so only one of those is returned.

```
mtrk_outb_id     orig_msg_id
8440                 10
8551                 10
8725                 11
```

In the following example, the return value of ux-func-ack-handler is assigned to a variable named **var**, which is then displayed.

```
(define var (ux-func-ack-handler connection-handle (list) (list
  (list "I_CONTROL_NUM" "000000011")) "23^Twenty Three Desc~54^Fifty
  Four Desc"))
(display var)
```

ux-get-error-str

Syntax

```
(ux-get-error-str)
```

Description

ux-get-error-str is used when a function fails. It returns the last error message that was encountered.

Parameters

ux-get-error-str requires no parameters.

Return Values

string
Returns the last error message encountered by an e*Xchange API.

Throws

Exception-InvalidArg.

Additional Information

ux-get-error-str can be used for inbound or outbound messages. It does the following:

- Retrieves the message associated with the last error encountered by another e*Xchange API.
- Returns the error message to the calling API.

If **ux-get-error-str** is included in a display, the error shows in the log file.

When an e*Xchange API encounters a problem and cannot process a message as expected, an error message is stored in the memory buffer. **ux-get-error-str** retrieves this error message from the buffer.

Example

The following sample Monk script calls **ux-get-error-str** to retrieve the message associated with the last error encountered. In this example, the error message is displayed, followed by a new line.

```
(display (ux-get-error-str))
 (newline)
```

ux-get-fb-count

Syntax

```
(ux-get-fb-count connection-handle FB_UNIQUE_ID)
```

Description

ux-get-fb-count returns the total record count from es_mtrk_outb table with the same fast batch unique_id.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
FB_UNIQUE_ID	string	Required. The fast batch unique ID.

Return values

Returns one of the following values:

Number
Returns the total record count.

Throws

None.

Examples

```
(define fb_unique_id "AAAAA")
(define  total_fb_cnt
            (ux-get-fb-count connection-handle fb_unique_id)
)
```

ux-get-header

Syntax

```
(ux-get-header level type)
```

Description

ux-get-header returns values that are stored in the global structures (g_ic, and g_ts). The global structures contain data from the Trading Partner Profile. The B2B Protocol structure contains information about the protocol, version, direction, external trading partner, and communication protocol. The Message Profile structure contains information about the specific message, validation collaboration, transfer mode, and response messages. The structures are populated by calling **ux-init-trans** (**ux-init-ic** or **ux-init-ts**). Therefore, **ux-init-trans** must be called, and have successful execution, before **ux-get-header** is called. Otherwise, **ux-get-header** returns null values (empty strings).

Specifically, **ux-get-header** does the following:

- Returns a list of values retrieved from the global structures or a value that indicates that the API did not process successfully.

Parameters

Name	Type	Description
level	string	Required. The level from which to obtain TP Profile information. Valid values: I—B2B Protocol level information T—Message Profile level information A—Both levels
type	string	Required. O—Original message

Return Values

Returns one of the following values:

vector

Returns one of three vectors containing header data depending on the value of the level argument.

Level Argument	Vector Element Number	Data Type	Description
I	1	string	tpic_id
	2	string	tph_id
	3	string	tran_type
	4	string	version
	5	string	direction
	6	string	rtn_rcpt
	7	string	test_ind
	8	string	sec_key_type
	9	string	comm_port
	10	string	logical_name
	11	string	file_name
	12	string	user_name
	13	string	password
	14	string	host
	15	string	port
	16	list (of strings)	(ext_data_col_name, ext_data_col_value) These repeat for as may entries as there are in es_ext_data/ ex_ext_detail for this level I. There is an internal limit of 50 col_name/col_value pairs.

Level Argument	Vector Element Number	Data Type	Description
T	1	string	tpts_id
	2	string	tpic_id
	3	string	alt_id
	4	string	version
	5	string	tran_mode
	6	string	bus_collab
	7	string	db_collab
	8	string	msg_compress
	9	string	rtn_ts_id
	10	string	rtn_rcpt
	11	list (of strings)	(ext_data_col_name, ext_data_col_value) These repeat for as may entries as there are in es_ext_data/ ex_ext_detail for this level T. There is an internal limit of 50 col_name/col_value pairs.

Level Argument	Vector Element Number	Data Type	Description
A	1	string	tpic_id
	2	string	tph_id
	3	string	tran_type
	4	string	version
	5	string	direction
	6	string	rtn_rcpt
	7	string	test_ind
	8	string	sec_key_type
	9	string	comm_port
	10	string	logical_name
	11	string	file_name
	12	string	user_name
	13	string	password
	14	string	host
	15	string	port
	16	list (of strings)	(ext_data_col_name, ext_data_col_value) These repeat for as may entries as there are in es_ext_data/ ex_ext_detail for this level I. There is an internal limit of 50 col_name/col_value pairs.
	17	string	tpts_id
	18	string	tpic_id
	19	string	alt_id
	20	string	version
	21	string	tran_mode
	22	string	bus_collab
	23	string	db_collab
	24	string	msg_compress
	25	string	rtn_ts_id
	26	string	rtn_rcpt
	27	list (of strings)	(ext_data_col_name, ext_data_col_value) These repeat for as may entries as there are in es_ext_data/ ex_ext_detail for this level T. There is an internal limit of 50 col_name/col_value pairs.

Boolean

Returns **#f** (false) if the API did not process successfully. Use **ux-get-error-str** to retrieve the corresponding error message.

Throws

Exception-InvalidArg.

Additional Information

You can call **ux-get-header** from a Monk script that handles inbound or outbound TP Profiles. The ePM Batching e*Way calls this API to retrieve the enveloping information needed to process outbound batch messages.

A global TP Profile structure is a structure in memory that stores information for validating or assembling the message as required by its eBusiness Protocol during the processing of the message. The global structures are populated with information retrieved from trading partner profiles in the database with the **ux-init-trans**, **ux-init-ic**, or **ux-init-ts** APIs.

Example

The following sample Monk script calls the **ux-get-header** API with the assumption that the **ux-init-trans** processed successfully for the current original message. The **ux-get-header** API returns a list that contains data for the current message B2B Protocol level. The sample DO loop displays each string in the data list. If an error occurs, then **#f** is returned. The error can be identified by calling the **ux-get-error-str** API.

```
(define type "o")
 (define level "i")
 (define header-values (ux-get-header level type))
 (cond ((not (boolean? header-values))
        (do ((i 0 (+ i 1)) (value-count (vector-length
                                  header-values)))
       ((= i value-count))
          (display "header value <")
          (display i)
          (display "> = ")
          (display (vector-ref header-values i))
          (newline)
       )
     )
     (else
          (display (ux-get-error-str))
          (newline)
     )
 )
```

ux-get-key-cert

Syntax

```
(ux-get-key-cert connection-handle key-type [tpic-id])
```

Description

ux-get-key-cert retrieves security keys from the database. Use the **ux-get-key-cert** API for inbound or outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
key-type	string	The type of security key, certificate, or algorithm retrieved. Use the value obtained using sec_key_type in es_tpic. Only one key-type single character can be passed in. If sec_key_type contains more than one security trait, then these traits are separated by a vertical bar "|" and must be parsed into single characters before passing the value to **ux-get-key-cert**. Possible values are: E—Encryption certificate name S—Signature key name I — Signature Key Passphrase D—Decryption key name B—Decryption key passphrase V—Signature verification certificate name F—SSL Keystore Name G—SSL Keystore Type H—SSL Keystore Passphrase K—SSL Client Key Name T—SSL Client Key Type (only key name returned) C—SSL Client Certificate Name P—SSL Client Certificate Type (only key name returned) Y—Encryption algorithm (only key name returned) A—Signature algorithm (only key name returned) N—None
tpic-id	string	(Optional) The ID of the B2B Protocol for the trading partner profile. If you do not provide this parameter, then the value of tpic-id from the main global B2B Protocol structure stored in memory is used.

Return Value

Returns one of the following:

Boolean

Returns **#f** (false)—if an error was encountered; otherwise returns **#t** (true)—if no security certificate was found for supplied criteria.

vector

Returns a vector containing the following three elements if a security certificate was found:

Element Number	Type	Description
1	string	Security key name.
2	integer	Length of security key. This element is zero if there is no security key.
3	string	Security key. This element is empty if there is no security key associated with the security key name stored in the database.

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Example 1

This example passes in key-type of "K" and tpic_id of "161" and expects back a key-values vector containing three elements: SSL Client Key name in the first position, security key length in the second position, and SSL Client Key value in the last position.

```
(define key-type "K")
(define tpic-id "161")
(define key-values
    (ux-get-key-cert connection-handle key-type tpic-id))
(cond
    ((not (boolean? key-values))
        (do ((i 0 (+ i 1)) (value-count (vector-length key-values)))
            ((= i value-count))
            (display "key value <")
            (display i)
            (display "> = ")
            (display (vector-ref key-values i))
            (newline)))
            ; retrieve key-values
    (else
        (cond
            (key-values (display "No security key found\n"))
            (else (display (ux-get-error-str))
                (newline)))))
```

Example 2

Using a key-type of "A", this example relies on the global structure in memory, loaded from either **ux-init-trans** or **ux-init-ic**, to obtain tpic_id. It expects back only one useful value: the Signature Algorithm in the first position of the key-values vector, the second position contains 0, and the third position is empty.

```
(define key-type "A")
(define key-values (ux-get-key-cert connection-handle key-type))
(cond
    ((not (boolean? key-values))
        (do ((i 0 (+ i 1)) (value-count (vector-length key-values)))
            ((= i value-count))
            (display "key value <")
            (display i)
            (display "> = ")
            (display (vector-ref key-values i))
            (newline))
        ; retrieve key-values
    )
    (else
        (cond
            (key-values (display "No security key found\n"))
            (else
                (display (ux-get-error-str))
                (newline)))))
(define key-values
    (ux-get-key-cert conn-handle "E" "ENCRYPT_CERT_NAME"))
(cond
    ((not (boolean? key-values))
        (define sec-key-len (vector-ref key-values 0))
        (define sec-key (vector-ref key-values 1)))
    (else
        (if  key-values
            (display "No security was found for supplied criteria\n")
            (begin
                (display (ux-get-error-str))
                (newline)))))
(display "Testing ux-get-key-cert\n")
(define key-vec (ux-get-key-cert connection-handle "V"  "STC SIG" ))
(if (eq? key-vec #f)
    (begin
        (display (ux-get-error-str))
        (newline ))
    (begin
        (if (= 0 (string->number (vector-ref key-vec 0)))
            (begin
                (comment "No keys retrieved from the DB" ))
            (begin
                (display "Size of Key is = <")
                (display (vector-ref key-vec 0))
                (display ">\n\n")))))
```

ux-get-lock-ext-attrib-db

Syntax

```
(ux-get-lock-ext-attrib-db connection-handle column-name level)
```

Description

ux-get-lock-ext-attrib-db performs an insignificant update to the table first. This blocks if an external update is occurring to the record and also locks the record for the current process. Once the update has been performed, the specified attribute is retrieved from the DB and updated in the global structures and returned.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
column-name	string	Required. The name of the attribute to be retrieved.
level	string	Required. Indicates the level the value should be retrieved from. I - B2B Protocol level T - Message Profile level

Return Value

string

Returns a string containing the column value if found and successfully retrieved.

Boolean

Returns **#t** (true) if no values could be found for retrieval; otherwise returns **#f** (false)—if an error was encountered.

Throws

None.

Example

```
(define col-value (ux-get-lock-ext-attrib-db connection-handle
                    "IC_CONTROL_REF" "I"))
(if (!boolean col-value)
    (display (string-append "Column value: <"col-value">\n"))
    (if (eq? col-value #t)
        (display "No column value could be found in the DB\n")
        (display (string-append "Encountered error: <"(ux-get-
                        error-str)">\n"))
    )
)
```

ux-get-mtrk-attrib

Syntax

```
(ux-get-mtrk-attrib connection-handle list)
```

Description

ux-get-mtrk-attrib retrieves extended attributes for messages (B2B Protocol or Message Profile) stored in either the es_mtrk_inb or es_mtrk_outb tables. Uses of this include retrieving the Response ID or e*Insight Activity ID from an outbound message. This API is very useful when sending response messages back to e*Insight and associating the responses with a specific process and activity.

Use the **ux-get-mtrk-attrib** API for inbound or outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
list	list: direction unique_id level mtrk_id sub-list	Required. Information about the message. All list arguments must be strings, except for the sub-lists which are lists containing strings. All elements before the sub-list section are required, but can be empty strings (""), with the exception of direction and level, which return an error if no value is provided.
	List member	**Description**
	direction	Required. Indicates the direction of the message: I—Inbound O—Outbound
	unique_id	Optional only if mtrk_id is provided. The unique identifier for the original message.
	level	Optional only if mtrk_id is provided. Valid values: I—B2BProtocol level T—Message Profile level
	mtrk_id	Optional. Message tracking ID. If there is a list of extended attributes (sub-list), then mtrk_id or an empty string "" must be included.
	sub-list	Optional and repeating. The sub-list format is: "Column_Name" "Column_Value") ; may contain some of the extended attributes if already known.

Return Value

Boolean

Returns **#f** (false)—if an error was encountered; otherwise returns **#t** (true)—no extended attributes could be found for the given input data.

vector

Returns a vector containing the following 2*N* elements (where *N* is the number of extended attributes) if a extended attributes are found for the message:

Element Number	Type	Description
1	string	Column 1 name.
2	string	Column 1 value.
3	string	Column 2 name.
4	string	Column 2 value.
2*N*-1	string	Column *N* name.
2*N*	string	Column *N* value.

Throws

Exception-InvalidArg, Exception-Mapping, Exception-Catastrophic (can't be caught).

Example

```
(define unique_id "ACME ELIG1000000134")
(define mtrk-info (list "O"     ; direction
                        unique_id    ; unique-id
                        "T"          ; level
                        ""           ; represents mtrk_id needed
                                     ; because of the following sub-lists
                        (list "I_CONTROL_NUM" "000000002")))
(display "calling ux-get-mtrk-attrib\n")
(define ext-values (ux-get-mtrk-attrib connection-handle mtrk-info))
(display ext-values)
(newline)
(cond ((not (boolean? ext-values))
    (do ((i 0 (+ i 1)) (value-count (vector-length ext-values)))
        ((= i value-count))
            (display "mtrk ext value <")
            (display i)
            (display "> = ")
        (display (vector-ref ext-values i))
            (newline)
        )
      )
      (else
    (display (ux-get-error-str))
    (newline)
      )
)
(display "done calling ux-get-mtrk-attrib\n")
```

ux-get-seq-value

Syntax

```
(ux-get-seq-value connection-handle table_name)
```

Description

ux-get-seq-value retrieves the current sequence value for the specified table and returns the value in the seq_value parameter. To handle concurrency with multiple e*Ways accessing the same table sequence value simultaneously, this function catches locking or deadlocking errors up to 10 times and retry until sequence value is returned. If retrieval fails after the 10th time, an error indication is returned.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
table_name	string	Required. The table name for the sequence value.

Return Values

Returns one of the following values:

string
Returns a string containing the incremented sequence value.

Boolean
Returns **#f** (false) if a problem occurs.

Throws

None.

Example

```
(define seq_value (ux-get-seq-value connection-handle "es_sd_msg"))
```

ux-incr-control-num

Syntax

```
(ux-incr-control-num connection-handle level type)
```

Description

ux-incr-control-num increments the specified control number of an outbound Message Profile or B2B Protocol stored in the database and global structure.

Use the **ux-incr-control-num** API for outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
level	string	Required. The level of the control number to increment for the current message. Valid values: I—Interchange control number G—Functional group control number T—Transaction set control number
type	string	Required. Indicates which cached profile to update. Valid value: O—original struct

Return Values

Returns one of the following values:

string

string (incremented control number)—if the appropriate control number was successfully incremented.

Boolean

Returns **#f** (false)—if a problem occurred and the control number could not be duplicated. Use the **ux-get-error-str** API to retrieve the error message.

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Additional Information

ux-incr-control-num does the following:

- Receives a connection handle to the database and an indicator that specifies the type of control number to increment for the current message (interchange, functional group, or transaction set control number).
- If used at the transaction level, **ux-incr-control-num** increments, by one digit, the control number for the transaction stored in the database and in the global structure (the functional group and interchange control numbers are not changed).

- Returns either the incremented control number or a value indicating that the API did not process successfully.

You can call the **ux-incr-control-num** API from a Monk script that handles outbound TP Profiles. The ePM Batching e*Way calls this API to obtain the control number needed to process an outbound batch message.

All control numbers are stored in the es_ext_data/es_ext_detail tables, which correspond to the global structures.

The control number rolls over to 0 when it reaches 9999999999.

Example

The following sample Monk script calls the **ux-incr-control-num** API with the assumption that the **ux-init-trans** API processed successfully for the current B2B Protocol or Message Profile level. The **ux-incr-control-num** returns a string that contains the incremented functional group control number. If an error occurs, then **#f** (false) is returned. The error can be identified by calling the **ux-get-error-str** API.

```
(define type "O")
(define level "G")
(define control-number(ux-incr-control-num connection-
     handle level type))

  (cond ((not (boolean? control-number))
              (display "incremented control-number = <")
              (display control-number)
              (display ">\n")
        )
        (else
              (display (ux-get-error-str))
              (newline)
        )
  )
```

ux-init-exdb

Syntax

```
(ux-init-exdb connection-handle max_msg_size max_eWay_cnt
                                 eWay_instance_num)
```

Description

ux-init-exdb performs database and global variable initialization and binds for SQL statements and their parameters, and returns system default data stored in sb_defaults table.

Use the **ux-init-exdb** API on connection to the database.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
max_msg_size	string	The maximum size of message that can be stored in one blob. If a message is larger, it is broken into multiple pieces when stored.
max_eWay_cnt	string	The maximum number of ePM Batching e*Ways. This value is populated from the e*Way configuration setting and is set to the total number of ePM Batching e*Ways in the schema. For all other e*Way types the value should be set to “1”.
eWay_instance_num	string	The e*Way instance for this particular ePM Batching e*Way. This value is populated from the e*Way configuration setting. For all other e*Way types the value should be set to “1”.

Return Value

list

Returns a list containing sub lists of names and values as stored in system defaults, the sb_defaults table.

Boolean

Returns **#f** (false) if something fails. Use **ux-get-error-str** to see the error.

Throws

Exception-InvalidArg.

Additional Information

It is important that **ux-init-exdb** be called every time a connection is made, after login and the connection-handle is created.

Example

```
(if (db-login connection-handle HOSTNAME USERNAME PASSWORD)
    (begin
        (display "Logged in\n")
        (define sys-def (ux-init-exdb connection-handle 500000 5 3))
        (display sys-def)
        (cond ((not (boolean? sys-def))
            (do ((i 0 (+ i 1)) (value-count (vector-length sys-def)))
             ((= i value-count))
             (display "system default value <")
             (display i)
             (display "> = ")
             (display (vector-ref sys-def i))
             (newline)
            )
        (else
            (display (ux-get-error-str))
            (newline)
        )
    )
)
```

ux-init-ic

Syntax

```
(ux-init-ic connection-handle transact-info)
```

Description

ux-init-ic retrieves the trading partner profile, based on the items included in the *transact-info* list. The retrieved information is only for the B2B Protocol level and is stored in global structures. **ux-get-header** with level "I" can be used to extract the data from the global structures.

If the trading partner profile information has previously been loaded into the global structures, the function returns an indicator showing in which global structure the data is located. It does not make a query to the database.

Use the **ux-init-ic** API for inbound or outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
transact-info	list: alt_id company_name tran_type tpic_version tpts_version direction tran_mode tpic_id tpts_id rtn_ts_id comm_port logical_name file_name sub-list: (0->many) (optional)	Required. A set of identifying information contained within the current message. This information is matched against corresponding information in the e*Xchange database so that the correct trading partner profile can be retrieved. All list arguments must be strings, except for the sub-lists which are lists containing strings. All elements before the sub-list section are required, but can be empty strings (""), with the exception of **direction** which returns an error if no value is provided.
	List member	**Description**
	alt_id	The identification number of the trading partner, assigned by an external application (1–20 characters).
	company_name	The name of the company to which the message relates. If the trading partner is a subdivision of a larger company, this is the name of the company (1–35 characters).

Name	Type	Description
	tran_type	The code representing the name of the eBusiness Protocol used to format the message: possible values include X12, EDF and ROS. EDF represents UN/EDIFACT and ROS represents RosettaNet.
	tpic_version	The version of the protocol being used by the trading partner. This is the human-readable version, as used in the es_tpic table, not the code for the version taken directly from the B2B Protocol level. Example: 4040 for X12 version 4040, or 1.1 for RosettaNet version 1.1.
	tpts_version	The version of the protocol being used by the trading partner. This is the human-readable version, as used in the es_tpts table, not the code for the version taken directly from the Message Profile level. Since **ux-init-ic** is only applicable to B2B Protocol levels, any value supplied for this parameter is ignored. However, a placeholder ("") must be supplied.
	direction	Required. Indicates the direction of the message: I—Inbound O—Outbound
	tran_mode	The way in which messages are exchanged with the trading partner: I (interactive)—The message is sent to or from the trading partner individually to facilitate a "question and answer" type of B2B Protocol. B (batch)—The message is accumulated with other messages, which are then transmitted to or from the trading partner as a group. FB (fast batch)—A group of messages that are to be batched together in one interchange and identified by an associating unique ID.
	tpic_id	The record ID of the es_tpic table.

Name	Type	Description
	tpts_id	The record ID for the es_tpts table. Since **ux-init-ic** is only applicable to B2B Protocol levels, any value supplied for this parameter is ignored. However, a placeholder ("") must be supplied.
	rtn_ts_id	The record ID of the return Message Profile set. Since **ux-init-ic** is only applicable to the B2B Protocol level, any value supplied for this parameter is ignored. However, a placeholder ("") must be supplied.
	comm_port	The communications protocol used in the message; for example, HTTP.
	logical_name	The name of the trading partner, as set up in the Logical Name field in the General section of the B2B Protocol properties.
	file_name	The path and file name of the FTP file containing the message.
	(sub-list) (0->many)	Optional. The sub-list format is: level = "I" or "T" col_name col_value

Return Values

Returns one of the following values:

string

Returns "O" if the trading partner profile information was successfully loaded into the global structure.

Boolean

Returns **#f** (false)—if a corresponding trading partner profile was not found, or a problem occurred and the global structures were not initialized successfully. Use the **ux-get-error-str** API to retrieve the error message.

Note: *A failure generally means that the information passed into the function does not match any of the TP Profiles set up. Review this data and verify that it is correct.*

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Additional Information

ux-init-trans, **ux-init-ic**, and **ux-init-ts** only retrieve for active trading partners. From the e*Xchange Web Interface, verify the status of the intended TP Profile.

ux-init-ic does the following:

- Receives a set of identifying values contained within the current message and a connection handle to the database from the calling process.
- Retrieves information from the database for the trading partner profile that matches the set of identifying values.
- Populates the B2B Protocol global structure with information from the trading partner profile.
- Returns a value to the calling process that indicates whether or not the global structures were initialized successfully.

Trading partner profiles, which include information required by trading partners, are defined by users and are stored in the following e*Xchange tables: es_company, es_tph, es_tpcat, es_tpic, es_tpts, es_ext_data, and es_ext_detail. The **ux-init-trans** and associated APIs retrieve the information that is stored in these tables and load it into the global structures.

A global structure stores B2B Protocol data in memory, while e*Xchange processes the message. Other APIs access the information stored in the global structure to facilitate in processing the messages.

ux-init-ic includes sec_key_type as part of the global structure.

Example

The following sample Monk script calls the **ux-init-ic** API to populate the global structures. In this sample, the global structures can only be initialized if the following values match the values defined for a trading partner profile in the database:

- The alternate identification of the trading partner must be ACMEDIV1
- The name of the trading partner must be ACME Division ONE
- The sender identification number must be sender_id and the receiver identification number must be the ID number published by the receiver (this varies depending on the industry, but could be the DUNS number or some equivalent).
- The functional identification code of the message must be HB
- The EDI standard used to format the message must be X12
- The version number of the EDI standard used to format the message must be 4010
- The Message Profile identification number for the message must be 271
- The message must be an outbound B2B Protocol level.

If a trading partner profile in the e*Xchange database matches the information specified above, then data is retrieved from the trading partner profile and placed into the global structures. **#t** (true) is returned to indicate that the structures were successfully initialized.

If there is not a match, or if an error occurs, then **#f** (false) is returned. The error can be identified by calling the **ux-get-error-str** API.

```
(define transact-info (list "ACMEDIV1"     ; alt_id
                        "ACME Division ONE" ; name
                        "X12"               ; tran_type
                        "4010"              ; in es_tpic
                        "4010"              ; tpts_version in es_tpts
                        "I"                 ; direction
                        "B"                 ; tran_mode
                        ""                  ; tpic_id
                        ""                  ; tpts_id
                        ""                  ; rtn_ts_id
                        ""                  ; comm_port
                        ""                  ; logical_name
                        ""                  ; file_name
                        (list "I" "SENDER_ID" "sender_id")
                        (list "I" "RCVR_ID" "hliu")
                        (list "T" "FUNC_ID_CODE" "HB")
                        (list "T" "TRAN_SET_ID" "271")
                        (list "I" "VERSION" "00401") ; version to
                                                       match data
))
if (struc (ux-init-ic connection-handle transact-info)
     (display "ux-init-ic was successful\n")
   (begin
    (display "ux-init-ic was not successful\n")
    (display ux-get-error-str)
    (newline)
   )
)
```

ux-init-trans

Syntax

```
(ux-init-trans connection-handle transact-info)
```

Description

ux-init-trans retrieves the trading partner profile based on the items included in the transact-info list. **ux-init-trans** can retrieve information for both the B2B Protocol and Message Profile levels.

The information is stored in global structures. **ux-get-header** can be used to extract the data from the global structures.

Use the **ux-init-trans** API for inbound or outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
transact-info	list: alt_id company_name tran_type tpic_version tpts_version direction tran_mode tpic_id tpts_id rtn_ts_id comm_port logical_name file_name sub-list: (0->many) (optional)	A set of identifying information contained within the current message. This information is matched against corresponding information in the e*Xchange database so that the correct trading partner profile can be retrieved. All list arguments must be strings, except for the sub-lists which are lists containing strings. All elements before the sub-list section are required, but can be empty strings (""), with the exception of direction which returns an error if no value is provided.

Name	Type	Description
	List member	**Description**
	alt_id	The identification number of the trading partner, assigned by an external application (1–20 characters).
	company_name	The name of the company to which the message relates. If the trading partner is a subdivision of a larger company, this is the name of the company (1–35 characters).
	tran_type	The code representing the name of the eBusiness Protocol used to format the message: possible values include X12, EDF and ROS. EDF represents UN/EDIFACT and ROS represents RosettaNet.
	tpic_version	The version of the protocol being used by the trading partner. This is the human-readable version, as used in the es_tpic table, not the code for the version taken directly from the B2B Protocol level. Example: 4040 for X12 version 4040, or 1.1 for RosettaNet version 1.1.
	tpts_version	The version of the protocol being used by the trading partner. This is the human-readable version, as used in the es_tpic table, not the code for the version taken directly from the Message Profile level. Example: 4040 for X12 version 4040, or 1.1 for RosettaNet version 1.1.
	direction	Required. Indicates the direction of the message: I—Inbound O—Outbound

Name	Type	Description
	tran_mode	The way in which messages are exchanged with the trading partner: I (interactive)—The message is sent to or from the trading partner individually to facilitate a "question and answer" type of B2B Protocol level. B (batch)—The message is accumulated with other messages, which are then transmitted to or from the trading partner as a group. FB (fast batch)—A group of messages that are to be batched together in one interchange and identified by an associating unique ID.
	tpic_id	The record ID of the es_tpic table.
	tpts_id	The record ID for the es_tpts table.
	rtn_ts_id	The record ID of the return Message Profile set.
	comm_port	The communications protocol used in the message; for example, HTTP.
	logical_name	The name of the trading partner, as set up in the B2B Protocol General Section, Logical Name field.
	file_name	The path and file name of the FTP file containing the message.
	(sub-list) (0->many)	Optional. The sub-list format is: level = "I" or "T" col_name col_value

Return Values

Returns one of the following values:

string
Returns "O" if the trading partner profile information was successfully loaded into the global structure.

Boolean
Returns **#f** (false)—if a corresponding trading partner profile was not found, or a problem occurred and the global structures were not initialized successfully. Use the **ux-get-error-str** API to retrieve the error message.

Note: *A failure generally means that the informatio passed into the function does not match any of the TP Profiles set up. Review this data and verify that it is correct.*

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Additional Information

ux-init-trans, **ux-init-ic**, and **ux-init-ts** only retrieve for active trading partners. From the ePM Web Interface verify the status of the intended TP Profile.

ux-init-trans does the following:

- Receives a set of identifying values contained within the current message and a connection handle to the database from the calling process.
- Retrieves information from the database for the trading partner profile that matches the set of identifying values.
- Populates the global structure with information from the trading partner profile.
- Returns a value to the calling process that indicates whether or not the global structures were initialized successfully.

Trading partner profiles, which include information required by trading partners, are defined by users and are stored in the following e*Xchange tables:

- es_company
- es_tph,es_tpcat
- es_tpic,es_tpts
- es_ext_data
- es_ext_detail

ux-init-trans retrieves the information that is stored in these tables and loads it into the global structures.

Global structures store TP Profile data in memory while e*Xchange processes the message. Other APIs access the information stored in the global structures to facilitate in processing the messages.

Example

```
(define transact-info (list "ACMEDIV1"     ; alt_id
                       "ACME Division ONE" ; name
                       "X12"               ; tran_type
                       "4010"              ; in es_tpic
                       "4010"              ; in es_tpts
                       "I"                 ; direction
                       "B"                 ; tran_mode
                       ""                  ; tpic_id
                       ""                  ; tpts_id
                       ""                  ; rtn_ts_id
                       ""                  ; comm_port
                       ""                  ; logical_name
                       ""                  ; file_name
                       (list "I" "SENDER_ID" "sender_id")
                       (list "I" "RCVR_ID" "hliu")
                       (list "T" "FUNC_ID_CODE" "HB")
                       (list "T" "TRAN_SET_ID" "271")
                       (list "I" "VERSION" "00401") ; version to
                                                      match data
))
(if (ux-init-trans connection-handle transact-info)
     (display "ux-init-trans was successful\n")
   (begin
    (display "ux-init-trans was not successful\n")
    (display ux-get-error-str)
    (newline)
   )
)
```

ux-init-ts

Syntax

```
(ux-init-ts connection-handle transact-info)
```

Description

ux-init-ts retrieves the trading partner profile from the e*Xchange database based on the items included in the transact-info list. The retrieved information is only used by the Message Profile level and is stored in global structures. **ux-get-header** with level "T" can be used to extract the data from the global structures.

Use the **ux-init-ts** API for inbound or outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
transact-info	list: alt_id company_name tran_type tpic_version tpts_version direction tran_mode tpic_id tpts_id rtn_ts_id comm_port logical_name file_name sub-list: (0->many) (optional)	Required. A set of identifying information contained within the current message. This information is matched against corresponding information in the e*Xchange database so that the correct trading partner profile can be retrieved. All list arguments must be strings, except for the sub-lists which are lists containing strings. All elements before the sub-list section are required, but can be empty strings (""), with the exception of direction which returns an error if no value is provided.
	List member	**Description**
	alt_id	The identification number of the trading partner, assigned by an external application (1–20 characters).
	company_name	The name of the company to which the message relates. If the trading partner is a subdivision of a larger company, this is the name of the company (1–35 characters).
	tran_type	The code representing the name of the eBusiness Protocol used to format the message: possible values include X12, EDF and ROS. EDF represents UN/EDIFACT and ROS represents RosettaNet.

Name	Type	Description
	tpic_version	The version of the protocol being used by the trading partner. This is the human-readable version, as used in the es_tpic table. Since **ux-init-ts** is only applicable to Message Profiles, any value supplied for this parameter is ignored. However, a placeholder ("") must be supplied.
	tpts_version	The version of the protocol being used by the trading partner. This is the human-readable version, as used in the es_tpts table. Example: 4040 for X12 version 4040, or 1.1 for RosettaNet version 1.1.
	direction	Required. Indicates the direction of the message: I—Inbound O—Outbound
	tran_mode	The way in which messages are exchanged with the trading partner: I (interactive)—The message is sent to or from the trading partner individually to facilitate a "question and answer" type of B2B Protocol level. B (batch)—The message is accumulated with other messages, which are then transmitted to or from the trading partner as a group. FB (fast batch)—A group of messages that are to be batched together in one interchange and identified by an associating unique ID.
	tpic_id	The record ID of the es_tpic table.
	tpts_id	The record ID for the es_tpts table.
	rtn_ts_id	The record ID of the return inner envelope set.
	comm_port	The communications protocol used in the message; for example, HTTP.

Name	Type	Description
	logical_name	The name of the trading partner, as set up in the Logical Name field in the General tab of the Outer Envelope window.
	file_name	The path and file name of the FTP file containing the message.
	(sub-list) (0->many)	Optional. The sub-list format is: level = "I" or "T" col_name col_value

Return Values

Returns one of the following values:

string
Returns "O" if the trading partner profile information is successfully loaded into the global structure.

Boolean
Returns **#f** (false)—if a corresponding trading partner profile was not found, or a problem occurred and the global B2B Protocol structures were not initialized successfully. Use the **ux-get-error-str** API to retrieve the error message.

Note: *A failure generally means that the information passed into the function does not match any of the TP Profiles set up. Review this data and verify that it is correct.*

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Additional Information

ux-init-trans, **ux-init-ic**, and **ux-init-ts** only retrieve for active trading partners. From the e*Xchange Web Interface verify the status of the intended TP Profile.

ux-init-ts does the following:

- Receives a set of identifying values contained within the current message and a connection handle to the database from the calling process.
- Retrieves information from the database for the trading partner profile that matches the set of identifying values.
- Populates the Message Profile global structures with information from the trading partner profile.
- Returns a value to the calling process that indicates whether or not the global structures were initialized successfully.

Trading partner profiles, which include the information required by trading partners, are defined by users and are stored in the following e*Xchange tables: es_company, es_tph, es_tpcat, es_tpic, es_tpts, es_ext_data, and es_ext_detail. The **ux-init-trans** and associated APIs retrieve the information that is stored in these tables and loads it into the global structures.

A global structure stores data for TP Profile in memory while e*Xchange processes the message. Other APIs access the information stored in the global structures to facilitate in processing the messages.

Example

```
(define transact-info (list "ACMEDIV1" ; alt_id
       "ACME Division ONE" ; name
       "X12" ; tran_type
       "4010" ; tpic_version in es_tpic
       "4010" ; tpts_version in es_tpts
       "I" ; direction
       "B" ; tran_mode
       "" ; tpic_id
       "" ; tpts_id
       "" ; rtn_ts_id
       "" ; comm_port
       "" ; logical_name
       "" ; file_name
       (list "I" "SENDER_ID" "sender_id")
       (list "I" "RCVR_ID" "hliu")
       (list "T" "FUNC_ID_CODE" "HB")
       (list "T" "TRAN_SET_ID" "271")
       (list "I" "VERSION" "00401") ; version to match data
))
(if (ux-init-ts connection-handle transact-info)
     (display "ux-init-ts was successful\n")
   (begin
    (display "ux-init-ts was not successful\n")
    (display ux-get-error-str)
    (newline)
   )
)
```

ux-md5-digest

Syntax

```
(ux-md5-digest message)
```

Description

ux-md5-digest returns the MD5 digest of the input message.

Parameters

Name	Type	Description
message	string	Required. The message to digest.

Return Values

Returns one of the following values:

String
Returns the digested message, if the message is digested successfully.

Boolean

Returns **#f** (false) if the request did not process successfully. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

None.

Example

```
(define msg "AAAAAAAAAAAAAAAAAAAAAAAAA")
(define dig-msg (ux-md5-digest msg))
```

ux-ret-edf-batch-ts-msgs

Syntax

```
(ux-ret-edf-batch-ts-msgs connection-handle file_size)
```

Description

ux-ret-edf-batch-ts-msgs returns batch UN/EDIFACT messages to batch out.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
file-size	string	Contains the current size of batched messages. This value is updated and returned.

Return Values

Returns one of the following values:

vector
Returns a vector that contains three elements. The first element indicates the progressive size of the assembled batch message. If the file size exceeds 90% of the size specified in system defaults, this value is reset to "-1". The second element contains the size of the messages returned in this function call. The third element is a vector with as many elements as there are messages found. Each element of this vector is itself a vector. Each sub-vector has a message as its first element and sub-vectors as its subsequent elements. This contains the associated es_mtrk_outb and es_mtrk_ext_data record IDs.

Element Number	Type	Description
(all)	vector: total file size total message length vector	A vector containing a file size, message length and sub-vector.
(all)	vector: sub-vector 1 sub-vector 2 ... sub-vector N	A sub-vector containing a message and its associated tracking IDs as its elements.

Element Number	Type	Description	
	Sub-vector element	**Type**	**Description**
	message	string	A stored message to be sent using Batch transfer mode.
	vector: mtrk_outb_id fg_control_ref_mtrk_ext_data_id ic_control_ref_mtrk_ext_data_id	vector	A tracking number associated with the message.

Boolean

Returns **#t** when there are no messages to retrieve and no errors are encountered; otherwise returns **#f** (false) if the request did not process successfully. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

None.

Example

```
(define send-file-size "0")
(define mtrks-msgs (ux-ret-edf-batch-ts-msgs connection-handle send-file-
size))
 (cond
  ((> (vector-length mtrks-msgs) 0)
  (define send-immediate (vector-ref mtrks-msgs 0))
  (comment "If send-immediate is -1, the size of retrieved msgs exceeds
Maximum Batch File Size value in System Defaults" "")
  (display "\nSend Immediate : ")
  (display send-immediate)
  (newline)
  (define msg-size (vector-ref mtrks-msgs 1))
  (display "\ntotal_msg_size : ")
  (display msg-size)
  (newline)
  (define msgs-vec (vector-ref mtrks-msgs 2))
  (do ((I 0 (+ I 1)) (value-count (vector-length msgs-vec))) ((= I value-
count))
   (display "\nmtrks-msgs <")
   (display (+ I 1))
   (display "> = ")
   (display (vector-ref (vector-ref msgs-vec I) 0))
  (do ((j 1 (+ j 1)) (sub-val-count (vector-length (vector-ref msgs-vec I))))
((= j sub-val-count))
    (display "\nMtrk Outb Id: ")
    (display (vector-ref (vector-ref msgs-vec I) j) 0))
    (display "\nMtrk Ext Data Id <FG_CONTROL_REF>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec I) j) 1))
    (display "\nMtrk Ext Data Id <IC_CONTROL_REF>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec I) j) 2))
  )
 )
)
 (else
  (if (eq? mtrks-msgs #t)
   (display "Nothing to retrieve\n")
   (display (string-append "Encountered error: <" (ux-get-error-str) ">\n"))
  )
 )
)
```

ux-ret-edf-fb-ts-msgs

Syntax

```
(ux-ret-edf-fb-ts-msgs connection-handle file_size)
```

Description

ux-ret-edf-fb-ts-msgs returns fast batch UN/EDIFACT messages to batch out.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
file_size	string	Contains the current size of batched messages. This value is updated and returned.

Return Values

Returns one of the following values:

vector
Returns a vector that contains three elements. The first element indicates the progressive size of the assembled batch message. The second element contains the size of the messages returned in this function call. The third element is a vector with as many elements as there are messages found. Each element of this vector is itself a vector. Each sub-vector has a message as its first element and sub-vectors as its subsequent elements. This contains the associated es_mtrk_outb and es_mtrk_ext_data record IDs.

Element Number	Type	Description
(all)	vector: total file size total message length vector	A vector containing a file size, message length and sub-vector.
(all)	vector: fb_unique_id sub-vector 1 sub-vector 2 ... sub-vector N	A sub-vector containing a message and its associated tracking IDs as its elements.

Element Number	Type	Description	
	Sub-vector element	**Type**	**Description**
	message	string	A stored message to be sent using Batch transfer mode.
	vector: mtrk_outb_id fg_control_ref_mtrk_ext_data_id ic_control_ref_mtrk_ext_data_id	vector	A tracking number associated with the message.

Boolean

Returns **#t** when there are no messages to retrieve and no errors are encountered; otherwise returns **#f** (false)—if the request did not process successfully. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

None.

Example

```
(define mtrks-msgs (ux-ret-edf-fb-ts-msgs connection-handle "0"))
 (cond
  ((not (boolean? mtrks-msgs))
  (define msg-size (vector-ref mtrks-msgs 1))
  (display "\ntotal_msg_size : ")
  (display msg-size)
  (newline)
  (define msgs-vec (vector-ref mtrks-msgs 2))
  (display "Fast Batch Unique ID : ")
  (display (vector-ref msgs-vec 0))
  (newline)
  (do ((i 1 (+ i 1)) (value-count (vector-length msgs-vec))) ((= i value-
count))
   (display "\nmtrks-msgs <")
   (display (+ i 1))
   (display "> = ")
   (display (vector-ref (vector-ref msgs-vec i) 0))
  (do ((j 1 (+ j 1)) (sub-val-count (vector-length (vector-ref msgs-vec i))))
((= j sub-val-count))
    (display "\nMtrk Outb Id: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec i) j) 0))
    (display "\nMtrk Ext Data Id <FG_CONTROL_REF>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec i) j) 1))
    (display "\nMtrk Ext Data Id <IC_CONTROL_REF>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec i) j) 2))
  )
 )
 )
 (else
  (if (eq? mtrks-msgs #t)
   (display "Nothing to retrieve\n")
   (display (string-append "Encountered error: <" (ux-get-error-str) ">\n"))
  )
 )
)
```

ux-ret-X12-batch-ts-msgs

Syntax

```
(ux-ret-X12-batch-ts-msgs connection-handle file_size)
```

Description

ux-ret-edf-batch-ts-msgs returns batch X12 messages to batch out.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
file_size	string	Contains the current size of batched messages. This value is updated and returned.

Return Values

Returns one of the following values:

vector

Returns a vector that contains three elements. The first element indicates the progressive size of the assembled batch message. If the file size exceeds 90% of the size specified in system defaults, this value is reset to "-1". The second element contains the size of the messages returned in this function call. The third element is a vector with as many elements as there are messages found. Each element of this vector is itself a vector. Each sub-vector has a message as its first element and sub-vectors as its subsequent elements. This contains the associated es_mtrk_outb and es_mtrk_ext_data record IDs.

Element Number	Type	Description
(all)	vector: total file size total message length vector	A vector containing a file size, message length, and sub-vector.
(all)	vector: sub-vector 1 sub-vector 2 ... sub-vector N	A sub-vector containing a message and its associated tracking IDs as its elements.

Element Number	Type	Description	
	Sub-vector element	**Type**	**Description**
	message	string	A stored message to be sent using Batch transfer mode.
	vector: mtrk_outb_id ts_control_num_mtrk_ext_data_id fg_control_num_mtrk_data_id ic_control_num_mtrk_ext_data_id	vector	A tracking number associated with the message.

Boolean

Returns **#t** when there are no messages to retrieve and no errors are encountered; otherwise returns **#f** (false)—if the request did not process successfully. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

None.

Example

```
(define send-file-size "0")
(define mtrks-msgs (ux-ret-x12-batch-ts-msgs connection-handle send-file-
size))
(cond
 ((> (vector-length mtrks-msgs) 0)
  (define send-immediate (vector-ref mtrks-msgs 0))
  (comment "If send-immediate is -1, the size of retrieved msgs exceeds
Maximum Batch File Size value in System Defaults" "")
  (display "\nSend Immediate : ")
  (display send-immediate)
  (newline)
  (define msg-size (vector-ref mtrks-msgs 1))
  (display "\ntotal_msg_size : ")
  (display msg-size)
  (newline)
  (define msgs-vec (vector-ref mtrks-msgs 2))
  (do ((I 0 (+ I 1)) (value-count (vector-length msgs-vec))) ((= I value-
count))
   (display "\nmtrks-msgs <")
   (display (+ I 1))
   (display "> = ")
   (display (vector-ref (vector-ref msgs-vec I) 0))
  (do ((j 1 (+ j 1)) (sub-val-count (vector-length (vector-ref msgs-vec I))))
((= j sub-val-count))
    (display "\nMtrk Outb Id: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec I) j) 0))
    (display "\nMtrk Ext Data Id <T_CONTROL_NUM>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec I) j) 1))
    (display "\nMtrk Ext Data Id <G_CONTROL_NUM>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec I) j) 2))
    (display "\nMtrk Ext Data Id <I_CONTROL_NUM>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec I) j) 3))
   )
  )
 )
 (else
  (if (eq? mtrks-msgs #t)
   (display "Nothing to retrieve\n")
   (display (string-append "Encountered error: <" (ux-get-error-str) ">\n"))
  ) ))
```

ux-ret-X12-fb-ts-msgs

Syntax

```
(ux-ret-X12-fb-ts-msgs connection-handle file_size)
```

Description

ux-ret-edf-fb-ts-msgs returns fast batch X12 messages to batch out.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
file_size	string	Contains the current size of batched messages. This value is updated and returned.

Return Values

Returns one of the following values:

vector
Returns a vector that contains three elements. The first element indicates the progressive size of the assembled batch message. The second element contains the size of the messages returned in this function call. The third element is a vector with as many elements as there are messages found. Each element of this vector is itself a vector. Each sub-vector has a message as its first element and sub-vectors as its subsequent elements. This contains the associated es_mtrk_outb and es_mtrk_ext_data record IDs.

Element Number	Type	Description
(all)	vector: total file size total message length vector	A vector containing a file size, message length, and sub-vector.
(all)	vector: fb_unique_id sub-vector 1 sub-vector 2 ... sub-vector N	A sub-vector containing a message and its associated tracking IDs as its elements.

Element Number	Type	Description	
	Sub-vector element	**Type**	**Description**
	message	string	A stored message to be sent using Batch transfer mode.
	vector: mtrk_outb_id ts_control_num_mtrk_ext_data_id fg_control_num_mtrk_data_id ic_control_num_mtrk_ext_data_id	vector	A tracking number associated with the message.

Boolean

Returns **#t** when there are no messages to retrieve and no errors are encountered; otherwise returns **#f** (false)—if the request did not process successfully. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

None.

Example

```
(define mtrks-msgs (ux-ret-x12-fb-ts-msgs connection-handle "0"))
(cond
 ((not (boolean? mtrks-msgs))
  (define msg-size (vector-ref mtrks-msgs 1))
  (display "\ntotal_msg_size : ")
  (display msg-size)
  (newline)
  (define msgs-vec (vector-ref mtrks-msgs 2))
  (display "Fast Batch Unique ID : ")
  (display (vector-ref msgs-vec 0))
  (newline)
  (do ((i 1 (+ i 1)) (value-count (vector-length msgs-vec))) ((= i value-
count))
   (display "\nmtrks-msgs <")
   (display (+ i 1))
   (display "> = ")
   (display (vector-ref (vector-ref msgs-vec i) 0))
  (do ((j 1 (+ j 1)) (sub-val-count (vector-length (vector-ref msgs-vec i))))
((= j sub-val-count))
    (display "\nMtrk Outb Id: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec i) j) 0))
    (display "\nMtrk Ext Data Id <T_CONTROL_NUM>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec i) j) 1))
    (display "\nMtrk Ext Data Id <G_CONTROL_NUM>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec i) j) 2))
    (display "\nMtrk Ext Data Id <I_CONTROL_NUM>: ")
    (display (vector-ref (vector-ref (vector-ref msgs-vec i) j) 3))
   )
  )
 )
 (else
  (if (eq? mtrks-msgs #t)
   (display "Nothing to retrieve\n")
   (display (string-append "Encountered error: <" (ux-get-error-str) ">\n"))
  )
 )
)
```

ux-retrieve-997-error

Syntax

```
(ux-retrieve-997-error)
```

Description

ux-retrieve-997-error retrieves information from a 997 functional acknowledgment. It retrieves a vector containing 997 segment (AK2, AK3, AK4, or AK5) elements, originally stored by calling **ux-track-997-errors**. Segments are returned in the order stored. **ux-retrieve-997-error** returns a vector of segment elements from the head of the error linked-list and deletes that segment from the list. Hence, the head of the list is shifted to the next segment.

Use the **ux-retrieve-997-error** API for inbound messages.

Parameters

The **ux-retrieve-997-error** API requires no parameters.

Return Values

Returns one of the following values:

vector
Returns one of four types of vectors containing 997 segment information, if there are segments to retrieve.

Vector Type	Element Number	Type	Description
AK2	1	string	"AK2"
	2	string	tran_set_id
	3	string	ts_control_num
AK3	1	string	"AK3"
	2	string	seg_id_code
	3	string	seg_position
	4	string	loop_id_code
	5	string	syntax_error_code
AK4	1	string	"AK4"
	2	string	position_in_segment
	3	string	data_element_ref_no
	4	string	syntax_error_code
	5	string	bad_data

Vector Type	Element Number	Type	Description
AK5	1	string	"AK5"
	2	string	ts_ack_code
	3	string	ts_syntax_error_code_1
	4	string	ts_syntax_error_code_2
	5	string	ts_syntax_error_code_3
	6	string	ts_syntax_error_code_4
	7	string	ts_syntax_error_code_5

Note: *Each segment and element is returned in the order stored by* ***ux-track-997-errors****.*

Boolean

Returns **#t** (true)—if there are no more segments to retrieve; otherwise returns **#f** (false)—if the request did not process successfully. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

Exception-InvalidArg.

Example

The following Monk script example calls **ux-retrieve-997-error** with the assumption that **ux-track-997-errors** was executed successfully for at least one 997 segment. **ux-retrieve-997-error** returns a vector containing 997 segment elements. Segment seg_ak2345 contains the vector of returned values, and the internal DO loop displays each string in the vector. The external DO loop keeps calling **ux-retrieve-997-error** to retrieve each of the 997 segments until either **#t** (true) or **#f** (false) is encountered. When there are no more segments to retrieve, then **#t** (true) is returned. If an error occurs, then **#f** (false) is returned and the error string is printed by the display of **ux-get-error-str**.

```
(do  ((i 0 (+ i 1)) (seg_ak2345 ""))
      ((boolean? seg_ak2345))

      (set! seg_ak2345 (ux-retrieve-997-error))

      (cond ((not (boolean? seg_ak2345))
         (do ((i 0 (+ i 1)) (value-count (vector-length
         seg_ak2345)))
              ((= i value-count))
              (display "AK2345 element <")
              (display i)
              (display "> = ")
              (display (vector-ref seg_ak2345 i))
              (newline)
        )
       ; retrieve ak2345 values
      )
      (else
       (if seg_ak2345
             (display "No more to retrieve\n")
         (begin
             (display (ux-get-error-str))
```

```
                    (newline)
                  )
                 )
                );else
               );cond
          );do
```

ux-retrieve-997-error-tail

Syntax

```
(ux-retrieve-997-error-tail)
```

Description

ux-retrieve-997-error-tail retrieves the 997 segment (AK2, AK3, AK4, or AK5) that is at the end of the list. Once a vector of segment elements is returned, this API also deletes that segment from the list.

Use the **ux-retrieve-997-error-tail** API for inbound messages.

Parameters

The **ux-retrieve-997-error-tail** API requires no parameters.

Return Values

Returns one of the following values:

vector
Returns one of four types of vectors containing 997 segment information, if there is a segment to retrieve.

Vector Type	Element Number	Type	Description
AK2	1	string	"AK2"
	2	string	tran_set_id
	3	string	ts_control_num
AK3	1	string	"AK3"
	2	string	seg_id_code
	3	string	seg_position
	4	string	loop_id_code
	5	string	syntax_error_code
AK4	1	string	"AK4"
	2	string	position_in_segment
	3	string	data_element_ref_no
	4	string	syntax_error_code
	5	string	bad_data
AK5	1	string	"AK5"
	2	string	ts_ack_code
	3	string	ts_syntax_error_code_1
	4	string	ts_syntax_error_code_2
	5	string	ts_syntax_error_code_3
	6	string	ts_syntax_error_code_4
	7	string	ts_syntax_error_code_5

Boolean

Returns **#t**—if there are no more segments to retrieve; otherwise returns **#f** (false)—if the request did not process successfully. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

Exception-InvalidArg.

Example

The following Monk script example calls **ux-retrieve-997-error-tail** with the assumption that **ux-track-997-errors** was executed successfully for at least one 997 segment. **ux-retrieve-997-error-tail** returns a vector containing 997 segment elements. Segment seg_ak2345 contains the vector of returned values, and the internal DO loop displays each string in the vector. The external DO loop keeps calling **ux-retrieve-997-error-tail** to retrieve each of the 997 segments until either **#t** or **#f** is encountered. When there are no more segments to retrieve, then **#t** is returned. If an error occurs, then **#f** is returned and the error string is printed by the display of **ux-get-error-str**.

```
(do ((i 0 (+ i 1)) (seg_ak2345 ""))
    ((boolean? seg_ak2345))

    (set! seg_ak2345 (ux-retrieve-997-error-tail))

    (cond ((not (boolean? seg_ak2345))
        (do ((i 0 (+ i 1)) (value-count (vector-length seg_ak2345)))
            ((= i value-count))
            (display "AK2345 element <")
            (display i)
            (display "> = ")
           (display (vector-ref seg_ak2345 i))
            (newline)
            (sleep 5)
        )
       ; retrieve ak2345 values
      )
      (else
        (if seg_ak2345
           (display "No more to retrieve\n")
         (begin
          (display (ux-get-error-str))
           (newline)
         )
        )
      )
    ) ; cond
); do
```

ux-retrieve-message

Syntax

(ux-retrieve-message connection-handle msg-id)

Description

ux-retrieve-message retrieves a message from the es_msg_ascii table or the es_msg_binary table, depending on whether the message is compressed or not. Msg-id is used to identify the message.

Parameters

Name	Type	Description
connection-handle	connection-handle Required.	The previously established connection to the database.
msg-id	String	The message id as saved in the es_msg_storage table.

Return Values

Returns one of the following values:

string
Returns a Monk string representing the found message, when the function executes successfully.

Boolean
Returns **#f** (false) when the function fails to complete successfully.

Throws

Exception-InvalidArg, Exception-Mapping, Exception-Catastrophic (can't be caught).

Additional Information

ux-retrieve-message retrieves the entire message, even though it may be saved in the database in multiple rows.

Examples

The following Monk script example calls **ux-retrieve-message**. This script makes three assumptions:

- That **ux-init-trans** was executed successfully for the given message.
- That a connection to the database, conn-handle, has been established before **ux-retrieve-message** is called.
- That all variables in the first two statements below have been properly defined with values either from the message itself or from the partner profile in the database.

If **ux-retrieve-message** fails, then the error, a user defined function **SendFailureNotification**, is called.

```
(set! msg_content (ux-retrieve-message connection-handle msg_id))
(if msg_content
    (begin
        (display (string-append "Got msg_content=<"
                                msg_content ">\n"))
        (newline)
        (try ($event-parse input msg_content)
             (catch (always (set! success #f)))))
    (begin (set! success #f))
)
```

ux-return-receipt

Syntax

```
(ux-return-receipt level type)
```

Description

ux-return-receipt determines whether a return receipt (response) for an event is expected at the specified level. Use the **ux-return-receipt** API for inbound or outbound messages.

Parameters

Name	Type	Description
level	string	Required. The return receipt level. Acceptable values: I—B2B Protocol level information T—Message Profile level information
type	string	Required. O— original struct

Return Values

Returns one of the following values:

string

Returns "Y" if a return receipt is expected; otherwise returns "N" if a return receipt is not expected.

Boolean

Returns **#f** (false)—if the request did not process successfully. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

Exception-InvalidArg.

Example

The following Monk script example calls **ux-return-receipt** with the assumption that **ux-init-trans** was executed successfully for the given transaction. **ux-return-receipt** sets result to equal "Y" if a return receipt is expected for the B2B Protocol level. Otherwise, result equals "N", which means a return receipt is not expected for the B2B Protocol level. If an error occurs, then **#f** (false) is returned and the error string is printed by the display of **ux-get-error-str**.

```
(define level "I")
  (define res (ux-return-receipt level type))
  (cond ((not (boolean? res))
         (cond ((string-ci=? "Y" res)
                (display "Return receipt expected\n")
                )
                (else
                (display "Return receipt not expected\n")
                )
         )
        )
        (else
                (display (ux-get-error-str))
                (newline)
        )
  )
```

ux-set-fb-overdue

Syntax

```
(ux-set-fb-overdue connection-handle)
```

Description

ux-set-fb-overdue checks the database for fast batch settings. If it finds records that match the specified criteria, it sets the BATCH_SEND_IMM flag to Y. This value represents any fast batch record that has exceeded its timeout used for fast batch transactions.

ux-set-fb-overdue checks for the following values:

- BATCH_SEND_IMM = "N"
- es_mtrk_outb.es_id = g_ts.tpts_id
- es_mtrk_outb.es_opt = "TS"
- es_mtrk_outb.created_time <= current - fb_timeout in sb_defaults

If a record matches the above criteria, then BATCH_SEND_IMM is set to "Y". This criteria represents any (fast) batch record that has exceeded its timeout. The ePM Batching e*Way then picks up the timed out records to send out.

Use the **ux-set-fb-overdue** API for outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.

Return Values

Boolean

Returns **#t** (true)—if no errors are encountered.

Returns **#f** (false)—if errors are encountered.

Throws

Exception-InvalidArg

Additional Information

ux-init-trans or ux-init-ts must be called before this api is executed.

Example

```
(if (eq? #t (ux-set-fb-overdue connection-handle))
(display "ux-set-fb-overdue was successful! \n")
(display (string-append "ux-set-fb-overdue failed with error: <"(ux-
get-error-str) ">\n"))
)
```

ux-store-msg

Syntax

```
(ux-store-msg connection-handle msg store-info raw_msg)
```

Description

ux-store-msg stores a message (Message Profile or B2B Protocol level) in the e*Xchange database, inserting entries in multiple tables in the process. If the message is compressed before it is stored, it is stored in es_msg_binary. If it is not compressed, it is stored in es_msg_ascii. In either case, a record is inserted into es_msg_storage that has a column, "compressed", indicating the table in which the message is stored.

Depending on whether the message is inbound or outbound, **ux-store-msg** makes an additional entry in either the es_mtrk_inb or es_mtrk_outb table. This table indicates send or receive time, send count, transaction count, and so on. If there is information specific to the transaction type (RosettaNet or X12) to store in the message, this data is stored in es_mtrk_ext_data and associated to specific records in es_mtrk_ext_det.

Use the **ux-store-msg** API for inbound or outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
msg	string	Required. The processed message to be stored. The processed message must be provided for an inbound message. For an outbound message, the processed message does not need to be stored if the raw message is stored. If you do not want to store the processed message, then use an empty string ("" Note: If both the raw and processed messages are empty strings then ux-store-msg fails. Storage location: Inbound—es_mtrk_inb Outbound—es_mtrk_outb

Name	Type	Description
store-info	list: direction unique_id type error_data level tp_loc msg_being_sent compressed mtrk_id sub-list: Optional: (0->many)	Required. List of items regarding the Message Profile. All list arguments must be strings, except for the sub-lists which are lists containing strings. All elements before the sub-list section are required, but can be empty strings (""), with the exception of direction, unique_id, type, and level, which return an error if no value is provided.
	List member	**Description**
	direction	Required. Indicates the direction of the message: I—Inbound O—Outbound
	unique_id	Required. The unique identifier for the original message.
	type	Required. The kind of message being stored. The following are valid values: "O"—Original (only the original message is stored) "W"—Wrapped (the original message is stored in one location and the wrapped message is stored in another location) "C"—Combined (only the wrapped message stored)
	error_data	Error information. Optional—code^description~code^description (^ separates the values for an error and ~ separates the errors).
	level	Required. The storage level. Valid values: "I"—B2B Protocol "T"—Message Profile
	msg_being_sent	Required for original and wrapped: "Y"—Message is being sent to e*Gate "N"—Message is not being sent to e*Gate
	compressed	Indicates whether the message is to be compressed before the msg is stored in the database: "Y"—Yes "N"—No

Name	Type	Description
	mtrk_id	Optional—future versions may use this value to store messages.
	sub-list	Optional. The sub-list format is: level = "I" or "T" col_name col_value
	tp_loc	location of trading partner. Valid value: "O" —original
raw_msg	string	Optional. The raw transaction to be stored in the database.

Return Values

Returns one of the following values:

string

Returns **string that contains the mtrk_id** (mtrk_outb_id for outbound or mtrk_inb_id for inbound)—if the message is successfully stored.

Boolean

Returns **#f** (false)—if the message is not successfully stored. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

Exception-InvalidArg, Exception-Mapping, Exception-Catastrophic (can't be caught).

Additional Information

If the length of the message is greater than the max size specified by the **ux-init-exdb** API, the message is broken up and stored in multiple rows in es_msg_binary (if compressed), es_msg_ascii (if not compressed), or es_msg_security (if encrypted or contains a digital signature).

All messages, whether or not they are created by e*Xchange, are stored in the Stored Messages table (es_msg_storage).

Example

The example below shows how to create the store information list.

```
(define store-info (list "O";  direction
             "TESTVAL119"; unique_id
             "W"; type
             "123^Not feeling so good~345^Pain in toe"; error_data
             "T"; level
             "O"; original
             "Y"; msg_being_sent
             "Y" ; compressed
             ""  ; mtrk_id
             (list "I_CONTROL_NUM" "556")
             (list "G_CONTROL_NUM" "776")
             (list "T_CONTROL_NUM" "886")
           )
)
```

The example below shows how to call the API.

```
(define raw_msg "")
(define msg "abcdefghijklmnopqrstuvwxyz1234567890")
(define mtrk-id (ux-store-msg connection-handle msg store-info
                                                              raw_msg))
(if (not (boolean? mtrk-id))
  (begin
   (display "Storing of message succeeded!\n")
   (display "returned mtrk_id = <")
   (display mtrk-id)
   (display ">\n")
  )
  (begin
   (display "Storing of message failed!\n")
   (display (ux-get-error-str))
   (newline)
  )
)
```

ux-store-msg-errors

Syntax

```
(ux-store-msg-errors connection-handle mtrk_id direction errorlist)
```

Description

Stores errors in the e*Xchange database that are associated with a message that is already stored in the database.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
mtrk_id	string	Required. Message tracking ID that errors are associated with.
direction	string	Required. Direction of message. Either O—Outbound I—Inbound
errorlist	string	Required. Use the following format: code1^desc1~code2^desc2 ~code3^desc3... Code is the numeric identifier for the error. Desc explains the error. The code and description are separated by a "^", and each code/desc pair is separated by a "~".

Return Values

Boolean

Returns **#t** (true)—if the errors are stored successfully; otherwise returns **#f** (false)—if the errors do not store properly. Use **ux-get-error-str** to see the error.

Throws

Exception-InvalidArg, Exception-Mapping, Exception-Catastrophic (can't be caught).

Example

```
(if (ux-store-msg-errors conn-handle "12" "O" "45^Invalid Country
    Code~56^Invalid Zipcode")
    (display "Stored errors successfully\n")
    (display "Failed to store errors\n")
)
```

ux-store-msg-ext

Syntax

```
(ux-store-msg-ext connection-handle msg store-info store-mode
                  msg_storage_id raw_msg)
```

Description

This API is similar to **ux-store-msg** except for the addition of two arguments store-mode and msg_storage_id. **ux-store-msg-ext** stores a message (Message Profile or B2B Protocol) in the e*Xchange database, inserting entries in multiple tables in the process. If the message is compressed before it is stored, it is stored in es_msg_binary. If it is not compressed, it is stored in es_msg_ascii. In either case, a record is inserted into es_msg_storage that has a column, "compressed", indicating the table in which the message is stored.

Depending on whether the message is inbound or outbound, **ux-store-msg-ext** makes an additional entry in either the es_mtrk_inb or es_mtrk_outb table. This table indicates send or receive time, send count, transaction count, and so on. If there is information specific to the transaction type (RosettaNet or X12) to store in the message, this data is stored in es_mtrk_ext_data and associated to specific records in es_mtrk_ext_det.

Use the **ux-store-msg-ext** API for inbound or outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
msg	string	Required. The transaction to be stored. Storage location: Inbound—es_mtrk_inb Outbound—es_mtrk_outb
store-info	list: direction unique_id type error_data level tp_loc msg_being_sent compressed mtrk_id sub-list: Optional: (0->many)	Required. List of items regarding the Message Profile. All list arguments must be strings, except for the sub-lists which are lists containing strings. All elements before the sub-list section are required, but can be empty strings (""), with the exception of direction, unique_id, type, and level, which return an error if no value is provided.

Name	Type	Description
	List member	**Description**
	direction	Required. Indicates the direction of the message: I—Inbound O—Outbound
	unique_id	Required. The unique identifier for the original message.
	type	Required. The kind of Message Profile being stored. The following are valid values: "O"—Original (only the original message is stored) "W"—Wrapped (the original message is stored in one location and the wrapped message is stored in another location) "C"—Combined (only the wrapped message stored)
	error_data	Error information. Optional—code^description~code^description (^ separates the values for an error and ~ separates the errors).
	level	Required. The storage level the control number represents. Valid values: "I"—B2B Protocol level information "T"—Message Profile level information
	tp_loc	location of trading partner "O" original or "A" ack (response struct)
	msg_being_sent	Required for original and wrapped: "Y"—Message is being sent to e*Gate "N"—Message is not being sent to e*Gate
	compressed	Indicates whether the message is to be compressed before the msg is stored in the database: "Y"—Yes "N"—No
	mtrk_id	Optional—future versions may use this value to store messages.
	sub-list	Optional. The sub-list format is: level = "I" or "T" col_name col_value

Name	Type	Description
store-mode	string	0—Saves the message and updates the message tracking table. 1—Saves the message only (no update to the message tracking table). 2—Updates the message tracking table only (the message is not saved)
msg_storage_id	string	Required for store-mode 2.
raw_msg	string	Required. The raw transaction to be stored in the database. If there is no raw transaction associated with this message, or you do not want to store the raw message, then use an empty string ("").

Return Values

Returns one of the following values:

vector

Returns a vector containing the following two elements if a security certificate was found:

Element Number	Type	Description
1	string	mtrk_id (message tracking ID)
2	string	msg_storage_id

Boolean

Returns **#f** (false)—if the message is not successfully stored. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

Exception-InvalidArg, Exception-Mapping, Exception-Catastrophic (can't be caught).

Additional Information

If the length of the message is greater than the max size specified by the **ux-init-exdb** API, the message is broken up and stored in multiple rows in es_msg_binary (if compressed), es_msg_ascii (if not compressed) or es_msg_security (if encrypted or contains a digital signature).

All messages, whether or not they are created by e*Xchange, are stored in the Stored Messages table (es_msg_storage).

Example

```
(define raw_msg "")
(define store_mode 0)
(define store_rtn_vec (ux-store-msg-ext connection-handle
                      output_data store_orig_info store_mode
                      msg_storage_id raw_msg))
(if (boolean? store_rtn_vec)
    (begin
        (eX-ePM-log "ux-store-msg-ext failed\n"))
    (begin
        (set! msg_storage_id (vector-ref store_rtn_vec 1))
        (set! mtrk-id (vector-ref store_rtn_vec 0))
        (set! store_mode 2)))
```

ux-store-shutdown-uid

Syntax

```
(ux-store-shutdown-uid connection-handle list)
```

Description

ux-store-shutdown-uid inserts a row into the es_sd_data table with es_id, es_opt and unique_id when an eX_ePM shutdown occurs.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
list	list	Required. Contains any number of lists containing es_id, es_opt, and unique_id.
	List member	**Description**
	es_id	Required. This contains the tpts_id if es_opt is “T”; otherwise, this contains the tpic_id if es_opt is “I”.
	es_opt	Required. The message level.Valid values: "I"—B2B Protocol level information "T"—Message Profile level information
	unique_id	Required. A string that uniquely identifies the transaction.

Return Values

Boolean

Returns **#t** (true)—if the errors are stored successfully; otherwise returns **#f** (false)—if the errors do not store properly. Use **ux-get-error-str** to see the error.

Throws

None.

Example

```
(define store_result (ux-store-shutdown-uid connection-handle
                          (list    (list "1" "T" "AAAA")
                                   (list "1" "T" "BBBBB")
                          )
                     )
)
```

ux-track-997-errors

Syntax

```
(ux-track-997-errors list of AK2, AK3, AK4, or AK5 elements)
```

Description

ux-track-997-errors stores the error information for a 997 in a linked-list, so errors can be tracked as they are encountered in a validation. The error information is used to create a 997 functional acknowledgment.

Use the **ux-track-997-errors** API for inbound messages.

The head of the linked-list must be an AK2 segment.

Parameters

Name	Type	Description
List of AK2, AK3, AK4, or AK5 elements	list Lists vary based on type provided.	All list elements must be strings. Each list must begin with a segment_code, such as "AK2", "AK3", "AK4", or "AK5". The first segment to store must be an AK2 before an AK3, AK4, or AK5 are accepted. Each segment is stored in the order that **ux-track-997-errors** is called.
	List Type	**Description**
	AK2: tran_set_id ts_control_num	The transaction set (Message Profile) response header.
	AK3: seg_id_code seg_position loop_id_code syntax_error_code	A data segment note. loop_id_code and syntax_error_code are optional; however, "" must be in place if no value is to be stored.
	AK4: position_in_segment data_element_ref_no syntax_error_code bad_data	A data element note. data_element_ref_no and bad_data are optional; however, "" must be in place if no value is to be stored.
	AK5: ts_ack_code ts_syntax_code_error_1 ts_syntax_code_error_2 ts_syntax_code_error_3 ts_syntax_code_error_4 ts_syntax_code_error_5	The transaction set response trailer.

Return Values

Boolean

Returns **#t** (true)—if the strings are successfully stored; otherwise returns **#f** (false)—if the storage attempt is unsuccessful. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

Exception-InvalidArg, Exception-Mapping.

Example

The following Monk script example calls **ux-track-997-errors** with the assumption that **ux-track-997-errors** was executed successfully for an "AK2" and "AK3" previously. **ux-track-997-errors** first validates that the given segment_code "AK4" is valid. If valid, then a node is added to the end of the linked-list of 997 segments containing the provided AK4 information. If successful, then **ux-track-997-errors** returns **#t** and displays "Tracking 997 errors succeeded!". If an error occurs, then **#f** is returned, displays "Tracking 997 errors failed!" and prints the error string by the display of **ux-get-error-str**.

```
(define ak2345_data (list  "AK4" ; segment_code
                           "567" ; postion_in_segment
                           "" ; data_element_ref_no
                           "67" ; syntax_error_code
                           "Weshington, DC" ; bad data
                    )
)
(if (ux-track-997-errors ak2345_data)
       (display "Tracking 997 errors succeeded!\n")
       (begin
            (display "Tracking 997 errors failed!\n")
            (display (ux-get-error-str))
            (newline)
       )
)
```

ux-update-batch-imm

Syntax

```
(ux-update-batch-imm connection-handle update-value type)
```

Description

ux-update-batch-imm updates the value in the e*Xchange database that is used to determine whether a transaction is ready to sent out using batch transfer mode. This corresponds to the value for **SEND BATCH IMMEDIATE** displayed on the **Extended** tab of the Message Profile for a transaction.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
update-value	string	Required. Y—Yes N—No
type	string	O—Original A—Acknowledgment/Response

Return Values

Boolean

Returns **#t** (true)—if the transaction is updated successfully with the update-value; otherwise returns **#f** (false)—if the transaction fails to update. Use **ux-get-error-str** to see the error.

Throws

Exception-InvalidArg, Exception-Mapping, Exception-Catastrophic (can't be caught).

Example

```
(if (ux-update-batch-imm connection-handle "N" message_type)
    (begin)
    (begin
        (eX-ePM-log "ux-update-batch-imm failed")))
```

ux-update-control-num

Syntax

```
(ux-update-control-num connection-handle level type control-num)
```

Description

ux-update-control-num replaces the specified control number in the database and global structure with the one provided. If the control number provided contains leading zeros, they are stripped off before replacing the number.

ux-update-control-num updates the control number provided for the given transaction level (I = i_control_num, G = g_control_num, T = ts_control_num). If the given control number is invalid (contains characters other than digits) then **ux-update-control-num** returns an error.

Use the **ux-update-control-num** API for outbound messages.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
level	string	Required. The level the control number represents. Valid values: I—Interchange control number G—Functional group control number T—Transaction set control number
type	string	Required. Indicates which global struct to query. Acceptable value: O— original struct
control-num	string	Required. The new control number value which replaces the existing control number in the database.

Return Values

Boolean

Returns **#t** (true)—if the control number is successfully updated; otherwise returns **#f** (false)—if the control number is not successfully updated. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Example

The following Monk script example calls **ux-update-control-num** with the assumption that **ux-init-trans** was executed successfully for the given Message Profile. **ux-update-control-num** first checks to be sure that the control-num contains all digits.

An update of the database sets es_ext_data.ext_data_value = 55 where es_id = tpid_id in the global structure and es_ext_detail.col_name = "G_CONTROL_NUMBER". A

commit immediately follows the update. Also, the control number in the global structure gets updated to 55. If successful, then **ux-update-control-num** returns **#t** and "Update of control-num succeeded!" is displayed. If an error occurs, then **#f** is returned and the error string is printed by the display of **ux-get-error-str**.

```
(define type "O")
(define level "G")
(define control-num "55")
 (if (ux-update-control-num connection-handle level
       control-num)
       (display "Update of control-num succeeded!\n")
       (begin
             (display "Update of control-num failed!\n")
             (display (ux-get-error-str))
             (newline)
       )
```

ux-update-last-batch-send-time

Syntax

```
(ux-update-last-batch-send-time connection-handle send_time type)
```

Description

ux-update-last-batch-send-time updates the batch last send time using input time.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
send_time	string	Required. The time used for updating.
type	string	Required. Indicates which cached profile to update. Acceptable value: O— original struct

Return Values

Boolean

Returns **#t** (true)—if the batch last send time is successfully updated; otherwise returns **#f** (false)—if the batch last send time is not successfully updated. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

None.

Example

```
(define upd-result   (ux-update-last-batch-send-time
                         connection-handle
                         "01/01/2001 12:00:00"
                         "O"
                     )
)
```

ux-upd-mtrk-data-item

Syntax

```
(ux-upd-mtrk-data-item connection-handle id data_value)
```

Description

ux-upd-mtrk-element updates the es_mtrk_ext_data.mtrk_data_value for the specified primary key record id.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
id	string	Required. The es_mtrk_ext_data.mtrk_data_id.
data_value	string	Required. The value to update the table with.

Return Values

Boolean

Returns **#t** (true)—if the value is successfully updated; otherwise returns **#f** (false)—if the value is not successfully updated. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

None.

Example

```
(define ic_ref_id "1000")
(define ic_control_num "000000111")
(define upd-result   (ux-upd-mtrk-data-item
                         connection-handle
                         ic_ref_id
                         ic_control_num
                     )
)
```

ux-upd-mtrk-element

Syntax

```
(ux-upd-mtrk-element connection-handle col_name1 col_value1 col_name2
col_value2)
```

Description

ux-upd-mtrk-element updates the specified column value in the extended msg tracking data elements for the given column name and an additional col name/value pair. This updates accross mtrk_id values. The additional name/value pair should be some unique identifier that does not update non-related records.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
col_name1	string	Required. The message tracking extended column name used as a unique identifier.
col_value1	string	Required. The value used as a unique identifier.
col_name2	string	Required. The message tracking extended column name.
col_value2	string	Required. The value used to update column.

Return Values

Boolean

Returns **#t** (true)—if the element is successfully updated; otherwise returns **#f** (false)—if the element is not successfully updated. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

None.

Example

```
(define fb_unique_id "AAAAA11111")
(define upd_element (ux-upd-mtrk-element
                                connection-handle
                                "BATCH_UNIQUE_ID"
                                fb_unique_id
                                "BATCH_SEND_IMM"
                                "Y"
                        )
)
```

ux-upd-mtrk-ext-data

Syntax

```
(ux-upd-mtrk-ext-data connection-handle mtrk_id direction col_name
col_value)
```

Description

ux-upd-mtrk-element updates the specified column value in the extended msg tracking data elements for the given column name, direction and message tracking id.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
mtrk_id	string	Required. Either mtrk_outb_id, or mtrk_inb_id.
direction	string	Required. Indicates the direction of the message: I—Inbound (mtrk_id is mtrk_inb_id) O—Outbound (mtrk_id is mtrk_outb_id)
col_name	string	Required. The message tracking extended column name.
col_value	string	Required. The value used to update column.

Return Values

Boolean

Returns **#t** (true)—if the control number is successfully updated; otherwise returns **#f** (false)—if the control number is not successfully updated. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

None.

Example

```
(define upd-result   (ux-upd-mtrk-ext-data
                         connection-handle
                         "1"
                         "T_CONTROL_NUM"
                         "0004"
                     )
)
```

ux-wait-for-ack

Syntax

```
(ux-wait-for-ack connection-handle tp-loc resp_tm retry_max
                 [mtrk-outb-id])
```

Description

ux-wait-for-ack creates a row in the es_waiting_ack database table for the given Message Profile. The row contains information tied to the wrapped Message Profile already stored in es_mtrk_outb using mtrk_outb_id, if provided. Otherwise uses g_mtrk_id and provides information to the Ack Monitor about the acknowledgment expected.

Use the **ux-wait-for-ack** API for outbound Message Profiles.

Parameters

Name	Type	Description
connection-handle	connection-handle	Required. The previously established connection to the database.
tp-loc	string	Required. Indicates which global struct to query. Acceptable value: O— original struct
resp_tm	string	How long in seconds e*Xchange should wait for an acknowledgment. If resp_tm is NULL or 0, then a row is not put in es_waiting_ack.
retry_max	string	The maximum number of times to resend the data. If retry_max is NULL, then a 0 is put in es_waiting_ack for retry_max.
mtrk-outb-id	string	Optional. ID that corresponds to Message Profile in es_mtrk_outb. If this parameter is not provided, the system uses the g_mtrk_id, which corresponds to the row in es_mtrk_outb where the information for this acknowledgment was stored.

Return Values

Boolean

Returns **#t** (true)—if a row was successfully created in the es_waiting_ack table; otherwise returns **#f** (false)—if a row was not successfully created in the es_waiting_ack table. Use the **ux-get-error-str** API to retrieve the corresponding error message.

Throws

Exception-InvalidArg, Exception-Catastrophic (can't be caught).

Example

The following Monk script Example call **ux-wait-for-ack** with the assumption that **ux-init-trans** was executed successfully for the given Message Profile. **ux-wait-for-ack** first checks to see if an mtrk_outb_id has been provided. The first example has a value of "75" for mtrk_outb_id, which is used for the insertion into es_waiting_ack. If this ID already exists in es_mtrk_outb and there is not already a row containing this mtrk_outb_id in es_waiting_ack, then a row should be inserted into es_waiting_ack and a **#t** is returned. In this case, "Wait for Ack succeeded!" is displayed. If an error occurs, then **#f** is returned and the error string is printed by the display of **ux-get-error-str**. The second example does not provide an mtrk_outb_id, so g_mtrk_id is used. If g_mtrk_id is invalid or a row already exists in es_waiting_ack with that value in mtrk_outb_id, then the insertion fails and an error string is displayed. Otherwise, on success "Wait for Ack succeeded!" is displayed.

```
(define type "A")
(define mtrk-outb-id "75")
(if (ux-wait-for-ack connection-handle type "20" "" mtrk-outb-id)
      (display "Wait for Ack succeeded!\n")
      (begin
            (display "Wait for Ack failed!\n")
            (display (ux-get-error-str))
            (newline)
      )
 )


 (if (ux-wait-for-ack connection-handle)
      (display "Wait for Ack succeeded!\n")
      (begin
            (display "Wait for Ack failed!\n")
            (display (ux-get-error-str))
            (newline)
      )
 )
```

13.3 Monk Functions Used by the Validation Rules Builder

A set of monk functions has been provided for the Validation Rules Builder. These functions are used within the validation Collaborations created by the VRB. These Collaborations are used by e*Xchange to validate the EDI data it receives from e*Gate.

The validations are based on the implementation guidelines specified in the SEF file that is converted to e*Gate ETD and Collaboration files.

The VRB functions are described on the following pages:

compare-equal

Syntax

```
(compare=? string1 string2)
```

Description

compare-equal determines whether the two strings contained in the parameters are equal. If the string values are numeric, it converts the strings to numbers before making the comparison so that a valid numeric comparison is made.

Parameters

Name	Type	Description
string1	string	The first of the string values to be compared.
string2	string	The second of the string values to be compared.

Return Values

Boolean

Returns **#t** (true) if the two strings are equal; otherwise returns **#f** (false) if they are not equal.

Throws

None.

Example

```
(if (compare=? "A0B" "A1B")
   (display "A0B = A1B\n")
   (display "A0B != A1B\n")
)

=> A0B != A1B
```

compare-ge

Syntax

```
(compare>=? string1 string2)
```

Description

compare-ge determines whether *string1* is greater than or equal to *string2*. If the string values are numeric, it converts the strings to numbers before making the comparison so that a valid numeric comparison is made.

Parameters

Name	Type	Description
string1	string	The first of the string values to be compared.
string2	string	The second of the string values to be compared.

Return Values

Boolean

Returns **#t** (true) if *string1* is greater than or equal to *string2*; otherwise **#f** (false) if *string1* is less than *string2*.

Throws

None.

Example

```
(if (compare>=? "A3B" "A1B")
   (display "A3B >= A1B\n")
   (display "A3B < A1B\n")
)

=> A3B >= A1B
```

compare-gt

Syntax

```
(compare>? string1 string2)
```

Description

compare-gt determines whether *string1* is greater than *string2*. If the string values are numeric, it converts the strings to numbers before making the comparison so that a valid numeric comparison is made.

Parameters

Name	Type	Description
string1	string	The first of the string values to be compared.
string2	string	The second of the string values to be compared.

Return Values

Boolean

Returns **#t** (true) if *string1* is greater than *string2*; otherwise **#f** (false) if *string1* is less than or equal to *string2*.

Throws

None.

Example

```
(if (compare>? "A3B" "A1B")
   (display "A3B > A1B\n")
   (display "A3B <= A1B\n")
)

=> A3B > A1B
```

compare-le

Syntax

```
(compare<=? string1 string2)
```

Description

compare-le determines whether *string1* is less than or equal to *string2*. If the string values are numeric, it converts the strings to numbers before making the comparison so that a valid numeric comparison is made.

Parameters

Name	Type	Description
string1	string	The first of the string values to be compared.
string2	string	The second of the string values to be compared.

Return Values

Boolean

Returns **#t** (true) if *string1* is less than or equal to *string2*; otherwise **#f** (false) if *string1* is greater than *string2*.

Throws

None.

Example

```
(if (compare<=? "A3B" "A1B")
   (display "A3B <= A1B\n")
   (display "A3B > A1B\n")
)

=> A3B > A1B
```

compare-lt

Syntax

```
(compare<? string1 string2)
```

Description

compare-lt determines whether *string1* is less than *string2*. If the string values are numeric, it converts the strings to numbers before making the comparison so that a valid numeric comparison is made.

Parameters

Name	Type	Description
string1	string	The first of the string values to be compared.
string2	string	The second of the string values to be compared.

Return Values

Boolean

Returns **#t** (true) if *string1* is less than *string2*; **#f** (false) if *string1* is greater than or equal to *string2*.

Throws

None.

Example

```
(if (compare<? "A3B" "A1B")
   (display "A3B < A1B\n")
   (display "A3B >= A1B\n")
)

=> A3B >= A1B
```

string-alpha

Syntax

```
(string-alpha? string)
```

Description

string-alpha determines whether the string parameter contains only alphabetic characters.

Parameters

Name	Type	Description
string	string	The string to be evaluated.

Return Values

Boolean

Returns **#t** (true) if string contains only alphabetic characters; otherwise **#f** (false) if string contains at least one character that is not alphabetic.

Throws

None.

Example

```
(if (string-alpha? "AbC")
   (display "AbC is alphabetic\n")
   (display "AbC is NOT alphabetic\n")
)

=> AbC is alphabetic
```

string-alphanumeric

Syntax

```
(string-alphanumeric? string)
```

Description

string-alphanumeric determines whether string contains only alphabetic and/or numeric characters.

Parameters

Name	Type	Description
string	string	The string to be evaluated.

Return Values

Boolean

Returns **#t** (true) if the string contains only alphabetic and/or numeric characters; otherwise **#f** (false) if the string contains at least one character that is not alphabetic or numeric.

Throws

None.

Example

```
(if (string-alphanumeric? "AbC")
(display "AbC is alphanumeric\n")
(display "AbC is NOT alphanumeric\n")
)

=> AbC is alphanumeric
```

string-numeric

Syntax

```
(string-numeric? string)
```

Description

string-numeric determines whether the string parameter contains only numeric characters.

Parameters

Name	Type	Description
string	string	The string to be evaluated.

Return Values

Boolean

Returns **#t** (true) if the string contains only numeric characters; otherwise **#f** (false) if the string contains at least one character that is not numeric.

Throws

None.

Example

```
(if (string-numeric? "145a3")
   (display "145a3 is numeric\n")
   (display "145a3 is NOT numeric\n")
)

=> 145a3 is NOT numeric
```

valid-date-yyyy

Syntax

```
(valid-date-yyyy? YYYYMMDD or YYMMDD)
```

Description

valid-date-yyyy determines whether the date value YYYYMMDD or YYMMDD is a valid date.

Parameters

Name	Type	Description
YYYYMMDD or YYMMDD	string	Date is composed of year, month and day: ▪ YYYY—4-digit year; for example, 2000 ▪ YY—2-digit year; for example, 99 for 1999 ▪ MM—2-digit month; for example, 05 for May ▪ DD—2-digit day; for example, 03 or 29

Return Values

Boolean

Returns **#t** (true) if the string is a valid date; otherwise **#f** (false) if it is not a valid date.

Throws

None.

Example

```
(if (valid-date-yyyy? "20000229")
   (display "20000229 is a valid date\n")
   (display "20000229 is NOT a valid date\n")
)

=> 20000229 is a valid date
```

valid-time

Syntax

```
(valid-time? timestamp)
```

Description

valid-time determines whether the timestamp is a valid time.

Parameters

Name	Type	Description
timestamp	string	Date and time stamp, in one of the following formats: ▪ HHMM ▪ HHMMSS ▪ HHMMSSD ▪ HHMMSSDD where the time stamp is composed of the following values: ▪ HH = 00—23 (hours) ▪ MM = 00—59 (minutes) ▪ SS = 00—59 (seconds) ▪ D = 0—9 (sub-second single digit) ▪ DD = 00—99 (sub-second double digit)

Return Values

Boolean

Returns **#t** (true) if the timestamp is a valid time; otherwise **#f** (false) if it is not a valid time.

Throws

None.

Example

```
(if (valid-time? "0000117a")
   (display "\n0000117a is a valid time\n")
   (display "\n0000117a is NOT a valid time\n")
)

=> 0000117a is NOT a valid time
```

13.4 e*Xchange MIME Functions

These functions use MIMEsimple.ssc, a simple message structure, to parse and compose MIME messages.

The following table lists the MIME functions and their locations in this document.

util-mime-get-header-value

Syntax

```
(util-mime-get-header-value node-path mime_field_name)
```

Description

util-mime-get-header-value retrieves the value of the specified field in a mime message pointed to by the given path.

Parameters

Name	Type	Description
node-path	path	The node path to the MIME structure or substructure.
mime_field_name	string	The name of the field in the MIME header.

Return Values

string

Returns the value of the specified field if one exists; otherwise, returns a null string if the header doesn't exist or a failure occurred.

Throws

None.

Example

If ~input%MIMEsimple is mapped to the following MIME component:

Content-Type: multipart/related;

boundary="RN-boundary";

Content-Description: This is the content description;

This is the message body...

then,

(util-mime-get-header-value ~input%MIMEsimple "Content-Type")

```
=> "multipart/related\r\n boundary=\"RN-boundary\""
```

util-mime-get-par-value

Syntax

```
(util-mime-get-par-value <node-path> mime_field_name mime_par_name)
```

Description

util-mime-get-par-value retrieves the value of the specified parameter in the specified field in a mime message pointed to by the given path.

Parameters

Name	Type	Description
node-path	path	The node path to the MIME structure or substructure.
mime_field_name	string	The name of the field in the MIME header.
mime_par_name	string	The name of the parameter under the MIME field.

Return Values

string

Returns the value of the specified parameter if it exists; otherwise, returns a null string if the header or parameter doesn't exist or failure occurred.

Throws

None.

Example

If ~input%MIMEsimple is mapped to the following MIME component:

Content-Type: multipart/related;

boundary="RN-boundary";

Content-Description: This is the content description;

This is the message body...

then,

(util-mime-get-par-value ~input%MIMEsimple "Content-Type" "boundary")

```
=> "RN-boundary"
```

util-mime-make-mime-message

Syntax

```
(util-mime-make-mime-message <node-path>)
```

Description

util-mime-make-mime-message composes and returns a MIME message string from the specified node.

Note that ($event->string) does not work properly due to the way the MIMEsimple structure is composed. Instead, use (util-mime-make-mime-message) to compose a MIME message from a node.

Parameters

Name	Type	Description
node-path	path	The node path to the MIME structure or substructure

Return Values

string

Returns a MIME message string.

Throws

None.

Example

If ~input%root.MIMEsimple is mapped to the following MIME component:

Content-Type: multipart/related;
boundary="RN-boundary";
Content-Description: This is the content description;

This is the message body...

then,

(util-mime-make-mime-message ~input%root.MIMEsimple)

returns the original message string:

Content-Type: multipart/related;
boundary="RN-boundary";
Content-Description: This is the content description;

This is the message body...

util-mime-map-event

Syntax

```
(util-mime-map-event mime_event_map mime_message_string)
```

Description

util-mime-map-event populates the given mime_event_map with the given mime message string.

Parameters

Name	Type	Description
mime_event_map	path	The node path to the MIME structure or substructure
mime_message_string	string	The MIME message string

Return Values

Undefined.

Throws

eXception-Mapping.

Example

If ~input%root.MIMEsimple is a MIME structure and mime-string is the following string:

Content-Type: multipart/related;

boundary="RN-boundary";

Content-Description: This is the content description;

This is the message body...

then,

(util-mime-map-event ~input%root.MIMEsimple mime-string) parses the message string "mime-string" with the MIME structure "~input%root.MIMEsimple".

util-mime-pack-encrypted-msg

Syntax

```
(util-mime-pack-encrypted-msg filename base64_pkcs7_msg)
```

Description

util-mime-pack-encrypted-msg composes and returns an encrypted MIME message.

Parameters

Name	Type	Description
filename	value	The value of this parameter is ignored by the function.
base64_pkcs7_msg	string	The base64 encoded encrypted message

Return Values

string

Returns the encrypted message in MIME format.

Throws

None.

Example

If base64_pkcs7_msg is a base64 encoded encrypted message, then

```
(util-mime-pack-encrypted-msg "" base64_pkcs7_msg)
=> the encrypted message (base64_pkcs7_msg) in MIME format.
```

util-mime-pack-signed-msg

Syntax

```
(util-mime-pack-signed-msg protocol micalg content signature)
```

Description

util-mime-pack-signed-msg composes and returns a signed MIME message.

Parameters

Name	Type	Description
protocol	value	The value assigned to the 'protocol' parameter Content-Type field of the MIME message.
miclag	value	The value assigned to the 'micalg' parameter Content-Type field of the MIME message.
content	string	The message string.
signature	string	The result of signing the content with the "micalg" algorithm.

Return Values

string

Returns the signed message in MIME format.

Throws

None.

Example

For "content" and "signature" are the content and signature of a message string, respectively, then

```
(util-mime-pack-signed-msg "application/pkcs7-signature" "sha1"
content signature)

=> the message string with its signature in MIME format, where the
protocol field in the MIME header is set to "application/pkcs7-
signature" and the micalg field in the MIME header is set to "sha1".
```

util-mime-unpack-signed-message

Syntax

```
(util-mime-unpack-signed-message mime_message_string)
```

Description

util-mime-unpack-signed-message unpacks the given message string and returns a vector of strings: (protocol micalg content signature). **protocol** and **micalg** are the values of the protocol and micalg parameters in the Content-Type field of the input message, respectively. **content** and **signature** are the content and signature of the signed message, respectively.

Parameters

Name	Type	Description
mime_message_string	string	The MIME message string.

Return Values

Boolean

Returns **#t** (true) if the input message is not signed; otherwise, returns **#f** (false) if the input message is signed but the function fails to get signature, content, micalg, or protocol.

Throws

Exception-Mapping and Exception-Generic.

Example

If msg is a MIME message string as follows:

Content-Type: multipart/signed;

boundary="RN-sign";

protocol="application/pkcs7-signature";

micalg=sha1

--RN-sign

this is the content...

--RN-sign

this is the MIME header for the signature component

this is the signature...

--RN-sign--

then,

```
(util-mime-unpack-signed-message msg)

=> #("application/pkcs7-signature", "sha1" "this is the content...",
"this is the signature...")
```

13.5 e*Xchange RosettaNet 2.0 Functions

The following RosettaNet 2.0 functions are available.

These functions use ROS20Generic.ssc, a message structure for the RNGM. It maps the preamble, delivery header, and service header parts of the RNBM. The service content of the RNBM is mapped to an end node. Any attachments of an RNBM are mapped to a repetitive node that uses the MIMEsimple structure as a template.

The following table lists the RosettaNet functions and their locations in this document.

eX-ROS20-Generic-To-String

Syntax

```
(eX-ROS20-Generic-To-String input)
```

Description

eX-ROS20-Generic-To-String retrieves an RNGM message string from the input ROS20Generic event map. This function should be used instead of ($event->string) when converting an RNGM event map to a Monk string.

Parameters

Name	Type	Description
input	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns an RNGM message string.

Throws

Exception-Mapping and Exception-Generic.

Example

If input is a event map that represents a RNGM message, then

```
(eX-ROS20-Generic-To-String input)
=> Monk string that represents the same RNGM message.
```

eX-ROS20-Parse-Generic

Syntax

```
(eX-RSO20-Parse-Generic input vector_of_strings)
```

Description

eX-RSO20-Parse-Generic parses the various string components in the specified vector using the given RNGM event map.

Parameters

Name	Type	Description
input	event	The variable name of the event structure that contains the ROS20 Generic message.
vector_of_strings	string	The vector elements: preamble, delivery header, service header, service content, attachments (repeating)

Return Values

Boolean

Returns **#t** (true) on success.

Throws

Exception-Mapping and Exception-Generic.

Example

If **input** is a RNGM event map and **vec** is a vector containing the preamble, delivery header, service header, service content, and attachments of a RNBM, then:

```
(eX-ROS20-Parse-Generic input vec)

returns #t and after the call "input" contains the RNGM message.
```

eX-ROS20-Pack-RNBM

Syntax

```
(eX-ROS20-Pack-RNBM db_connection_handle input_rngm encryption_flag
sec_keys tpic_id)
```

Description

eX-ROS20-Pack-RNBM composes and returns an RNBM from the given RNGM event map (input).

Parameters

Name	Type	Description
db_connection_handle	database connection handle	A connection handle to the database that contains the security information.
input_rngm	event	The variable name of the event structure that contains the ROS20 Generic message.
encryption_flag	string	The encryption required. The following values are available: ▪ 0 if no encryption required. ▪ 1 if only service content and. attachments need to be encrypted ▪ 2 if service header, service content and attachments need to be encrypted.
sec_keys	string	The security keys to be used when signing/encrypting the message.
tpic_id	string	The index to the trading partner profile.

Return Values

string

Returns the RNBM.

Throws

Exception-Mapping and Exception-Generic.

Example

If input_rngm is an event map containing a RNGM, then,

```
(eX-ROS20-Pack-RNBM db_connection_handle input_rngm encryption_flag
sec_keys tpic_id)

returns the corresponding RNBM, in which the message is encrypted and/
or signed with the security information in the database as specified
by tpic_id, sec_keys, and encryption_flag.
```

eX-ROS20-Unpack-RNBM

Syntax

```
(eX-ROS20-Unpack-RNBM db_connection_handle message_string
security_keys tpic_id)
```

Description

eX-ROS20-Unpack-RNBM parses an RNBM and returns a vector of strings (preamble, delivery header, service header, service content, attachments, and so on.)

Parameters

Name	Type	Description
db_connection_handle	database connection handle	The connection handle to the database that contains the security information.
message_string	string	The input RNBM
security_keys	string	The security keys to be used when signing/encrypting the message.
tpic_id	string	The index to the trading partner profile.

Return Values

vector

Returns a vector of strings (preamble, delivery header, service header, service content, attachments, and so on).

On failure, the global variable error_data is appended to include the failure reason.

Elements of the global variable vector 'g_output' are set as the various body parts (preamble, deliver header, service header, and service content) and are unpacked.

Throws

Exception-Mapping and Exception-Generic.

Example

If message_string is a Monk string representing a RNBM, and security_keys and tpic_id specifies the correction security information in the database, then

```
(eX-ROS20-Unpack-RNBM db_connection_handle message_string
security_keys tpic_id)

returns a vector containing the preamble, delivery header, service
header, service content, and attachments (if any) of the RosettaNet
Business Message (RNBM).
```

eX-ROS20-Validate-Preamble

Syntax

```
(eX-ROS20-Validate-Preamble)
```

Description

eX-ROS20-Validate-Preamble validates the preamble. It uses parameters passed in to a global Monk variable g_input, which is of type vector. The elements of g_input are described in the following table in the order they appear.

Global Parameters used in g_input

Name	Type	Description
input_rngm	event	The variable name of the event structure that contains the ROS20 Generic message.
prf_attrib	string	The vector of partner profile attributes as returned by the (ux-get-header) Monk function.

Parameters

None.

Return Values

Boolean

Returns **#t** (true) if preamble is valid; otherwise, returns **#f** (false). Also error_data is appended to reflect the error, if any.

Throws

Exception-Mapping and Exception-Generic.

Example

(set! input_rngm (eX-ROS20-Parse-RNGM...)) ;; parse the input RNGM

(set! prf_attrib (ux-get-header "A" transaction_info)) ;; set the partner profile attributes

...

(set! g_input (vector input_rngm prf_attrib)) ;; set the global variable g_input

(eX-ROS20-Validate-Preamble) => #t/#f depending on whether the preamble contained in input_rngm is valid.

eX-ROS20-Validate-ServiceHeader

Syntax

```
(eX-ROS20-Validate-ServiceHeader)
```

Description

eX-ROS20-Validate-ServiceHeader validates the service header. It uses parameters passed in to a global Monk variable g_input, which is of type vector. The elements of g_input are described in the following table in the order they appear.

Global Parameters used in g_input

Name	Type	Description
input_rngm	event	The variable name of the event structure that contains the ROS20 Generic message.
prf_attrib	string	The vector of partner profile attributes as returned by the (ux-get-header) Monk function.

Parameters

None.

Return Values

Boolean

Returns **#t** (true) if service header is valid; otherwise, returns **#f** (false). Also error_data is appended to reflect the error, if any.

Throws

Exception-Mapping and Exception-Generic.

Example

(set! input_rngm (eX-ROS20-Parse-RNGM...)) ;; parse the input RNGM

(set! prf_attrib (ux-get-header "A" transaction_info)) ;; set the partner profile attributes

...

(set! g_input (vector input_rngm prf_attrib)) ;; set the global variable g_input

(eX-ROS20-Validate-ServiceHeader) => #t/#f depending on whether the service header contained in input_rngm is valid.

eX-ROS20-Validate-DeliveryHeader

Syntax

```
(eX-ROS20-Validate-DeliveryHeader)
```

Description

eX-ROS20-Validate-DeliveryHeader validates the delivery header.It uses parameters passed in to a global Monk variable g_input, which is of type vector. The elements of g_input are described in the following table in the order they appear.

Global Parameters used in g_input

Name	Type	Description
input_rngm	event	The variable name of the event structure that contains the ROS20 Generic message.
prf_attrib	string	The vector of partner profile attributes as returned by the (ux-get-header) Monk function.

Parameters

None.

Return Values

Boolean

Returns **#t** (true) if delivery header is valid; otherwise, returns **#f** (false). Also error_data is appended to reflect the error, if any.

Throws

Exception-Mapping and Exception-Generic.

Example

(set! input_rngm (eX-ROS20-Parse-RNGM...)) ;; parse the input RNGM

(set! prf_attrib (ux-get-header "A" transaction_info)) ;; set the partner profile attributes

...

(set! g_input (vector input_rngm prf_attrib)) ;; set the global variable g_input

(eX-ROS20-Validate-DeliveryHeader) => #t/#f depending on whether the delivery header contained in input_rngm is valid.

eX-ROS20-Populate-Preamble

Syntax

```
(eX-ROS20-Populate-Preamble)
```

Description

eX-ROS20-Populate-Preamble populates the preamble header with the extended attributes from the database. It uses parameters passed in a global Monk variable g_input, which is of type vector. The elements of g_input are described in the following table in the order they appear.

Global Parameters used in g_input

Name	Type	Description
input_rngm	event	The variable name of the event structure that contains the ROS20 Generic message. The preamble part of this event map is partially filled before the call to eX-ROS20-Populate-Preamble and is fully populated after the call.
prf_attrib	string	A vector of partner profile attributes as returned by the (ux-get-header) Monk function.

Parameters

None.

Return Values

Boolean

Returns #t (true) on success; otherwise, returns #f (false).

Throws

Exception-Mapping and Exception-Generic.

Example

(set! input_rngm (eX-ROS20-Parse-RNGM...)) ;; parse the partial input RNGM

(set! prf_attrib (ux-get-header "A" transaction_info)) ;; set the partner profile attributes

...

(set! g_input (vector input_rngm prf_attrib)) ;; set the global variable g_input

(eX-ROS20-Populate-Preamble)

eX-ROS20-Populate-ServiceHeader

Syntax

```
(eX-ROS20-Populate-ServiceHeader)
```

Description

eX-ROS20-Populate-ServiceHeader populates the service header with the extended attributes. It uses parameters passed in a global Monk variable g_input, which is of type vector. The elements of g_input are described in the following table in the order they appear.

Global Parameters used in g_input

Name	Type	Description
input_rngm	event	The variable name of the event structure that contains the ROS20 Generic message. The preamble part of this event map is partially filled before the call to eX-ROS20-Populate-Preamble and is fully populated after the call.
prf_attrib	string	A vector of partner profile attributes as returned by the (ux-get-header) Monk function.

Parameters

None.

Return Values

Boolean

Returns #t (true) on success; otherwise, returns #f (false).

Throws

Exception-Mapping and Exception-Generic.

Example

(set! input_rngm (eX-ROS20-Parse-RNGM...)) ;; parse the partial input RNGM

(set! prf_attrib (ux-get-header "A" transaction_info)) ;; set the partner profile attributes

...

(set! g_input (vector input_rngm prf_attrib)) ;; set the global variable g_input

(eX-ROS20-Populate-ServiceHeader)

eX-ROS20-Populate-DeliveryHeader

Syntax

```
(eX-ROS20-Populate-DeliveryHeader)
```

Description

eX-ROS20-Populate-DeliveryHeader populates the delivery header with the extended attributes.It uses parameters passed in a global Monk variable g_input, which is of type vector. The elements of g_input are described in the following table in the order they appear.

Global Parameters used in g_input

Name	Type	Description
input_rngm	event	The variable name of the event structure that contains the ROS20 Generic message. The preamble part of this event map is partially filled before the call to eX-ROS20-Populate-Preamble and is fully populated after the call.
prf_attrib	string	A vector of partner profile attributes as returned by the (ux-get-header) Monk function.

Parameters

None.

Return Values

Boolean

Returns #t (true) on success; otherwise, returns #f (false).

Throws

Exception-Mapping and Exception-Generic.

Example

(set! input_rngm (eX-ROS20-Parse-RNGM...)) ;; parse the partial input RNGM

(set! prf_attrib (ux-get-header "A" transaction_info)) ;; set the partner profile attributes

...

(set! g_input (vector input_rngm prf_attrib)) ;; set the global variable g_input

(eX-ROS20-Populate-DeliveryHeader)

eX-ROS20-Unique-ID

Syntax

```
(eX-ROS20-Unique-ID rngm)
```

Description

eX-ROS20-Unique-ID retrieves the unique id for this ROS20 message.

Parameters

Name	Type	Description
rngm	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns the unique id for the message if this is a valid response message; otherwise, returns a null string if this message is not a response message or the input rngm does not contain enough information to compose a request ID.

Throws

Exception-Mapping and Exception-Generic.

Example

If rngm contains the following fields:

Initiating_partner_ID = "1234"

PIP_code = "3A4"

PIP_Instance_ID = "567"

Activity_ID = "Create Order"

signal_code = "Order Request Action"

```
(eX-ROS20-Unique-ID rngm)
=> "1234|3A4|567|Create Order|OrderRequest Action"
```

eX-ROS20-Request-ID

Syntax

```
(eX-ROS20-Request-ID RNGM)
```

Description

eX-ROS20-Request-ID retrieves the unique id of the original request message if this message is a response.

Parameters

Name	Type	Description
RNGM	event	The RNGM

Return Values

string

Returns the unique id of the original request message if this message is a valid response; otherwise, returns a null string if this message is not a response message or input rngm doesn't contain enough information to compose a request ID.

Throws

Exception-Mapping and Exception-Generic.

Example

If rngm contains the following fields:

Initiating_partner_ID = "1234"

PIP_code = "3A4"

PIP_Instance_ID = "567"

Activity_ID = "Create Order"

signal_code = "Order Request Action"

```
(eX-ROS20-Request-ID rngm)
=> "1234|3A4|567|Create Order|OrderRequest Action"
```

eX-ROS20-Ack-Type

Syntax

```
(eX-ROS20-Ack-Type rngm)
```

Description

eX-ROS20-Ack-Type returns "P" if this message is a positive response; "N" if negative response; "" if not a response.

Parameters

Name	Type	Description
rngm	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns one of the following values:

"P"	if this message is a positive response
"N"	if this message is a negative response
null string	if this message is not a valid response

Throws

Exception-Mapping and Exception-Generic.

Example

If rngm contains a Receipt acknowledgement exception, then

```
(eX-ROS20-Ack-Type rngm) => "N"
```

eX-ROS20-IsResponse?

Syntax

```
(eX-ROS20-IsResponse? rngm)
```

Description

eX-ROS20-IsResponse? checks whether a message is a response message.

Parameters

Name	Type	Description
rngm	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

Boolean

Returns **#t** (true) if this message is a response message; otherwise, returns **#f** (false).

Throws

Exception-Mapping and Exception-Generic.

Example

if rngm contains a 3A4 request message,

```
(eX-ROS20-IsResposne? rngm)=> #f
```

eX-ROS20-IsSignal?

Syntax

```
(eX-ROS20-IsSignal? rngm)
```

Description

eX-ROS20-IsSignal? checks whether the message is a business signal.

Parameters

Name	Type	Description
rngm	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

Boolean

Returns **#t** (true) if this message is a business signal; otherwise, returns **#f** (false).

Throws

Exception-Mapping and Exception-Generic.

Example

if rngm is a 3A4 request message, then

(eX-ROS20-IsSignal? rngm) => #f

if rngm is a Receipt Acknowledgement signal message, then

(eX-ROS20-IsSignal? rngm) => #t

eX-ROS20-Get-PipCode

Syntax

```
(eX-ROS20-Get-PipCode RNGM)
```

Description

eX-ROS20-Get-PipCode returns the PIP code in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

Returns the PIP code in the rngm.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Get-PipCode rngm)

=> "3A4"

eX-ROS20-Set-PipCode

Syntax

```
(eX-ROS20-Set-PipCode RNGM value)
```

Description

eX-ROS20-Set-PipCode sets the PIP code in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.
value	string	The Pip Code.

Return Values

Undefined.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Set-PIPCode rngm "3A4")

=> undefined

eX-ROS20-Get-SigActCode

Syntax

```
(eX-ROS20-Get-SigActCode RNGM)
```

Description

eX-ROS20-Get-SigActCode returns the signal code for a business signal or the action code for an action message.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns the signal code for a business signal or the action code for an action message; otherwise, returns a null string if this message is not a response message or input rngm doesn't contain enough information to compose a request ID.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Get-SigAckCode rngm)

=> "Purchase Order Request Action"

eX-ROS20-Set-SigActCode

Syntax

```
(eX-ROS20-Set-SigActCode RNGM value)
```

Description

eX-ROS20-Set-SigActCode sets the signal code for a business signal or the action code for an action message.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.
value	string	The signal or action code.

Return Values

Undefined.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Set-SigAckCode rngm "Purchase Order Request Action")

=> undefined

eX-ROS20-Get-SigActVerId

Syntax

```
(eX-ROS20-Get-SigActVerId RNGM)
```

Description

eX-ROS20-Get-SigActVerId retrieves the signal version ID for an business signal or the action version ID for an action message.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns the signal version ID for an business signal or the action version ID for an action message.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Get-SigAckVerID rngm)

=> "2.0"

eX-ROS20-Set-SigActVerId

Syntax

```
(eX-ROS20-Set-SigActVerId RNGM value)
```

Description

eX-ROS20-Set-SigActVerId sets the signal version ID for an business signal or the action version ID for an action message.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.
value	string	The signal or action version ID.

Return Values

Undefined.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Set-SigAckVerId rngm "2.0")

=> undefined

eX-ROS20-Get-PipVerId

Syntax

```
(eX-ROS20-Get-PipVerId RNGM)
```

Description

eX-ROS20-Get-PipVerId retrieves the PIPVersion.VersionIdentifier in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns the PIPVersion.VersionIdentifier in the rngm.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Get-PipVerId rngm)

=> "2.0"

eX-ROS20-Set-PipVerId

Syntax

```
(eX-ROS20-Set-PipVerId RNGM value)
```

Description

eX-ROS20-Set-PipVerId sets the PIPVersion.VersionIdentifier in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.
value	string	The PIP Version.VersionIdentifier.

Return Values

Undefined.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Set-PipVerId rngm "2.0")

=> undefined

eX-ROS20-Get-PipId

Syntax

```
(eX-ROS20-Get-PipId RNGM)
```

Description

eX-ROS20-Get-PipId retrieves the PIPInstanceId.InstanceIdentifier in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns the PIPInstanceId.InstanceIdentifier in the rngm.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Get-PipId rngm)

=> "12345"

eX-ROS20-Set-PipId

Syntax

```
(eX-ROS20-Set-PipId RNGM value)
```

Description

eX-ROS20-Set-PipId sets the PIPInstanceId.InstanceIdentifier in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.
value	string	The PIPInstanceId.InstanceIdentifier.

Return Values

Undefined.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Set-PipId rngm "12345")

=> undefined

eX-ROS20-Get-ActId

Syntax

```
(eX-ROS20-Get-ActId RNGM)
```

Description

eX-ROS20-Get-ActId returns the activity ID in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns the activity ID in the rngm.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Get-ActId rngm)

=> "Purchase Order Request"

eX-ROS20-Set-ActId

Syntax

```
(eX-ROS20-Set-ActId RNGM value)
```

Description

eX-ROS20-Set-ActId sets the activity ID in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.
value	string	The activity ID.

Return Values

Undefined.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Set-AckId rngm "Purchase Order Request")

=> undefined

eX-ROS20-Get-InReplyTo-MsgId

Syntax

```
(eX-ROS20-Get-InReplyTo-MsgId RNGM)
```

Description

eX-ROS20-Get-InReplyTo-MsgId retrieves InReplyTo.MessageInstanceID.InstanceIdentifier in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns InReplyTo.MessageInstanceID.InstanceIdentifier in the rngm.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Get-InReplyTo-MsgId rngm)

=> "1234"

eX-ROS20-Set-InReplyTo-MsgId

Syntax

```
(eX-ROS20-Set-InReplyTo-MsgId RNGM value)
```

Description

eX-ROS20-Set-InReplyTo-MsgId sets InReplyTo.MessageInstanceID.InstanceIdentifier in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.
value	string	The InReplyTo.MessageInstanceID.InstanceIdentifier.

Return Values

Undefined.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Set-InReplyTo-MsgId rngm "1234")

=> undefined

eX-ROS20-Get-InReplyTo-ActCode

Syntax

```
(eX-ROS20-Get-InReplyTo-ActCode RNGM)
```

Description

eX-ROS20-Get-InReplyTo-ActCode returns InReplyTo.ActionIdentity.GlobalBusinessActionCode in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns InReplyTo.ActionIdentity.GlobalBusinessActionCode in the rngm.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Get-InReplyTo-ActCode rngm)

=> "Purchase Order Request Action"

eX-ROS20-Set-InReplyTo-ActCode

Syntax

```
(eX-ROS20-Set-InReplyTo-ActCode RNGM value)
```

Description

eX-ROS20-Set-InReplyTo-ActCode sets InReplyTo.ActionIdentity.GlobalBusinessActionCode in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.
value	string	The InReplyTo.ActionIdentity.GlobalBusinessActionCode.

Return Values

Undefined.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Set-InReplyTo-ActCode rngm "Purchase Order Request Action")

=> undefined

eX-ROS20-Get-InitPartnerId

Syntax

```
(eX-ROS20-Get-InitPartnerId RNGM)
```

Description

eX-ROS20-Get-InitPartnerId returns the Initiating Partner Global Business Identifier in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.

Return Values

string

Returns the Initiating Partner Global Business Identifier in the rngm.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Get-InitPartnerId rngm)

=> "1234567"

eX-ROS20-Set-InitPartnerId

Syntax

```
(eX-ROS20-Set-InitPartnerId RNGM value)
```

Description

eX-ROS20-Set-InitPartnerId sets the Initiating Partner Global Business Identifier in the rngm.

Parameters

Name	Type	Description
RNGM	event	The variable name of the event structure that contains the ROS20 Generic message.
value	string	The Initiating Partner Global Business Identifier.

Return Values

Undefined.

Throws

Exception-Mapping and Exception-Generic.

Example

(eX-ROS20-Set-InitPartnerId rngm "1234567")

=> undefined

eX-ROS20-Create-0A1Notification

Syntax

```
(eX-ROS20-Create-0A1Notification)
```

Description

eX-ROS20-Create-0A1Notification creates an 0A1 Notification message and populates the reply RNGM. It uses parameters passed in to a global Monk variable g_input, which is of type vector. The elements of g_input are described in the following table in the order they appear.

Global Parameters used in g_input

Name	Type	Description
prf_attrib	vector	The trading partner profile attributes (Message Profile) as returned by the Monk function (ux-get-header).
reply_rngm	event	The variable name of the event structure that contains the 0A1 Notification message on return.
error_detail	string	The error text.
unique_id	string	The unique id.

Parameters

None.

Return Values

string

Returns a null string.

Throws

Exception-Mapping and Exception-Generic.

Example

If prf_attrib contains the Message Profile attributes associated with an action message, unique_id is the unique ID of the same action message, error_text contains a text description of the reason for creating the 0A1 notification, and reply_rngm is an event map to the RNGM structure, then after the following statements

(set! g_input (vector prf_attrib, reply_rngm, error_text, unique_id)

(eX-ROS20-Create-0A1Notification)

"reply_rngm" contains the 0A1 notification message created in response to the original action message identified by unique_id.

eX-ROS20-Create-ReceiptAck

Syntax

```
(eX-ROS20-Create-ReceiptAck)
```

Description

eX-ROS20-Create-ReceiptAck creates a Receipt Acknowledgement message and populates the reply RNGM, based on the original RNGM. It uses parameters passed in to a global Monk variable g_input, which is of type vector. The elements of g_input are described in the following table in the order they appear.

Global Parameters used in g_input

Name	Type	Description
original_rngm	event	The original ROS20 message in RNGM format.
original_rnbm	string	The original ROS20 message in RNBM format.
reply_rngm	event	The newly created Receipt Acknowledgement message in RNGM format.

Parameters

None.

Return Values

string

Returns a null string.

Throws

Exception-Mapping and Exception-Generic.

Example

If original_rngm is the an event map containing the original ROS20 message, original_rnbm is a Monk string containing the same original ROS20 message, and reply_rngm is an empty event mapped to the RNGM structure, then, after the following statements:

(set! g_input (vector original_rngm original_rnbm reply_rngm))

(eX-ROS20-Create-ReceiptAck)

reply_rngm contains the newly created Receipt Acknowledgement in response to the original ROS20 message.

eX-ROS20-Create-Except

Syntax

```
(eX-ROS20-Create-Except)
```

Description

eX-ROS20-Create-Except creates an Exception message and populates the reply RNGM, based on the original RNGM. It uses parameters passed in to a global Monk variable g_input, which is of type vector. The elements of g_input are described in the following table in the order they appear.

Global Parameters used in g_input

Name	Type	Description
original_rngm	event	The original ROS20 message in RNGM format.
reply_rngm	event	The newly created Receipt Acknowledgement message in RNGM format.

Name	Type	Description
error_code	string	The error code. Possible values are: ▪ UNP.MESG.SIGNERR: Error during unpackaging – Verifying the signature of the RosettaNet Business Message ▪ UNP.PRMB.READERR: Error during unpackaging – Reading the Preamble ▪ UNP.PRMB.VALERR: Error during unpackaging – Validating the Preamble ▪ UNP.DHDR.READERR: Error during unpackaging – Reading the Delivery Header ▪ UNP.DHDR.VALERR: Error during unpackaging – Validating the Delivery Header ▪ UNP.SHDR.READERR: Error during unpackaging – Reading the Service Header ▪ UNP.SHDR.VALERR: Error during unpackaging – Validating the Service Header ▪ UNP.SHDR.MNFSTERR: Error during unpackaging – Verifying Manifest against the actual attachment body parts ▪ UNP.MESG.SEQERR: Error during unpackaging – Validating the message sequence ▪ UNP.MESG.RESPTYPERR: Unexpected Response type in the HTTP header ▪ UNP.MESG.DCRYPTERR: Error Decrypting the message ▪ UNP.SCON.READERR: Error during unpackaging – Reading the Service Content ▪ UNP.SCON.VALERR: Error during unpackaging – Validating the Service Content ▪ PKG.MESG.GENERR: Error during packaging – General error ▪ PRF.ACTN.GENERR: Error during action performance – General Error ▪ PRF.DICT.VALERR: Error during action performance – Validating the Service Content against a PIP-specified dictionary ▪ UNP.MESG.GENERR: Error during unpackaging – General error

Name	Type	Description
error_detail	string	The description of the error.
error_component	string	The message component where the error occurred. Possible values include: ▪ Preamble ▪ DeliveryHeader ▪ ServiceHeader ▪ ServiceContent ▪ Attachment
except_type	string	The type of exception to be created. Possible values include General Exception and Receipt Acknowledgement exception.

Parameters

None.

Return Values

string

Returns a null string.

Throws

Exception-Mapping and Exception-Generic.

Example

If original_rngm is the an event map containing the original ROS20 message, and reply_rngm is an empty event mapped to the RNGM structure, then, after the following statements:

(set! g_input (vector original_rngm reply_rngm "UNP.MESG.SIGNERR" "sign error" "ServiceContent" "Receipt Acknowledgement Exception"))

(eX-ROS20-Create-Exception)

reply_rngm contains the newly created Receipt Acknowledgement Exception in response to the orignal ROS20 message.

13.6 e*Xchange Security Functions

This section provides detailed information on e*Xchange Security functions, divided into two groups. The first group contains generic functions that can be used with any data format. It includes the following functions:

- util-security-decrypt-msg.monk
- util-security-encrypt-msg.monk
- util-security-sign-msg.monk
- util-security-verify-sig.monk

The second group is specifically for working with the e*Xchange APIs and database. It includes the following functions:

- eX-security-get-keys-certs.monk
- eX-ROS20-decrypt-msg.monk
- eX-ROS20-encrypt-msg.monk
- eX-ROS20-sign-msg.monk
- eX-ROS20-verify-sig.monk
- eX-ROS20-get-ssl-keys.monk

13.6.1 Operational Groups

The e*Xchange security functions can also be divided into two operational groups. The first group contains the following functions:

- util-security-decrypt-msg.monk
- util-security-encrypt-msg.monk
- util-security-sign-msg.monk
- util-security-verify-sig.monk
- eX-security-get-keys-certs.monk
- eX-ROS20-decrypt-msg.monk
- eX-ROS20-encrypt-msg.monk
- eX-ROS20-sign-msg.monk
- eX-ROS20-verify-sig.monk

These functions are used within e*Xchange RosettaNet 2.0 scripts, as described below.

A Initialization – (eX-init.monk)

- Load stc_monksmime.dll successfully and define SMIMEH.
- Load dummy decryption key into SME cache.

B Inbound

- Verify digital signature.
- Extract content from RosettaNet Business Message.
- Extract signature from RosettaNet Business Message.

Note: *The signature should already be in base64 format, so no conversion is necessary.*

- Obtain sec_key_type values and tpic_id using ux-get-header.
- Call eX-ROS20-verify-sig passing in content, signature, algorithm, sec_key_type values and tpic_id.
- If eX-ROS20-verify-sig returns true (#t) then continue with normal processing.
- If eX-ROS20-verify-sig returns false (#f) then go to error processing.

C Decrypt message

- Extract encrypted portion of RosettaNet Business Message.

Note: *The encrypted portion should already be in base64 format, so no conversion is necessary.)*

- Obtain sec_key_type values and tpic_id using ux-get-header.
- Call eX-ROS20-decrypt-msg passing in encrypted message, sec_key_type values and tpic_id.
- If eX-ROS20-decrypt-msg returns data (not #f) then continue with normal processing using decrypted message.
- If eX-ROS20-decrypt-msg returns false (#f) then go to error processing.

D Outbound

Add Digital Signature

- Extract content from RosettaNet Business Message.
- Obtain sec_key_type values and tpic_id using ux-get-header.
- Call eX-ROS20-sign-msg passing in content, sec_key_type values and tpic_id.
- If eX-ROS20-sign-msg returns a vector containing signature algorithm and base64 encoded signature (not #f) then continue with normal processing using digital signature.
- If eX-ROS20-sign-msg returns false (#f) then go to error processing.

E Encrypt message

- Extract portion from RosettaNet Business Message to be encrypted.
- Obtain sec_key_type values and tpic_id using ux-get-header.
- Call eX-ROS20-encrypt-msg passing in content, sec_key_type values and tpic_id.

- If eX-ROS20-encrypt-msg returns a base64 encoded message (not #f) then continue with normal processing using encrypted message.
- If eX-ROS20-encrypt-msg returns false (#f) then go to error processing

The second operational group contains the following function:

- eX-ROS20-get-ssl-keys

This function is used in e*Xchange RosettaNet 2.0 scripts as described below.

A Outbound

- Find out if communicating via HTTPS with Trading Partner profile.
- If protocol = HTTPS then retrieve SSL information by calling eX-ROS20-get-ssl-keys.
- Take return vector and place the values in the standard event for the output.
- element 0 = SSLClientKeyFileName under TP_EVENT
- element 1 = SSLClientKeyFileType under TP_EVENT
- element 2 = SSLClientCertFileName under TP_EVENT
- element 3 = SSLClientCertFileType under TP_EVENT

A Note Regarding Security Function Examples

Note: *It is assumed that db-login, creation of connection-handle, and creation of SME handle (SMIMEH) were already executed successfully before each example. Also, the util functions assume error_data has previously been defined.*

The following table lists the security functions and their locations in this document.

util-security-decrypt-msg

Syntax

```
(util-security-decrypt-msg msg key_name key_value)
```

Description

util-security-decrypt-msg processes decryption of a given message. Calls Secure Message Extension functions to decrypt a message using decryption key and value.

Parameters

Name	Type	Description
msg	string	The encrypted msg in base64 format.
key_name	string	The decryption key name.
key_value	string	The decryption key value.

Return Values

string

Returns decrypted msg in raw readable format.

Boolean

Returns **#f** (false) if an error is encountered. **error_data** contains error string(s).

Throws

None.

Example

```
(define FILE (open-input-file "d:/temp/xi_encrypted_ROS20_msg"))
(define msg (read FILE 65536))
(close-input-port FILE)
(define FILE (open-input-file "c:/temp/certs/stc-ssl-client-test-1.p12"))
(define key_value (read FILE 65536))
(close-input-port FILE)
(define key_name "STC SSL Client Test #1")
(define decrypted_msg (util-security-decrypt-msg msg key_name key_value))
(if (boolean? decrypted_msg)
   (begin
    (display (string-append "Decryption failed with error: " error_data
         "\n"))
)
   (begin
    (display "Decryption succeeded - Decrypted msg placed in d:/temp/
         decrypted_msg\n")
    (define FILE (open-output-file "d:/temp/decrypted_msg"))
    (display decrypted_msg FILE)
    (close-port FILE)
)
)
```

util-security-encrypt-msg

Syntax

(util-security-encrypt-msg *msg cert_name cert_value algorithm*)

Description

util-security-encrypt-msg processes encryption of a given message. Calls Secure Message Extension functions to encrypt message using encryption certificate name and value, and encryption algorithm.

Parameters

Name	Type	Description
msg	string	The message (in raw readable format) to be encrypted.
cert_name	string	The encryption cert name.
cert_value	string	The encryption cert value.
algorithm	string	The encryption algorithm.

Return Values

string

Returns encrypted msg in base64 format (S/MIME headers have been stripped off).

Boolean

Returns #f (false) if an error is encountered. **error_data** contains error string(s).

Throws

None.

Example

```
(define FILE (open-input-file "d:/stc/egate/client/data/data2/con3A4.dat"))
(define msg (read FILE 65536))
(close-input-port FILE)
(define FILE (open-input-file "c:/temp/certs/stc-ssl-client-test-1.pk7"))
(define cert_value (read FILE 65536))
(close-input-port FILE)
(define cert_name "STC SSL Client Test #1")
(define alg "RC2_128")
(define encrypted_msg (util-security-encrypt-msg msg cert_name cert_value
alg))
(if (boolean? encrypted_msg)
   (begin
    (display (string-append "Encryption failed with error: "error_data
         "\n"))
)
   (begin
    (display "Encryption succeeded - Encrypted msg placed in d:/temp/
         encrypted_msg\n")
    (define FILE (open-output-file "d:/temp/encrypted_msg"))
    (display encrypted_msg FILE)
    (close-port FILE)
)
)
```

util-security-sign-msg

Syntax

```
(util-security-sign-msg msg key_name key_value algorithm)
```

Description

util-security-sign-msg creates a digital signature for a given message. Calls Secure Message Extension functions to create a digital signature for the given message using signature key name and value, and signature algorithm. Removes S/MIME headers and raw content before returning base64 encoded digital signature.

Parameters

Name	Type	Description
msg	string	The message (in raw readable format) to use for creating digital signature.
key_name	string	The signature key name.
key_value	string	The signature key value.
algorithm	string	The signature algorithm.

Return Values

string

Returns digital signature in base64 format (S/MIME headers and content have been stripped off)

Boolean

Returns #f (false) if an error is encountered. **error_data** contains error strings.

Throws

None.

Example

```
(define FILE (open-input-file "d:/stc/egate/client/data/con3A4.dat"))
(define msg (read FILE 65536))
(close-input-port FILE)
(define FILE (open-input-file "c:/temp/certs/stc-ssl-client-test-1.p12"))
(define key_value (read FILE 65536))
(close-input-port FILE)
(define key_name "STC SSL Client Test #1")
(define alg "RSA_MD5")
(define signed_msg (util-security-sign-msg msg key_name key_value alg))
(if (boolean? signed_msg)
   (begin
    (display (string-append "Signing failed with error: "error_data "\n"))
)
   (begin
    (display "Signing succeeded-Signed msg placed in d:/temp/signed_msg\n")
    (define FILE (open-output-file "d:/temp/signed_msg"))
    (display signed_msg FILE)
    (close-port FILE)
)
)
```

util-security-verify-sig

Syntax

```
(util-security-verify-sig content signature algorithm cert_name
cert_value)
```

Description

util-security-verify-sig performs verification of a digital signature for a given message. Calls Secure Message Extension functions to verify if the digital signature is valid for a given message using the certificate.

Parameters

Name	Type	Description
content	string	The content (in raw readable format) portion of data.
signature	string	The digital signature (in base64 format).
algorithm	string	The signature algorithm, such as MD5 or SHA1.
cert_name	string	The verification certificate name.
cert_value	string	The verification certificate value.

Return Values

Boolean

Returns **#t** (true) if verification succeeded, that is the content and digital signature match; otherwise returns **#f** (false) if content and digital signature do not match or an error was encountered. **error_data** contains error string(s).

Throws

None.

Example

```
(define FILE (open-input-file "d:/stc/egate/client/data/con3A4.dat"))
(define content (read FILE 65536))
(close-input-port FILE)
(define FILE (open-input-file "d:/temp/signed_msg"))
(define sig (read FILE 65536))
(close-input-port FILE)
(define FILE (open-input-file "c:/temp/certs/stc-ssl-client-test-
    1.pk7"))
(define cert_value (read FILE 65536))
(close-input-port FILE)
(define cert_name "STC SSL Client Test #1")
(define alg "MD5")
(if (util-security-verify-sig content sig alg cert_name cert_value)
    (display "Verification succeeded!\n")
    (display (string-append "Verification failed with error: "
        error_data "\n"))
)
```

eX-security-get-keys-certs

Syntax

```
(eX-security-get-keys-certs connection-handle tpic_id key)
```

Description

eX-security-get-keys-certs retrieves the key name, key value length and key value for a particular trading partner and key type.

Parameters

Name	Type	Description
connection-handle	string	The database connection handle.
tpic_id	string	The id for the trading partner profile.
key	string	The key type in es_security_key. Valid key types: E, S, D, V, K, T, C, P, Y, A.

Return Values

vector

Returns a 3 element vector containing key name, key value length, and key value.

Boolean

Returns **#t** (true) if there is no key name or value to retrieve; otherwise returns **#f** (false) if an error occurred. **error_data** contains error string(s)

Throws

None.

Example

```
(let ((ret_vec "")
     (tpic_id "12")
     (key "A"))
(set! ret_vec (eX-security-get-keys-certs connection-handle tpic_id key))
(if (vector? ret_vec)
   (begin
     (display "key name = <")
     (display (vector-ref ret_vec 0))
     (display ">\n")
     (display "length of key value = <")
     (display (vector-ref ret_vec 1))
     (display ">\n")
     (display "key value = <")
     (display (vector-ref ret_vec 2))
     (display ">\n")
)
(begin
   (if (eq? #t ret_vec)
     (display "No key/value to retrieve\n")
     (display "eX-security-get-keys-certs failed to retrieve key/value.")
)
)
)
)
```

eX-ROS20-decrypt-msg

Syntax

```
(eX-ROS20-decrypt-msg connection-handle msg keys tpic_id)
```

Description

eX-ROS20-decrypt-msg processes decryption of a given message. Calls eX-security-get-keys-certs to obtain decryption key and value. Calls util-security-decrypt-msg to perform decryption of given message using decryption key and value.

Parameters

Name	Type	Description
connection-handle	string	The database connection handle.
msg	string	Encrypted msg in base64 format.
keys	string	The key types for trading partner profile (obtained from es_tpic.sec_key_type). Must contain key type D.
tpic_id	string	The id for the trading partner profile.

Return Values

string

Returns encrypted msg in raw readable format.

Boolean

Returns **#f** (false) if an error is encountered. **error_data** contains error string(s).

Throws

None.

Example

```
(let ((dec_msg "")
     (tpic_id "13")
     (keys "D"))
(define FILE (open-input-file "d:/temp/encrypted_mime1.txt"))
(define msg (read FILE 65536))
(close-input-port FILE)
(set! dec_msg (eX-ROS20-decrypt-msg connection-handle msg keys tpic_id))
(if (boolean? dec_msg)
   (begin
    (display "eX-ROS20-decrypt-msg failed!\n")
)
   (begin
    (define FILE (open-output-file "d:/temp/decrypted_ROS20_msg"))
    (display dec_msg FILE)
    (close-port FILE)
)
)
)
```

eX-ROS20-encrypt-msg

Syntax

```
(eX-ROS20-encrypt-msg connection-handle msg keys tpic_id)
```

Description

eX-ROS20-encrypt-msg processes encryption of a given message. Calls eX-security-get-keys-certs to obtain encryption certificate name and value, and encryption algorithm. Calls util-security-encrypt-msg to perform encryption of given message using certificate and algorithm.

Parameters

Name	Type	Description
connection-handle	string	The database connection handle.
msg	string	The message in raw readable format, to encrypt.
keys	string	The key types for trading partner profile (obtained from es_tpic.sec_key_type). Must contain key types E and Y.
tpic_id	string	The id for the trading partner profile.

Return Values

string

Returns encrypted msg in base64 format.

Boolean

Returns **#f** (false) if an error is encountered. **error_data** contains error string(s).

Throws

None.

Example

```
(let ((enc_msg "")
     (tpic_id "14")
     (keys "D|Y|E"))
(define FILE (open-input-file "d:/temp/mime1.txt"))
(define msg (read FILE 65536))
(close-input-port FILE)
(set! enc_msg (eX-ROS20-encrypt-msg connection-handle msg keys tpic_id))
(if (boolean? enc_msg)
   (begin
    (display "eX-ROS20-encrypt-msg failed!\n")
)
   (begin
    (define FILE (open-output-file "d:/temp/encrypted_ROS20_msg"))
    (display enc_msg FILE)
    (close-port FILE)
)
)
)
```

eX-ROS20-sign-msg

Syntax

```
(eX-ROS20-sign-msg connection-handle msg keys tpic_id)
```

Description

eX-ROS20-sign-msg creates a digital signature for a given message. Calls eX-security-get-keys-certs to obtain signature key name and value, and signature algorithm. Calls util-security-sign-msg to create a digital signature for a given message using key and algorithm.

Parameters

Name	Type	Description
connection-handle	string	The database connection handle.
msg	string	The message in raw readable format, for which to create a digital signature.
keys	string	The key types for trading partner profile (obtained from es_tpic.sec_key_type). Must contain key types S and A.
tpic_id	string	The id for the trading partner profile.

Return Values

vector

Returns a 2 element vector containing algorithm and signature in base64 format.

Boolean

Returns **#f** (false) if an error is encountered. **error_data** contains error string(s).

Throws

None.

Example

```
(let ((sign_msg_vec "")
     (tpic_id "14")
     (keys "S|A"))
(define FILE (open-input-file "d:/temp/encrypted_ROS20_msg"))
(define msg (read FILE 65536))
(close-input-port FILE)
(set! sign_msg_vec (eX-ROS20-sign-msg connection-handle msg keys tpic_id))
(if (boolean? sign_msg_vec)
   (begin
    (display "eX-ROS20-sign-msg failed!\n")
)
   (begin
    (display (string-append "Algorithm = "
         (vector-ref sign_msg_vec 0) "\n"))
    (define FILE (open-output-file "d:/temp/signed_ROS20_msg"))
    (display (vector-ref sign_msg_vec 1) FILE)
    (close-port FILE)
)
)
)
```

eX-ROS20-verify-sig

Syntax

```
(eX-ROS20-verify-sig connection-handle content signature algorithm
keys tpic_id)
```

Description

eX-ROS20-verify-sig performs verification of a digital signature for a given message. Calls eX-security-get-keys-certs to obtain verification certificate name and value. Calls util-security-verify-sig to verify if the digital signature is valid for a given message using the certificate.

Parameters

Name	Type	Description
connection-handle	string	The database connection handle.
content	string	The content (in raw readable format) portion of data.
signature	string	The digital signature (in base64 format).
algorithm	string	The signature algorithm, such as MD5 or SHA1.
keys	string	The key types for trading partner profile (obtained from es_tpic.sec_key_type). Must contain key type V.
tpic_id	string	The id for the trading partner profile.

Return Values

Boolean

Returns **#t** (true) if verification succeeded, that is the content and digital signature match; otherwise returns **#f** (false) if the content and digital signature do not match or an error was encountered. **error_data** contains error string(s).

Throws

None.

Example

```
(let ((sig_msg "")
     (tpic_id "13")
     (alg "MD5")
     (keys "V"))
(define FILE (open-input-file "d:/temp/encrypted_ROS20_msg"))
(define content (read FILE 65536))
(close-input-port FILE)
(define FILE (open-input-file "d:/temp/signed_ROS20_msg"))
(define sig (read FILE 65536))
(close-input-port FILE)
(if (eq? #t (eX-ROS20-verify-sig connection-handle content sig
          alg keys tpic_id))
(begin
     (display "eX-ROS20-verify-sig succeeded!\n")
```

```
)
(begin
     (display "eX-ROS20-verify-sig failed!\n")
)
)
)
```

eX-ROS20-get-ssl-keys

Syntax

```
(eX-ROS20-get-ssl-keys connection-handle keys tpic_id)
```

Description

eX-ROS20-get-ssl-keys retrieves the SSL Client key and certificate for a certain trading partner profile. Calls eX-security-get-keys-certs to obtain the key and certificate values, and the file types.

Parameters

Name	Type	Description
connection-handle	string	The database connection handle.
keys	string	The key types for trading partner profile (obtained from es_tpic.sec_key_type). Must contain key types K, T, C, and P.
tpic_id	string	The id for the trading partner profile.

Return Values

vector

Returns a 4 element vector containing the key, certificate, and file types

#(ssl_key_value in base64 format

ssl key file type (either PEM or ASN.1)

ssl certificate value in base64 format

ssl certificate file type (either PEM or ASN.1)

)

Boolean

Returns **#f** (false) if an error is encountered. **error_data** contains error string(s).

Throws

None.

Example

```
(let ((out_vec "")
    (tpic_id "12")
    (keys "K|C|T|P"))
(set! out_vec (eX-ROS20-get-ssl-keys connection-handle keys tpic_id))
(if (eq? #f out_vec)
   (begin
    (display "eX-ROS20-get-ssl-keys failed!\n")
)
   (begin
    (define FILE (open-output-file "d:/temp/ssl_key_base64"))
    (display (vector-ref out_vec 0) FILE)
    (close-port FILE)
    (display "SSL Key File Type = ")
```

```
        (display (vector-ref out_vec 1))
        (newline)
        (define FILE (open-output-file "d:/temp/ssl_cert_base64"))
        (display (vector-ref out_vec 2) FILE)
        (close-port FILE)
        (display "SSL Cert File Type = ")
        (display (vector-ref out_vec 3))
        (newline)
        (display "eX-ROS20-get-ssl-keys succeeded!\n")
    )
    )
    )
```

Chapter 14

Java Helper Methods

A number of Java methods have been added to make it easier to set information in the e*Xchange Event (**eX_StandardEvent.xsc** ETD) and to get information from it. These methods are contained in classes:

14.1 NameValuePair Class

public class NameValuePair

extends com.stc.jcsre.XMLETDImpl

implements com.stc.jcsre.ETD

A class to represent the NameValuePair object of an e*Xchange (Business Process Management) XML ETD. It is defined in the following DTD:

```
<!ELEMENT TPAttribute (NameValuePair*)>
<!ELEMENT NameValuePair (Name, Value)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```

These methods are described in detail on the following pages:

getNAME

Syntax

```
java.lang.String getNAME()
```

Description

getNAME retrieves the name portion of this object.

Parameters

None.

Return Values

java.lang.String
Returns the name of this object.

Throws

None.

Example

```
getNAME();
=> "COMM_PROT"
```

getVALUE

Syntax

```
java.lang.String getVALUE()
```

Description

getVALUE retrieves the value portion of this object.

Parameters

None.

Return Values

java.lang.String
Returns the value of the object.

Throws

None.

Example

```
getVALUE();
=> "X12"
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.ContentHandler	The handler that converts content within to XML.
errorHandler	org.xml.sax.ErrorHandler	The handler to address errors during conversion.

Return Values

None.

Throws

None.

setNAME

Syntax

```
void setNAME(java.lang.String val)
```

Description

setNAME sets the name portion of this object.

Parameters

Name	Type	Description
val	string	The case-sensitive name of this Partner Manager Attribute.

Return Values

None.

Throws

None.

Example

```
setNAME("COMM_PROT");
```

setVALUE

Syntax

```
void setVALUE(java.lang.String val)
```

Description

setVALUE sets the value portion of this object.

Parameters

Name	Type	Description
val	java.lang.String	The value of the Attribute.

Return Values

None.

Throws

None.

Example

```
setVALUE("X12");
```

toString

Syntax

```
java.lang.String toString()
```

Description

toString converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message represent by this ETD object.

Throws

None.

Example

```
toString();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the ACTIVITY XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.sml.SAXLexer	The SAX lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException - thrown when the data cannot be parsed

com.stc.jcsre.UnmarshalException - throw when the data cannot be unmarshalled

14.2 Payload Class

public class Payload

extends com.stc.jcsre.XMLETDImpl

implements com.stc.jcsre.ETD

A class to represent the Payload object of an e*Xchange (Partner Manager) XML ETD. It is defined in the following DTD:

```
<!--Payload to carry EDI message-->
<!ELEMENT Payload (#PCDATA)>
<!ATTLIST Payload
        TYPE (RAW | PROCESSED | ENCRYPTED) #REQUIRED
        LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
```

These methods are described on the following pages:

get$Text

Syntax

```
java.lang.String get$Text()
```

Description

get$Text retrieves the Payload data.

Parameters

None.

Return Values

java.lang.String

Returns the Payload data.

Throws

None.

getLOCATION

Syntax

```
java.lang.String getLOCATION()
```

Description

getLOCATION retrieves the location type of where the data for Payload is actually stored. In cases where the data is too long to be stored in standard database column, it can be stored in another table where the column can be defined as a "LONG RAW" for example, or it may be stored in a file on some file system. In such cases, a reference to the actual data location is stored as the data for Payload.

Parameters

None.

Return Values

java.lang.String

Returns the location type for the Payload data. This is one of the following values:

"FILE"	The Payload data contains the name of a file where actual data is stored.
"DB"	The Payload data contains a reference such as "ROWID" to a row in a table.
"URL"	The Payload data contains a URL to where the actual data is stored.
"EMBEDDED"	The Payload data contains the actual data (this is the default).
"AUTO"	The Payload data contains the actual data storage location is automatically determined by the e*Xchange engine.

Throws

None.

Example

```
getLOCATION();

=> "EMBEDDED"
```

getTYPE

Syntax

```
java.lang.String getTYPE()
```

Description

getTYPE retrieves the type of data stored in the Payload object.

Parameters

None.

Return Values

java.lang.String
Returns the type of data stored as one of the following values:

"RAW"	Indicates that the data in the **Data** node is in ASCII format, but not XML data that has been converted to ASCII using base 64 or some other conversion. The data must not contain any characters that would conflict with the XML nature of the e*Xchange ETD (for example, EDI delimiters that are the same as XML control characters).
"PROCESSED"	Indicates that the data in the **Data** node is XML data that has been encoded using the scheme described in the **ENCODING** node. Currently only base 64 encoding is supported.
"ENCRYPTED"	Indicates that the data in the **Data** node has been encrypted, and must be decrypted before it can be processed by e*Xchange.

Throws

None.

Example

```
getTYPE();

=> "STRING"
```

hasLOCATION

Syntax

```
boolean hasLOCATION()
```

Description

hasLOCATION checks if the location is defined for this Payload object.

Parameters

None.

Return Values

boolean
Returns true if location exists, otherwise returns false.

Throws

None.

Example

```
hasLOCATION();

=> true
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.ContentHandler	The handler that converts content within to XML.
errorHandler	or.xml.sax.ErrorHandler	The handler to address errors during conversion.

Return Values

None.

Throws

com.stc.jcsre.MarshalException

org.xml.sax.SAXException

omitLOCATION

Syntax

```
void omitLOCATION()
```

Description

omitLOCATION removes the location definition for this Payload object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitLOCATION();
```

set$Text

Syntax

```
void set$Text(java.lang.String val)
```

Description

set$Text sets the Payload data.

Parameters

Name	Type	Description
val	java.lang.String	The Payload data.

Return Values

None.

Throws

None.

setLOCATION

Syntax

```
void setLOCATION(java.lang.String val)
```

Description

setLOCATION sets the location type of where the data for a Payload is actually stored. In cases where the data is too long to be stored in standard database column, it can be stored in another table where the column can be defined as a "LONG RAW" for example, or it may be stored in a file on some file system.

Parameters

Name	Type	Description
val	java.lang.String	The location type for the Payload data. This can have one the following values: ▪ "FILE" - Attribute data is the name of a file where actual data is stored. ▪ "DB" - Attribute data is a reference such as "ROWID" to a row in a table. ▪ "URL" - Attribute data is the URL to where the actual data is stored. ▪ "EMBEDDED" - Attribute data is the actual data (this is the default). ▪ "AUTO" - The actual data storage location is automatically determined by the e*Xchange engine.

Return Values

None.

Throws

None.

Example

```
setLOCATION("FILE");
```

setTYPE

Syntax

```
void setTYPE(java.lang.String val)
```

Description

setTYPE sets the type of data stored in Payload.

Parameters

Name	Type	Description
val	java.lang.String	The type of data stored. This can take one of the following values: ▪ "RAW" - Indicates that the data in the **Data** node is in ASCII format, but not XML data that has been converted to ASCII using base 64 or some other conversion. The data must not contain any characters that would conflict with the XML nature of the e*Xchange ETD (for example, EDI delimiters that are the same as XML control characters). ▪ "PROCESSED" - Indicates that the data in the **Data** node is XML data that has been encoded using the scheme described in the **ENCODING** node. Currently only base 64 encoding is supported. ▪ "ENCRYPTED"- Indicates that the data in the Data node has been encrypted, and must be decrypted before it can be processed by e*Xchange.

Return Values

None.

Throws

None.

Example

setTYPE("STRING");

toString

Syntax

```
java.lang.String toString()
```

Description

toString Converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message represent by this ETD object.

Throws

None.

Example

toString();

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the e*Insight XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

14.3 TPAttribute Class

public class TPAttribute

extends com.stc.jcsre.XMLETDImpl

implements com.stc.jcsre.ETD

The TPAttribute class represents the e*Xchange section of the SeeBeyond eBI Standard XML ETD which is used to communicate with the e*Xchange engine. The DTD is:

```
<!--TP Attribute will contain optional repeating name value pair for
storing of TP-->
<!ELEMENT TPAttribute (NameValuePair*)>
```

These methods are described in detail on the following pages:

addNameValuePair

Syntax

```
void addNameValuePair(int index, NameValuePair value)
```

Description

addNameValuePair inserts a new NameValuePair into this Trading Partner object.

Parameters

Name	Type	Description
index	integer	(Optional) Zero-base index to where the NameValuePair object is to be inserted.
value	com.stc.eBIpkg.ATTRIBUTE	The NameValuePair object.

Return Values

None.

Throws

None.

clearNameValuePair

Syntax

```
void clearNameValuePair()
```

Description

clearNameValuePair removes all the NameValuePairs from this TPAttribute object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
clearNameValuePair();
```

countNameValuePair

Syntax

```
int countNameValuePair()
```

Description

countNameValuePair retrieves the number of NameValuePair objects this TPAttribute object.

Parameters

None.

Return Values

integer

Returns the number of NameValuePair objects as an integer.

Throws

None

Example

```
countNameValuePair();
=>3
```

getNameValuePair_Value

Syntax

```
java.lang.String getNameValuePair_Value(java.lang.String name)
```

Description

getNameValuePair_Value retrieves the value of a specific NameValuePair by name.

Parameters

Name	Type	Description
name	java.lang.String	The name of the NameValuePair object.

Return Values

java.lang.String
Returns the value of the NameValuePair object. Can be null if the Attribute of that name does not exist.

Throws

None.

getNameValuePair

Syntax

```
NameValuePair[] getNameValuePair()
NameValuePair getNameValuePair(int i)
NameValuePair getNameValuePair(java.lang.String name)
```

Description

getNameValuePair retrieves all the NameValuePair objects in the TPAttribute object, or can be used to retrieve a specific NameValuePair by name or index.

Parameters

Name	Type	Description
i	integer	(Optional) The offset to the list where the NameValuePair appears.
name	java.lang.String	The NameValuePair name.

Return Values

Returns one of the following values:

NameValuePair[]
Returns an array of NameValuePair objects if no name or offset were specified.

NameValuePair
Returns the NameValuePair object if the name or offset were specified.

Throws

None.

hasNameValuePair

Syntax

```
boolean hasNameValuePair(java.lang.String name)
```

Description

hasNameValuePair checks whether a specific TP Attribute contains a NameValuePair.

Parameters

Name	Type	Description
name	java.lang.String	The name of the Trading Partner Attribute (TPAttribute) that you want to retrieve.

Return Values

boolean

Returns true if the NameValuePair is defined; otherwise returns false.

Throws

None.

Example

```
hasNameValuePair();
=>true
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.ContentHandler	The handler that converts content within to XML.
errorHandler	or.xml.sax.ErrorHandler	The handler to address errors during conversion.

Return Values

None.

Throws

com.stc.jcsre.MarshalException

org.xml.sax.SAXException

removeNameValuePair

Syntax

```
void removeNameValuePair(java.lang.String name)
void removeNameValuePair(int index)
```

Description

removeNameValuePair removes a specific NameValuePair object from this TPAttribute object by name or index.

Parameters

Name	Type	Description
name	java.lang.String	The name of the NameValuePair.
index	int	The index to the list of global attributes (zero-based).

Return Values

None.

Throws

None.

setNameValuePair

Syntax

```
void setNameValuePair(NameValuePair[] val)
void setNameValuePair(int i, NameValuePair val)
void setNameValuePair(java.lang.String name java.lang.String value)
```

Description

setNameValuePair sets a NameValuePair object in the TPAttribute object.

Parameters

Name	Type	Description
val	NameValuePair[]	The NameValuePair object.
i	int	The list index of the NameValuePair to be retrieved (zero-based).
name	java.lang.String	The name of the NameValuePair.
value	java.lang.String	The value of the NameValuePair.

Return Values

None.

Throws

None.

toString

Syntax

```
java.lang.String toString()
```

Description

toString converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message to represent by this ETD object.

Throws

None.

Example

```
toString();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the e*Insight XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

14.4 TP_EVENT Class

public class TP_EVENT

extends com.stc.jcsre.SMLETDImpl

implements com.eBIpkg.TPEventETD

TP_EVENT class represents the e*Xchange section of the SeeBeyond eBI Standard XML ETD which is used to communicate with the e*Xchange engine. The DTD is:

```
<!--ePartner Manager Input/Output Event section-->
<!ELEMENT TP_EVENT (PartnerName?, InternalName?, Direction?,
    MessageID?, OrigEventC
<!--External Partner Name-->
<!ELEMENT PartnerName (#PCDATA)>
<!--Internal Sending ERP (ex.SAP)-->
<!ELEMENT InternalName (#PCDATA)>
<!--Direction of Transaction to/from Trading Partner (ex.Outbound=O
    Inbound=I)-->
<!ELEMENT Direction (#PCDATA)>
<!--Original request ID from Internal Sending ERP-->
<!ELEMENT MessageID (#PCDATA)>
<!--Original Event Classification (ex.QAP for Query Price and
    Availability)-->
<!ELEMENT OrigEventClass (#PCDATA)>
<!--Usage Indicator of EDI message by Trading Partner (Production=P
    Test=T)-->
<!ELEMENT UsageIndicator (#PCDATA)>
<!--Payload to carry EDI message-->
<!ELEMENT Payload (#PCDATA)>
<!ATTLIST Payload
        TYPE (RAW | PROCESSED | ENCRYPTED) #REQUIRED
        LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!--RAW=Need translation PROCESSED=Already X12 or RN ENCRYPTED=from
    Trading Partner>
<!--Communication Protocol (ex. BATCH, HTTP) for sending to Trading
    Partner-->
<!ELEMENT CommProt (#PCDATA)>
<!--URL for EDI message to be exchanged with Trading Partner-->
<!ELEMENT Url (#PCDATA)>
<!--SSL information-->
<!ELEMENT SSLClientKeyFileName (#PCDATA)>
<!ELEMENT SSLClientKeyFileType (#PCDATA)>
<!ELEMENT SSLClientCertFileName (#PCDATA)>
<!ELEMENT SSLClientCertFileType (#PCDATA)>
<!--Message Index for Batched delivery, ex. 1|20 means 1 of 20-->
<!ELEMENT MessageIndex (#PCDATA)>
<!--TP Attribute will contain optional repeating name value pair for
storing of TP-->
<!ELEMENT TPAttribute (NameValuePair*)>
<!ELEMENT NameValuePair (Name, Value)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```

These methods are described in detail on the following pages:

getCommProt

Syntax

```
java.lang.String getCommProt()
```

Description

getCommProt retrieves the communication protocol used.

Parameters

None.

Return Values

java.lang.String
Returns the communication protocol. Possible values include “BATCH”, “HTTP” and “HTTPS”.

Throws

None.

Example

```
getCommProt();
=>"BATCH"
```

getDirection

Syntax

```
java.lang.String getDirection()
```

Description

getDirection retrieves the direction of transaction relative to e*Xchange.

Parameters

None.

Return Values

java.lang.String
Returns the direction. Possible values include; “O” representing Outbound, or “I” representing Inbound.

Throws

None.

Example

```
getDirection();
=>"I"
```

getInternalName

Syntax

```
java.lang.String getInternalName()
```

Description

getInternalName retrieves the internal name of the Trading Partner as known by the sending ERP.

Parameters

None.

Return Values

java.lang.String
Returns the Internal Name of the Trading Partner per sending ERP.

Throws

None.

getMessageID

Syntax

```
java.lang.String getMessageID()
```

Description

getMessageID retrieves the original request ID from the sending ERP system.

Parameters

None.

Return Values

java.lang.String
Returns the message ID.

Throws

None.

getMessageIndex

Syntax

```
java.lang.String getMessageIndex()
```

Description

getMessageIndex retrieves the message index for batched delivery. For example, 5/7 indicates the fifth message in a batch of seven.

Parameters

None.

Return Values

java.lang.String
Returns the message index.

Throws

None.

getOrigEventClass

Syntax

```
java.lang.String getOrigEventClass()
```

Description

getOrigEventClass retrieves the original event classification from the sending ERP system.

Parameters

None.

Return Values

java.lang.String
Returns the original event classification.

Throws

None.

getPartnerName

Syntax

```
java.lang.String getPartnerName()
```

Description

getPartnerName retrieves the name of the Trading Partner.

Parameters

None.

Return Values

java.lang.String
Returns the Trading Partner Name.

Throws

None.

Example

```
getPartnerName();
=>"The Savvy Toy Company"
```

getPayload

Syntax

```
java.lang.String getPayload()
```

Description

getPayload retrieves the Payload object.

Parameters

None.

Return Values

Payload
Returns the Payload object.

Throws

None.

getSSLClientCertFileName

Syntax

```
java.lang.String SSLClientCertFileName()
```

Description

SSLClientCertFileName retrieves the name of the file that contains the SSL Client Certificate.

Parameters

None.

Return Values

java.lang.String
Returns the name of the file that contains the SSL Client Certificate.

Throws

None.

getSSLClientCertFileType

Syntax

```
java.lang.String getSSLClientCertFileType()
```

Description

SSLClientCertFileType retrieves the SSL Client Certificate file type.

Parameters

None.

Return Values

java.lang.String
Returns the SSL Client Certificate file type. Possible values include "ASN.1" and "PEM".

Throws

None.

Example

```
getSSLClientCertFileType();
=>"PEM"
```

getSSLClientKeyFileName

Syntax

```
java.lang.String SSLClientKeyFileName()
```

Description

SSLClientKeyFileName retrieves the name of the file that contains the SSL Client Key.

Parameters

None.

Return Values

java.lang.String
Returns the SSL Client Key file name.

Throws

None.

getSSLClientKeyFileType

Syntax

```
java.lang.String SSLClientKeyFileType()
```

Description

SSLClientKeyFileType retrieves the SSL Client Key file type.

Parameters

None.

Return Values

java.lang.String
Returns the SSL Client Key file type. Possible values include “ASN.1” and “PEM”.

Throws

None.

Example

```
getSSLClientKeyFileType();
=>"PEM"
```

getTPAttribute

Syntax

```
TPAttribute getTPAttribute()
```

Description

getTPAttribute retrieves the TPAttribute object.

Parameters

None.

Return Values

TPAttribute
Returns the TPAttribute object.

Throws

None.

getURL

Syntax

```
java.lang.String getURL()
```

Description

getURL retrieves the URL used for an EDI message exchanged with a Trading Partner.

Parameters

None.

Return Values

java.lang.String
Returns the URL.

Throws

None.

getUsageIndicator

Syntax

```
java.lang.String getUsageIndicator()
```

Description

getUsageIndicator retrieves the usage indicator for a Trading Partner object.

Parameters

None.

Return Values

java.lang.String
Returns the usage indicator. Possible values include “P” representing Production and “T” representing Testing.

Throws

None.

Example

```
getUsageIndicator();
=>"P"
```

hasCommProt

Syntax

```
boolean hasCommProt()
```

Description

hasCommProt checks whether the communication protocol has been defined in this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the communication protocol exists.

Throws

None.

Example

```
hasCommProt();
=>true
```

hasDirection

Syntax

```
boolean hasDirection()
```

Description

hasDirection checks whether the direction has been defined in this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the direction exists.

Throws

None.

Example

```
hasDirection();
=>true
```

hasInternalName

Syntax

```
boolean hasInternalName()
```

Description

hasInternalName checks whether the internal name of the Trading Partner, as known by the sending ERP, has been defined for this Trading Partner.

Parameters

None.

Return Values

boolean
Returns true if the internal name exists.

Throws

None.

Example

```
hasInternalName();
=>true
```

hasMessageID

Syntax

```
boolean hasMessageID()
```

Description

hasMessageID checks whether the original request ID from the internal sending ERP is defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the request id exists.

Throws

None.

Example

```
hasMessageID();
=>true
```

hasMessageIndex

Syntax

```
boolean hasMessageIndex()
```

Description

hasMessageIndex checks whether the message index for batched delivery has been defined for this Trading Partner.

Parameters

None.

Return Values

boolean
Returns true if the message index exists.

Throws

None.

Example

```
hasMessageIndex();
=>true
```

hasOrigEventClass

Syntax

```
boolean hasOrigEventClass()
```

Description

hasOrigEventClass checks whether the original Event classification has been defined for this Trading Partner.

Parameters

None.

Return Values

boolean
Returns true if the original Event classification exists.

Throws

None.

Example

```
hasOrigEventClass();
=>true
```

hasPartnerName

Syntax

```
boolean hasPartnerName()
```

Description

hasPartnerName checks whether the Trading Partner name has been defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the Partner name exists.

Throws

None.

Example

```
hasPartnerName();
=>true
```

hasPayload

Syntax

```
boolean hasPayload()
```

Description

hasPayload checks whether the Payload has been defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the Payload exists.

Throws

None.

Example

```
hasPayload();
=>true
```

hasSSLClientCertFileName

Syntax

```
boolean hasSSLClientCertFileName()
```

Description

hasSSLClientCertFileName checks whether the file name for the SSL Client Certificate has been defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the file name exists.

Throws

None.

Example

```
hasSSLClientCertFileName();
=>true
```

hasSSLClientCertFileType

Syntax

```
boolean hasSSLClientCertFileType()
```

Description

hasSSLClientCertFileType checks whether the SSL Client Certificate file type has been defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the file type exists.

Throws

None.

Example

```
hasSSLClientCertFileType();
=>true
```

hasSSLClientKeyFileName

Syntax

```
boolean hasSSLClientKeyFileName()
```

Description

hasSSLClientKeyFileName checks whether the file name for the SSL Client Key has been defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the file name exists.

Throws

None.

Example

```
hasSSLClientKeyFileName();
=>true
```

hasSSLClientKeyFileType

Syntax

```
boolean hasSSLClientKeyFileType()
```

Description

hasSSLClientKeyFileType checks whether the SSL Client Key file type has been defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the file type exists.

Throws

None.

Example

```
hasSSLClientKeyFileType();
=>true
```

hasTPAttribute

Syntax

```
boolean hasTPAttribute()
```

Description

hasTPAttribute checks whether the TPAttribute has been defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the TPAttribute exists.

Throws

None.

Example

```
hasTPAttribute();
=>true
```

hasUrl

Syntax

```
boolean hasUrl()
```

Description

hasUrl checks whether the URL for an EDI message has been defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the URL exists.

Throws

None.

Example

```
hasURL();
=>true
```

hasUsageIndicator

Syntax

```
boolean hasUsageIndicator()
```

Description

hasUsageIndicator checks whether the usage indicator has been defined for this Trading Partner object.

Parameters

None.

Return Values

boolean
Returns true if the usage indicator exists.

Throws

None.

Example

```
hasUsageIndicator();
=>true
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.ContentHandler	The handler that converts content within to XML.
errorHandler	org.xml.sax.ErrorHandler	The handler to address errors during conversion.

Return Values

None.

Throws

com.stc.jcsre.MarshalException

org.xml.sax.SAXException

omitCommProt

Syntax

```
void omitCommProt()
```

Description

omitCommProt removes the communication protocol from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitCommProt();
```

omitDirection

Syntax

```
void omitDirection()
```

Description

omitDirection removes the direction from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitDirection();
```

omitInternalName

Syntax

```
void omitInternalName()
```

Description

omitInternalName removes the definition of the Internal Name of the Trading Partner as known by the sending ERP System from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitInternalName();
```

omitMessageID

Syntax

```
void omitMessageID()
```

Description

omitMessageID removes the original request ID from the internal sending ERP from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitMessageID();
```

omitMessageIndex

Syntax

```
void omitMessageIndex()
```

Description

omitMessageIndex removes the message index for batched delivery from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitMessageIndex();
```

omitOrigEventClass

Syntax

```
void omitOrigEventClass()
```

Description

omitOrigEventClass removes the original Event classification from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitOrigEventClass();
```

omitPartnerName

Syntax

```
void omitPartnerName()
```

Description

omitPartnerName removes the Trading Partner name definition from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitPartnerName();
```

omitPayload

Syntax

```
void omitPayload()
```

Description

omitPayload removes the Payload definition from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitPayload();
```

omitSSLClientCertFileName

Syntax

```
void omitSSLClientCertFileName()
```

Description

omitSSLClientCertFileName removes the file name for the SSL Client Certificate from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitSSLClientCertFileName();
```

omitSSLClientCertFileType

Syntax

```
void omitSSLClientCertFileType()
```

Description

omitSSLClientCertFileType removes the SSL Client Certificate file type from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitSSLClientCertFileType();
```

omitSSLClientKeyFileName

Syntax

```
void omitSSLClientKeyFileName()
```

Description

omitSSLClientKeyFileName removes the file name for the SSL Client Key from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitSSLClientKeyFileName();
```

omitSSLClientKeyFileType

Syntax

```
void omitSSLClientKeyFileType()
```

Description

omitSSLClientKeyFileType removes the SSL Client Key file type from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitSSLClientKeyFileType();
```

omitTPAttribute

Syntax

```
void omitTPAttribute()
```

Description

omitTPAttribute removes the TPAttribute from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitTPAttribute();
```

omitUrl

Syntax

```
void omitUrl()
```

Description

omitUrl removes the URL for EDI messages from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitURL();
```

omitUsageIndicator

Syntax

```
void omitUsageIndicator()
```

Description

omitUsageIndicator removes the usage indicator from this Trading Partner object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitUsageIndicator();
```

setCommProt

Syntax

```
void setCommProt(java.lang.String val)
```

Description

setCommProt sets the communication protocol.

Parameters

Name	Type	Description
val	java.lang.String	The communication protocol. Possible values include: ▪ "BATCH" ▪ "HTTP" ▪ "HTTPS"

Return Values

None.

Throws

None.

Example

```
setCommProt("BATCH");
```

setDirection

Syntax

```
void setDirection(java.lang.String val)
```

Description

setDirection sets the direction of the transaction to or from the trading partner.

Parameters

Name	Type	Description
val	java.lang.String	The direction. Possible values include: ▪ "O" - Outbound ▪ "I" - Inbound

Return Values

None.

Throws

None.

Example

```
setDirection("O");
```

setInternalName

Syntax

```
void setInternalName(java.lang.String val)
```

Description

setInternalName sets the Internal Name of the Trading Partner as known by the sending ERP System.

Parameters

Name	Type	Description
val	java.lang.String	The internal sending ERP name.

Return Values

None.

Throws

None.

setMessageID

Syntax

```
void setMessageID(java.lang.String val)
```

Description

setMessageID sets the original request ID from the internal sending ERP.

Parameters

Name	Type	Description
val	java.lang.String	The original request ID from the internal sending ERP.

Return Values

None.

Throws

None.

setMessageIndex

Syntax

```
void setMessageIndex(java.lang.String val)
```

Description

setMessageIndex sets the message index for batched delivery. For example, 5/7 indicates the fifth message in a batch of seven.

Parameters

Name	Type	Description
val	java.lang.String	The message index for batched delivery.

Return Values

None.

Throws

None.

setOrigEventClass

Syntax

```
void setOrigEventClass(java.lang.String val)
```

Description

setOrigEventClass sets the original Event classification for the Trading Partner object.

Parameters

Name	Type	Description
val	java.lang.String	The original Event classification.

Return Values

None.

Throws

None.

setPartnerName

Syntax

```
void setPartnerName(java.lang.String val)
```

Description

setPartnerName sets the Trading Partner name for the Trading Partner object.

Parameters

Name	Type	Description
val	java.lang.String	The Trading Partner name.

Return Values

None.

Throws

None.

setPayload

Syntax

```
void setPayload(Payload val)
```

Description

setPayload sets the Payload for the Trading Partner object.

Parameters

Name	Type	Description
val	java.lang.String	The Payload object.

Return Values

None.

Throws

None.

setSSLClientCertFileName

Syntax

```
void setSSLClientCertFileName(java.lang.String val)
```

Description

setSSLClientCertFileName sets the file name for the SSL Client Certificate.

Parameters

Name	Type	Description
val	java.lang.String	The file name.

Return Values

None.

Throws

None.

setSSLClientCertFileType

Syntax

```
void setSSLClientCertFileType(java.lang.String val)
```

Description

setSSLClientCertFileType sets the SSL Client Certificate file type.

Parameters

Name	Type	Description
val	java.lang.String	The file type. Possible values include “ASN.1” and “PEM”.

Return Values

None.

Throws

None.

Example

```
setSSLClientCertFileType("PEM");
```

setSSLClientKeyFileName

Syntax

```
void setSSLClientKeyFileName(java.lang.String val)
```

Description

setSSLClientKeyFileName sets the file name for the SSL Client Key.

Parameters

Name	Type	Description
val	java.lang.String	The file name.

Return Values

None.

Throws

None.

setSSLClientKeyFileType

Syntax

```
void setSSLClientKeyFileType(java.lang.String val)
```

Description

setSSLClientKeyFileType sets the SSL Client Key file type.

Parameters

Name	Type	Description
val	java.lang.String	The file type. Possible values include “ASN.1” and “PEM”.

Return Values

None.

Throws

None.

Example

```
setSSLClientKeyFileType("PEM");
```

setTPAttribute

Syntax

```
void setTPAttribute(TPAttribute val)
```

Description

setTPAttribute sets the TPAttribute for the Trading Partner object.

Parameters

Name	Type	Description
val	TPAttribute	The TPAttribute object.

Return Values

None.

Throws

None.

setUrl

Syntax

```
void setUrl(java.lang.String val)
```

Description

setUrl sets the URL for an EDI message to be exchanged with a Trading Partner.

Parameters

Name	Type	Description
val	java.lang.String	The URL.

Return Values

None.

Throws

None.

setUsageIndicator

Syntax

```
void setUsageIndicator(java.lang.String val)
```

Description

setUsageIndicator sets the usage indicator of an EDI message.

Parameters

Name	Type	Description
val	java.lang.String	The usage indicator of an EDI message. Possible values include: ▪ "P" - Production ▪ "T" - Test

Return Values

None.

Throws

None.

Example

```
setUsageIndicator("P");
```

toString

Syntax

```
java.lang.String toString()
```

Description

toString converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message to represent by this ETD object.

Throws

None.

Example

```
toString();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the e*Insight XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

Appendix A

XML Structure for the e*Xchange Event

This appendix shows the XML structure for the e*Xchange Event Type Definitions (**eX_Standard_Event.ssc** and **eX_StandardEvent.xsc**). If your data conforms to this structure, you do not need to convert it upon entry to the e*Xchange system.

A.1 XML Structure

```
<<!-- edited with XML Spy v3.0 NT (http://www.xmlspy.com) by STC (STC)
-->
<!--DTD for eX_Standard_Event.ssc   $Id: eX_event.dtd,v 1.1.2.10
2000/09/07 04:43:14 galbers Exp $-->
<!--ePartner Manager Input/Output Event section-->
<!ELEMENT TP_EVENT (PartnerName?, InternalName?, Direction?,
    MessageID?, OrigEventC
<!--External Partner Name-->
<!ELEMENT PartnerName (#PCDATA)>
<!--Internal Sending ERP (ex.SAP)-->
<!ELEMENT InternalName (#PCDATA)>
<!--Direction of Transaction to/from Trading Partner (ex.Outbound=O
    Inbound=I)-->
<!ELEMENT Direction (#PCDATA)>
<!--Original request ID from Internal Sending ERP-->
<!ELEMENT MessageID (#PCDATA)>
<!--Original Event Classification (ex.QAP for Query Price and
    Availability)-->
<!ELEMENT OrigEventClass (#PCDATA)>
<!--Usage Indicator of EDI message by Trading Partner (Production=P
    Test=T)-->
<!ELEMENT UsageIndicator (#PCDATA)>
<!--Payload to carry EDI message-->
<!ELEMENT Payload (#PCDATA)>
<!ATTLIST Payload
        TYPE (RAW | PROCESSED | ENCRYPTED) #REQUIRED
        LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!--RAW=Need translation PROCESSED=Already X12 or RN ENCRYPTED=from
    Trading Partner>
<!--Communication Protocol (ex. BATCH, HTTP) for sending to Trading
    Partner-->
<!ELEMENT CommProt (#PCDATA)>
<!--URL for EDI message to be exchanged with Trading Partner-->
<!ELEMENT Url (#PCDATA)>
<!--SSL information-->
<!ELEMENT SSLClientKeyFileName (#PCDATA)>
<!ELEMENT SSLClientKeyFileType (#PCDATA)>
<!ELEMENT SSLClientCertFileName (#PCDATA)>
<!ELEMENT SSLClientCertFileType (#PCDATA)>
```

```
<!--Message Index for Batched delivery, ex. 1|20 means 1 of 20-->
<!ELEMENT MessageIndex (#PCDATA)>
<!--TP Attribute will contain optional repeating name value pair for
storing of TP-->
<!ELEMENT TPAttribute (NameValuePair*)>
<!ELEMENT NameValuePair (Name, Value)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```

Glossary

attribute (e*Insight)
Attributes pass user-defined control information (programming arguments) to and from the e*Insight Business Process Manager and its activities.

activity
An activity is an organizational unit for performing a specific function.

API
An acronym for Application Program Interface, which is a set of protocols, routines, and tools for building software applications. The e*Xchange API consists of a set of Monk functions that can be called from custom validation Collaborations to interface with the database.

business process
A business process is a collection of actions and messages, revolving around a specific business practice, that flow in a specific pattern to produce an end result.

business process instance (BPI)
A unique instantiation of a business process.

Collaboration
A component of an e*Way or BOB that receives and processes Events and forwards the output to other e*Gate components. Collaborations perform three functions: they subscribe to Events of a known type, they apply business rules to Event data, and they publish output Events to a specified recipient. Collaborations use Monk translation script files with the extension "**.tsc**" to do the actual data manipulation.

Company (e*Xchange)
An organization with which you conduct electronic business (eBusiness). A company can consist of one or more trading partners. See also *Trading partner*.

compensating transaction

A transaction that when executed undoes the effects of the transaction for which it is compensating. When a transaction and the compensating transaction are both executed, there should be no net change in the state of affairs. For example, a $100 debit is the compensating transaction for a $100 credit.

e*Insight Business Process Manager (e*Insight)
The component within the SeeBeyond eBI Product Suite that facilitates the automation of the business process flow of eBusiness activities.

eBusiness protocol
Generally accepted standards for formatting and exchanging electronic messages between trading partners. ANSI X12, RosettaNet, and BizTalk are examples of eBusiness protocols.

e*Xchange Partner Manager (e*Xchange)
An application within the SeeBeyond eBI Product Suite that you use to set up and maintain trading partner profiles and view processed messages. The e*Xchange also processes inbound and outbound messages according to certain eBusiness protocols and your validation Collaborations.

eSecurity Manager (eSM)
An application within the SeeBeyond eBI Product Suite that secures transmission of business-to-business exchanges over public domains such as the Internet.

Event (Message)
Data to be exchanged, either within e*Xchange or between e*Xchange and external systems, which has a defined data structure; for example, a known number of fields, with known characteristics and delimiters. Events are classified by type using Event Type Definitions.

Event Type Definition (ETD)
An Event Type template, defining Event fields, field sequences, and delimiters. Event Type Definitions enable e*Xchange systems to identify and transform Event Types. They are Monk script files with an extension of "**.ssc**," short for message structure script file.

e*Xchange Administrator
An application that allows you to establish user security for e*Xchange Partner Manager (e*Xchange) and e*Insight Business Process Manager (e*Insight).

extended attributes
Information you can store at the company, trading partner, outer envelope, and inner envelope levels, as needed for your business. For companies and trading partners, you can create extended attributes to store specific information about the company or trading partner. For outer and inner envelopes, the extended attributes are specific to a particular eBusiness protocol. Characteristics of ANSI X12 Interchange, Functional Group, and transaction set envelopes are examples of extended attributes you need to enter if you exchange X12 messages with a trading partner. Contrast with *General attributes*.

general attributes
Basic information that identifies companies and trading partners. For inner and outer envelopes, this includes the information you enter for a trading partner profile that is necessary for the exchange of messages but is not specific to a particular eBusiness protocol. The direction of a transmission or the password needed to send messages to an FTP site are examples of general attributes. Contrast with *Extended attributes*.

implementation guide (eBusiness Protocol)
A document, published for a particular electronic message standard by an industry subcommittee, that describes the structure and content of a specific transaction. You can use the Validation Rules Builder to convert electronic versions of ANSI X12 implementation guides to validation Collaborations used by e*Xchange.

inner envelope
An inner envelope definition is a set of parameters and other information you enter about each electronic inner envelope you process with e*Xchange Partner Manager. This definition associates the validation Collaborations that are needed to validate each kind of transaction.

The version number of the eBusiness protocol that applies to the transaction and whether the transaction is transmitted interactively or in batch are examples of inner envelope characteristics.

message tracking attributes
A set of attributes you can define to identify messages stored in the e*Xchange database. Special message tracking extended attributes can be set up and associated with a specific message type (protocol, version, and direction). Examples of attributes that are set up at the message tracking attribute level are Process Instance ID and Activity Instance ID for RosettaNet and FG and TS control numbers for X12.

outer envelope
The trading partner profile component that you use to enter technical information about the exchange of messages between you and your trading partner. The type of eBusiness protocol you agree to use, such as ANSI X12, RosettaNet, or BizTalk, is an example of an outer envelope characteristic.

Partner Manager Envelope Profile
A partner manager envelope profile is a set of default extended attribute values that you define for a trading partner profile component (company, trading partner, outer envelope, or inner envelope).

schema
Files and associated stores created by e*Gate that contain the parameters of all the components that control, route, and transform data as it moves through e*Gate.

Standard Exchange Format (SEF)
The Standard Exchange Format (SEF) is a flat file representation of an EDI implementation guideline. It is a standard that defines how data segments and data elements should be structured so that the message can be understood between trading partners. It also includes validation rules, for example what are the valid values for a data element, or conditions such as if Field A is present then Field B is required.

The purpose of SEF is to put the EDI implementation guidelines in a file in machine readable format so that translators can directly import the file and use the implementation guidelines to translate or map the EDI file. The file can also be used as a means to exchange the implementation guidelines between trading partners, and can

be posted on a public bulletin board or on the company's Web site in the Internet to convey to the public the implementation guidelines used by the company.

The SEF format was developed by Foresight Corporation and is now in the public domain. Programs that can directly import SEF files can save users considerable time in developing new translations or maps.

trading partner component
The trading partner profile component that you use to enter business information about your trading partner. The name of the trading partner, which could be a subdivision of a company, and the people you want to contact are examples of information you enter for a trading partner component.

transaction definition
A set of parameters and other information you enter about each electronic transaction you process with e*Xchange. This definition also associates the validation Collaborations that are needed to validate each kind of transaction.

transaction set
In X12, each business grouping of data is called a transaction set. For example, a group of benefit enrollments sent from a sponsor to a payer is considered a transaction set. Each transaction set contains groups of logically related data in units called segments. For example, the N4 segment conveys the city, state, ZIP code, and other geographic information.

A transaction set contains multiple segments, so the addresses of the different parties, for example, can be conveyed from one computer to the other. An analogy would be that the transaction set is like a freight train; the segments are like the train's cars, and each segment can contain several data elements in the same way that a train car can hold multiple crates.

Specifically, in X12, the transaction set is comprised of segments ST through SE.

transaction type
The kind of eBusiness protocol you agree to use to exchange data and information with a particular trading partner. For example, ANSI X12 and RosettaNet are two different transaction types.

user group
User groups allow you to grant access permissions to a set of users with similar processing needs without having to specify individual privileges for each user. For example, the User Administrator can set up a group for users who need full access to a specific trading partner profile, but who should not be able to view information about any other profile. The User Administrator assigns each user that meets this criterion to a particular user group. Then, your eX Administrator (or another user who has been granted appropriate privileges) grants access privileges to this user group so that all members of the group can view and modify the desired information.

validation Collaboration

A Collaboration that you create to define the syntax and validate the content of electronic business-to-business (B2B) messages. One validation Collaboration is required for each type of electronic transaction to be processed by e*Xchange. You can use the Validation Rules Builder to automatically generate a validation Collaboration for a specific kind of X12 transaction, according to specific implementation guidelines.

Validation Rules Builder

An e*Xchange Partner Manager tool for converting electronic EDI implementation guides into files that are compatible for use with e*Xchange. This conversion tool accepts Standard Exchange Format (SEF) version 1.4 or 1.5 files and converts then into e*Gate Integrator Event Type Definition (ETD) and Collaboration Rules files.

XML

Extensible Markup Language. XML is a language that is used in Events or messages in e*Insight Business Process Manager, containing structured information. XML is different from String in that XML messages can contain both content, and information about the content.

Index

A

B

C

E

F

G

H

I

J

M

N

O

P

R

S

T

U

V

X