SeeBeyond ICAN Suite

e*Way Intelligent Adapter for Odette FTP User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Version 20050406012454.

Contents

Chapter 1

Introduction 9

Odette FTP e*Way: Overview 9

Intended Reader 9

General e*Way Operation 10

e*Way and ODEX Application 10

Purpose of the e*Way 10

Product-specific Implementation 10

e*Way Basic Functions 11

e*Way Configuration 11

e*Way Design Overview 11

ODEX Components 13

Batch Operation 14

ETD Configuration and ODEX Operation 14

ODEX Products 14

e*Way Components 16

Supported Operating Systems 16

System Requirements 16

External System Requirements 16

Chapter 2

Installation 18

Windows Systems 18

Pre-installation 18

e*Way Installation Procedure 18

UNIX Systems 19

Pre-installation 19

Installation Procedure 19

Files/Directories Created by the Installation 20

Chapter 3

Multi-Mode e*Way Configuration 21

Multi-Mode e*Way Properties 21

JVM Settings 22
- JNI DLL Absolute Pathname 22
- CLASSPATH Prepend 22
- CLASSPATH Override 23
- CLASSPATH Append From Environment Variable 23
- Initial Heap Size 23
- Maximum Heap Size 24
- Maximum Stack Size for Native Threads 24
- Maximum Stack Size for JVM Threads 24
- Disable JIT 24
- Remote debugging port number 24
- Suspend Option for Debugging 25

General Settings 25
- Rollback Wait Interval 25
- Standard IQ FIFO 25

Chapter 4

e*Way Connection Configuration 26

Configuring e*Way Connections 26

Configuration Parameters 27
- Connector 27
  - Type 27
  - Class 27
  - Property.Tag 27
- Odette FTP ETD Configuration 28
  - ODEX 28
  - ODEX Command Processor 28
  - Batch File 28
  - Outbound Box 28
  - To Partner 29
  - Orig Local Node 29
  - Outbound File 29
  - Priority 29
  - Format 30
  - Record Size 30
  - Outbound Virtual File 30
  - Inbound Box 30
  - From Partner 30
  - Inbound File 31
  - Inbound Virtual File 31
  - UserData 31
  - Log Response 31
  - Temp Directory 32

Max DataIn Size 32
Max DataOut Size 32
Return Code Configuration 32
Success 33
Error 33
FatalError 33
NotAttempted 33
Terminated 34

Chapter 5

Implementation 35

Implementation: Introduction 35

Sample Implementations 35
ODEX Configuration for Examples 35
ETD Configuration 36
Schema Example 1 37
Schema Example 2 40
Schema Example 3 42
Schema Example 4 44
Schema Example 5 48

Schema Sample 49
Schema Setup and Components 50
Sample Schema: General Implementation Steps 50
Before Schema Implementation 51
Importing a Schema 51
Running a Schema 51

ODEX ETD Structure 52
ETD Exposed Methods 53
ETD Attributes 54
ODEX Last Reason Codes 60
ODEX File Status Flags 61
ODEX ETD and ODEX Batch Jobs 61

Collaboration Implementation 62
Example A: Send and Receive Payload 62
Simple Cases 62
Complex Cases 62
Example B: Using the Outbound Box 63

ODEX Configuration 63
User Directory Configuration 64
ODEX Plus 64
ODEX Professional 66
Basic ODEX Configurations 67
Communication Configuration 68
OFTP Configuration 68
Additional Configuration 69
EERP Handling 69

Recovery, Data Integrity, and Security 69

Chapter 6

e*Way Java Methods 71

Odette FTP e*Way Methods and Classes: Overview 71

ODEXConfig Class 72

getSendFiles 74
getSendFileCount 74
getRecvFiles 75
getRecvFileCount 75
restoreDefault 76
getWorkFile 76
setWorkFile 76
getOutboundFile 77
setOutboundFile 77
getOutboundVFN 78
setOutboundVFN 78
getOutboundBox 79
setOutboundBox 79
getToWhom 79
setToWhom 80
getOrigLocalNode 80
setOrigLocalNode 81
getEarlyDate 81
setEarlyDate 82
getEarlyTime 82
setEarlyTime 82
getFormat 83
setFormat 83
getRecSize 84
setRecSize 84
getPriority 85
setPriority 85
getUserData 86
setUserData 86
getMsgType 86
setMsgType 87
getFileType 87
setFileType 88
getBatchFile 88
setBatchFile 88
getNew 89
setNew 89
getOld 90
setOld 90
getInboundFile 91
setInboundFile 91
getInboundVFN 91
setInboundVFN 92
getInboundBox 92
setInboundBox 93
getFromWhom 93
setFromWhom 94
getVirtDate 94
setVirtDate 94
getVirtTime 95
setVirtTime 95
getReturnCode 96
getSuccess 96
getTerminated 97
getError 97

getFatalError 98
getNotAttempted 98

ODEXETD Class 99

initialize 100
reset 100
terminate 101
getBatchParameters 101
getBatchStatus 102
getDataOut 102
setDataOut 103
getSendFileCount 103
getRecvFileCount 103
getSendFiles 104
getRecvFiles 104
getDataIn 105
setDataIn 105
invokeBatch 106
makeCall 106
querySent 107
schFILE 108
schODETT 108
schData 109
schODETTData 109
queryRecv 110
rcvFILE 110
rcvODETT 111
rcvData 111
rcvODETTData 112

ODEXRecvFileDesc Class 112

getTRDName 113
getLocalCode 113
getVirtFile 114
getTryCount 114
getLocalFile 115
getWorkFile 115
getUserData 115
getMsgType 116
getEERPSentTime 116
getLocalFileTime 117
getFileRecvTime 117
getFileRecv 117
getEERPSent 118
getRetriable 118
getStatus 119
getLastReasonCode 119
getLastReasonTxt 119
getAttempted 120
getFailed 120
getProcessing 121
getProcessed 121
getForward 121

ODEXSendFileDesc Class 122

getTRDName 123
getLocalCode 123
getVirtFile 123
getLocalFile 124
getWorkFile 124
getTryCount 125
getUserData 125
getMsgType 125
getEERPRecvTime 126
getFileSentTime 126

getLocalFileTime 127
getFileSent 127
getEERPRecv 127
getRetriable 128
getStatus 128
getLastReasonCode 129
getLastReasonTxt 129
getAttempted 129
getFailed 130
getProcessing 130
getProcessed 131
getForward 131

Index 132

Chapter 1

Introduction

This document explains how to install, set up, and configure the SeeBeyond™ Technology Corporation's (SeeBeyond™) e*Way™ Intelligent Adapter for Odette FTP. This chapter provides an introduction to the e*Way.

1.1 Odette FTP e*Way: Overview

This guide explains how to configure and operate the Odette FTP e*Way. It also explains how to implement the e*Way in a sample SeeBeyond e*Gate Integrator environment. The rest of this section provides an overview of the e*Way and how it operates.

Note: *Odette stands for Organization for Data Exchange by Tele-transmission in Europe.*

e*Way Basic Functions

The basic functions of the e*Way are:

- Handle and operate data communication with ODEX
- Interface with a trading partner

1.2 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have high-level knowledge of Windows and/or UNIX operations and administration; to be thoroughly familiar with the ODEX application (see **"External System Requirements" on page 16**); and to be thoroughly familiar with Windows-style GUI operations.

1.3 General e*Way Operation

The Odette FTP e*Way enables the e*Gate system to exchange data with any system using the Odette FTP (OFTP) communication protocol. The e*Way does this operation via a third-party application, ODEX (see **"External System Requirements" on page 16**).

This section explains the e*Way's general operation and how it interfaces with ODEX.

1.3.1 e*Way and ODEX Application

To communicate with OFTP, the Odette FTP e*Way uses ODEX, a third-party stand-alone application. The e*Way only launches and monitors the application, while ODEX does the actual OFTP file transfer.

ODEX is an end-user oriented software with graphical user interfaces (GUIs) for trading-partner configuration, electronic data interchange (EDI) job configuration, scheduling (automation), and data communication. ODEX also provides a batch interface allowing you to do these tasks using a batch program.

Purpose of the e*Way

The basic purpose of the Odette FTP e*Way is to transfer files over OFTP using an Event Type Definition (ETD) specialized for this e*Way. This ETD is called the ODEX ETD and is based on the e*Gate ETD file **oftp.xsc**. Using this ETD, you can create e*Gate Collaboration Rules to customize the business logic and data transfers required by your system.

The ODEX ETD's configuration allows you to control basic data-exchange operations in ODEX. The ETD's implementation is based on the ODEX batch interface.

Note: *For more information on ETDs and their use in e*Gate, see the* ***e*Gate Integrator User's Guide****.*

Product-specific Implementation

Because the e*Way operates in conjunction with ODEX, its implementation is product-specific. Correct implementation of this e*Way assumes that ODEX is properly installed and configured on at least one e*Gate Participating Host. The e*Way cannot configure ODEX. It can only keep necessary information available to communicate with and control the third-party application.

Note: *For complete details on how to set up and configure ODEX, see the appropriate documentation for that product.*

e*Way Communication with ODEX

Communication between the Odette FTP e*Way's processes (e*Gate Collaboration thread) and ODEX processes is needed so the e*Way can operate ODEX.

Note: *For more information on Collaborations and data flow in e*Gate, see the* ***e*Gate Integrator User's Guide****.*

e*Way Basic Functions

The basic functions of the e*Way are:

- Handle and operate data communication with ODEX
- Interface with trading partners

e*Way Configuration

The configuration parameters for the Odette FTP e*Way, for convenience, allow you to set necessary ODEX parameters of operation. In turn, these e*Way parameters are adopted into the ODEX ETD's configuration.

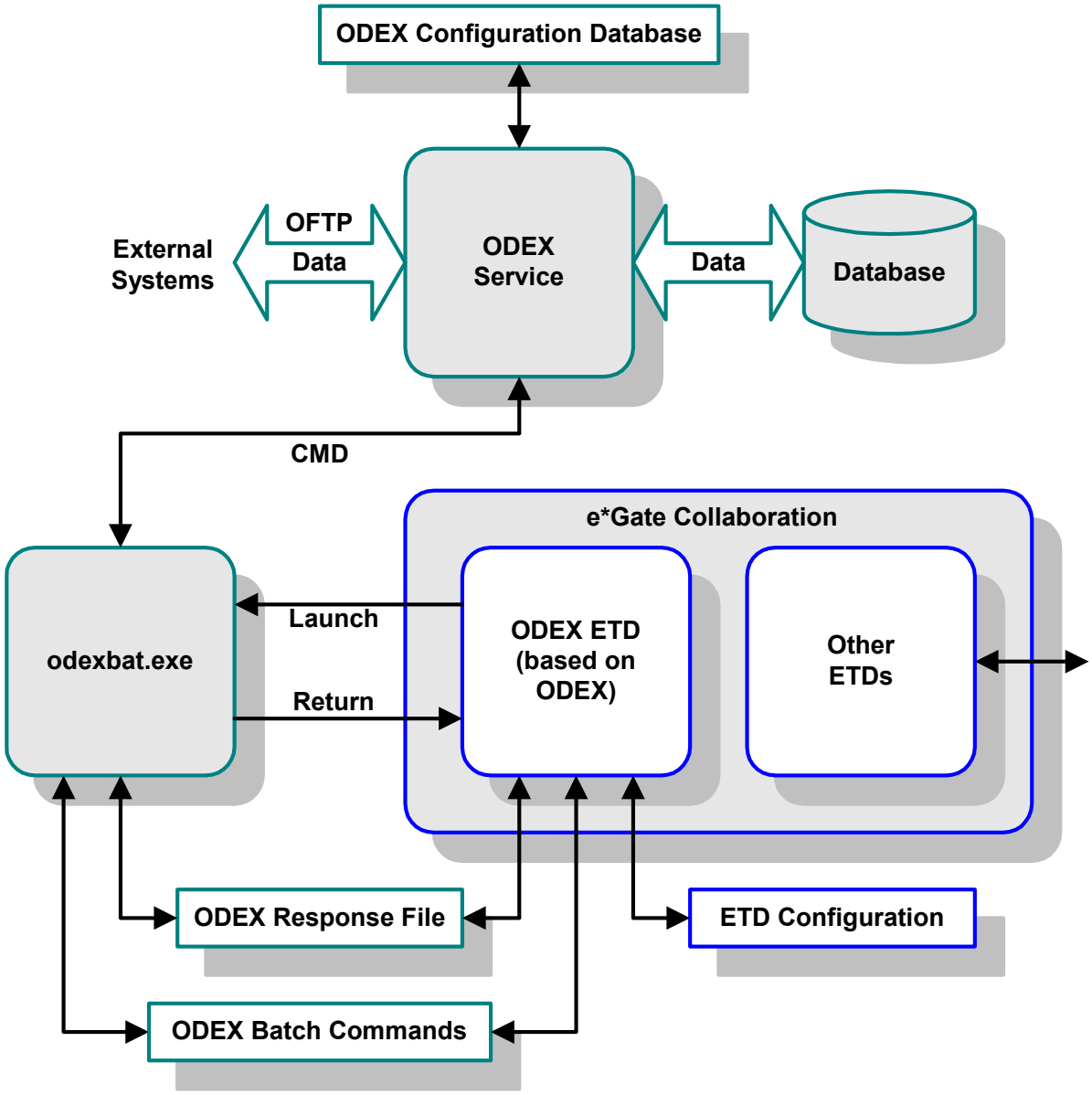
When you configure the Odette FTP e*Way, its set configuration parameters must match the corresponding ODEX configurations. If the e*Way and ODEX are sending and expecting different sets of instructions, communication cannot take place between the two.

See **Chapter 4** for details on how to set the Odette FTP e*Way configuration parameters.

e*Way Design Overview

Figure 1 on page 12 shows a general diagram of how the Odette FTP e*Way, the ODEX ETD, and the ODEX application operate in supporting one another.

Figure 1 Odette FTP e*Way/ODEX Operation



The advantages of this setup are:

- An easy-to-use e*Way
- Ability to operate ODEX using the e*Way
- The robustness and versatility already available with ODEX

1.3.2 ODEX Components

Figure 1 shows an architectural overview of the Odette FTP e*Way as it interacts with the following basic ODEX components:

- **Virtual File (VF):** The data transfer unit in OFTP. Before the e*Way can extract a specific VF, it must know the *virtual file name* (VFN) to look for. In turn, the VFN is mapped to a local file (EDI or non-EDI) and pointed to by a full path in a Participating Host's system.
- **Data:** The data sent or received via ODEX and can be an EDI file (for example, EDIFACT) or non-EDI file (for example, image or binary).

Note: *EDI files contain specific routing information in a "Service" section, but non-EDI files do not contain such information.*

- **ODEX server:** The ODEX engine that runs on the Participating Host and handles the actual data interchange jobs over OFTP.
- **odexbat.exe file:** The ODEX batch processor that communicates with the ODEX server to carry out a specific data transfer, that is, queuing a file to be sent to a destination or extracting a file from ODEX's received file repository.
- **Batch file**: A text file containing ODEX batch commands. This text syntax for this file is based on the ODEX batch language. See the appropriate ODEX user's guides for details.
- **ODEX configuration database:** The database used by ODEX to store metadata. It is not open to another application. On Windows, Microsoft Access 2000 is used as the repository, and on UNIX, an ISAM file is used.
- **ODEX response file:** A log file containing information on the ODEX batch file just executed. This file is overwritten each time a batch file operation (batch job) is performed. The response file is parsed by the ETD for detailed invoke information.
- **ODEX batch launch:** The startup of a batch job. The ETD implementation has methods to launch the **odexbat.exe** file. The Collaboration calls the methods on the ETD instance to launch an EDI job.
- **ODEX batch return:** Checking the status of a launched job. The ETD implementation has methods that check the return code and expose it to the Collaboration. The Collaboration then checks the status of the EDI job that has just been launched.

You can use the previous list as a convenient glossary of ODEX terminology. For more information, see the appropriate ODEX user's guides.

Batch Operation

Every data transfer job is initiated by a batch command file. Even though a batch command is called, for example, **SNDFILE** (see **"ETD Configuration and ODEX Operation" on page 14**), the actual OFTP data exchange is not done by the batch execution directly. Instead, **odexbat.exe** executes the batch file, which only delivers a job to the ODEX server. The ODEX server then carries out the data exchange asynchronously.

1.3.3 ETD Configuration and ODEX Operation

The ODEX ETD contains methods that allow you to control the batch operations of the ODEX application. By using the e*Gate Schema Designer's ETD Editor GUI to drag and drop the desired methods into the ETD node structure, you can call specific ODEX batch functions, as you wish.

The ODEX ETD exposes methods that call the following ODEX batch commands:

- **SNDFILE**: Hands over a local file to the ODEX server, to be sent to a destination, for example, to a trading partner. You must configure the desired destination at the ODEX server by using the ODEX configuration GUIs. When the **SNDFILE** command finishes, ODEX has taken the request, put this job in its queue, and copied the data into its repository. The actual data exchange happens asynchronously at the time the job's batch file executes.
- **SNDODETT**: The same as **SNDFILE**, but the file sent is an EDI-formatted file instead of a non-EDI file.
- **RCVFILE**: Extracts from the ODEX server's received repository, a file that meets certain specified criteria.
- **RCVODETT**: The same as **RCVFILE**, but the file is an EDI-formatted file instead of a non-EDI file.

For a detailed explanation of the Odette FTP e*Way methods related to the previous list, see **Chapter 6**. For an explanation of the ODEX ETD, see **"ODEX ETD Structure" on page 52**.

1.3.4 ODEX Products

ODEX application has two available products, ODEX Plus (UNIX) and ODEX Professional (Windows). These products differ somewhat in terms of their available features and software operation. For a complete explanation of the differences, see the appropriate ODEX user's guides.

Hardware Configuration

Figure 2 on page 15 shows a diagram of how ODEX Professional and its **odexbat.exe** (ODEX batch processor) can run on different machines.

Figure 2 ODEX Professional and Batch Processor

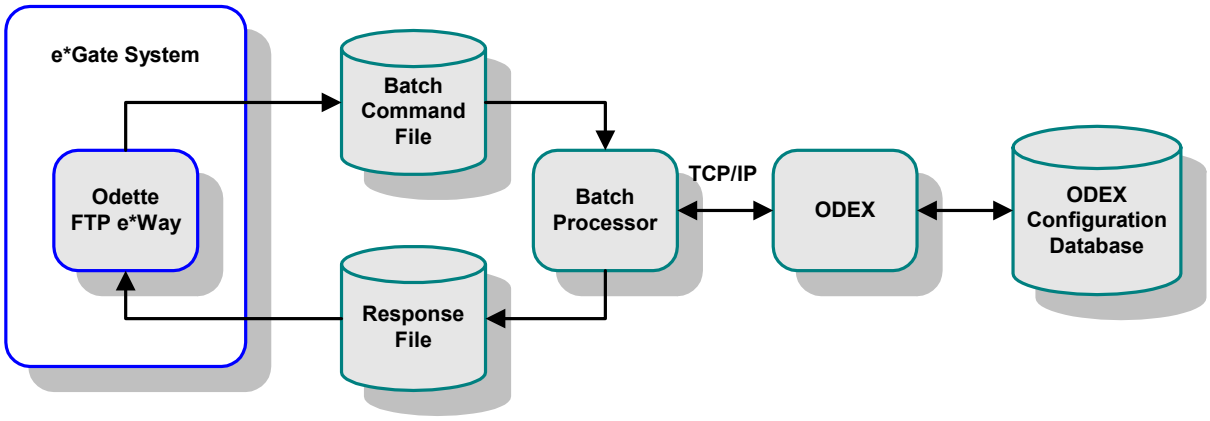
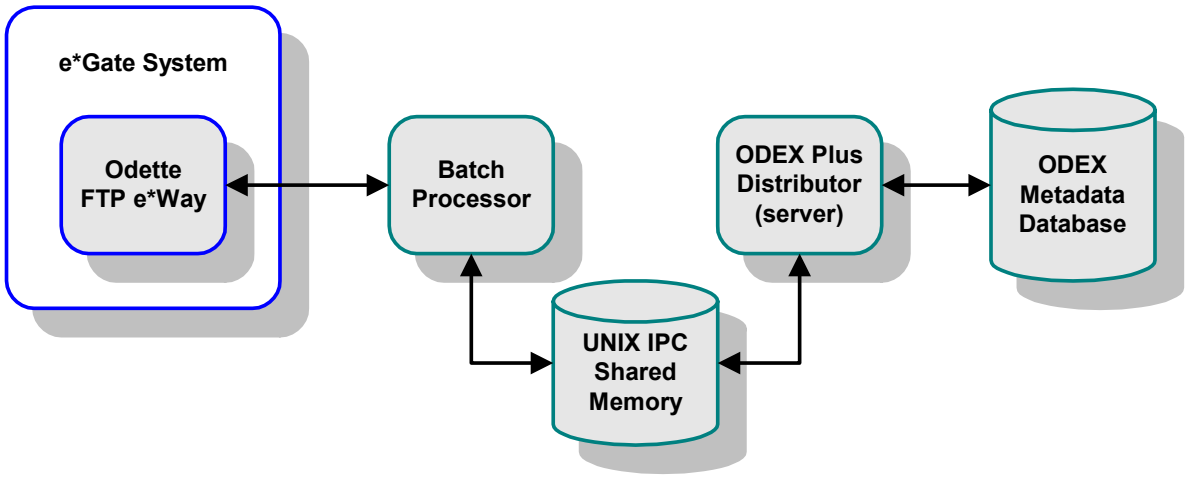


Figure 3 shows a diagram of ODEX Plus and **odexbat.exe** running on the same machine.

Figure 3 ODEX Plus and Batch Processor



In ODEX Professional: Its batch processor interacts with ODEX over TCP/IP. The Odette FTP e*Way communicates with the **odexbat.exe** via a batch command file and a response file.

Note that there is a difference between the batch interface of ODEX Professional and that of ODEX Plus. The batch processor in ODEX Professional runs as a TCP/IP client and talks to ODEX as a TCP/IP server. This setup allows the **odexbat.exe** file to run on other machines besides the machine where ODEX is running.

In ODEX Plus: ODEX and its batch processor (**odexbat.exe**) must run on the same machine.

1.3.5 e*Way Components

The Odette FTP e*Way is made up of the following components:

- Uses the Multi-Mode e*Way and its executable file, **stcew.exe** (see **Chapter 3**).
- Configuration files, which the e*Gate Schema Designer's e*Way Editor GUI uses to define configuration parameters
- Additional files necessary for operation, as shown in **Table 1 on page 20** (shows a complete list of installed files).

1.4 Supported Operating Systems

The Odette FTP e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP-UX 11.0 and 11i (PA-RISC)
- IBM AIX 5.1L and 5.2
- Sun Solaris 8 and 9

1.5 System Requirements

To use the Odette FTP e*Way, you need to meet the following requirements:

- An e*Gate Participating Host
- A TCP/IP network connection

The e*Way must be configured and administered using the e*Gate Schema Designer graphical user interface (GUI).

Note: *Additional disk space can be required to process and queue the data that this e*Way processes. The amount necessary can vary based on the type and size of the data being processed and any external applications doing the processing.*

1.6 External System Requirements

The Odette FTP e*Way supports Data Interchange's ODEX high-speed asynchronous communications software application. The e*Way uses this application to interface with OFTP. To use the e*Way, you must have ODEX installed on an e*Gate Participating Host with the e*Way (see **"ODEX Products" on page 14**). ODEX must also be operational and accessible to the e*Way.

You can also visit Data Interchange's Web site at the following URL:

http://www.dip.co.uk/Products/ODEX/

Note: *Internet links may change after the date of publication. If Data Interchange has changed this URL, you can search their Web site for the appropriate Web pages.*

As explained earlier in this chapter, you can use either of two available ODEX products with the e*Way. These products are:

- **ODEX Plus (version 2.7.2):** UNIX
- **ODEX Professional (version 3.2):** Windows

See the appropriate ODEX user's guides for details on product differences, as well as explanations of system requirements, operation, and procedures on how to install and operate this application. The e*Way interfaces with both ODEX products via the applications' batch interfaces.

Installing ODEX

See the appropriate ODEX user's guides for complete information on how to install ODEX. Be sure to correctly configure ODEX to meet your communication needs then configure the Odette FTP e*Way to conform to your ODEX parameters.

It is helpful to keep a record of your ODEX configuration information. Enter configuration parameters for the Odette FTP e*Way according to your ODEX configuration.

Note: *For information on how to configure ODEX with the e*Way, see* **"ODEX Configuration" on page 63***. For details on how to configure the e*Way Connection for the Odette FTP e*Way, see* **Chapter 4***.*

Chapter 2

Installation

This chapter explains how to install the e*Way Intelligent Adapter for Odette FTP.

2.1 Windows Systems

This section explains how to install the Odette FTP e*Way with a Windows platform.

2.1.1 Pre-installation

- Exit all Windows programs before running the setup program, including any anti-virus applications.
- You must have Administrator privileges to install this e*Way.

2.1.2 e*Way Installation Procedure

To install the Odette FTP e*Way on Windows Systems

1. Log in as an Administrator on the workstation where you want to install the e*Way.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's **Auto-run** feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the **setup.exe** file on the CD-ROM drive.
4. After the **InstallShield** setup application launches, follow the on-screen instructions to install the e*Way.

Be sure to install the e*Way files in the suggested **\client** installation directory. The installation utility detects and suggests the appropriate installation directory.

Caution: *Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.*

Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions.

2.2 UNIX Systems

This section explains how to install the Odette FTP e*Way with a UNIX platform.

2.2.1 Pre-installation

You do not require root privileges to install this e*Way. Log in under your name with which you wish to own the e*Way files. Be sure that this user has sufficient privileges to create files in the e*Gate directory tree.

2.2.2 Installation Procedure

To install the Odette FTP e*Way on a UNIX system

1. Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
2. If necessary, mount the CD-ROM drive.
3. At the shell prompt, type:

   cd /cdrom/setup
4. Start the installation script by typing:

   setup.sh
5. A menu of options appears. Select the **e*Gate Add-on Applications** option. Then, follow any additional on-screen directions.

Be sure to install the e*Way files in the suggested **\client** installation directory. The installation utility detects and suggests the appropriate installation directory.

Caution: *Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.*

Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions.

2.3 Files/Directories Created by the Installation

Whether for Windows or UNIX, the Odette FTP e*Way installation installs the files shown in Table 1 within the e*Gate directory tree. Files are installed within the **eGate\client** directory on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files Created by Installation

e*Gate Directories	Files
\classes	oftp.jar
\etd	oftp.ctl
\etd\oftp	oftp.xsc
\configs\oftp	oftp.def

Chapter 3

Multi-Mode e*Way Configuration

This chapter describes how to configure the e*Gate Integrator's Multi-Mode e*Way Intelligent Adapter.

3.1 Multi-Mode e*Way Properties

Set the Multi-Mode e*Way properties using the e*Gate Schema Designer.

To set properties for a new Multi-Mode e*Way

1. Select the Navigator pane's Components tab in the Main window of the Schema Designer.
2. Open the host and Control Broker where you want to create the e*Way.
3. On the Palette, click on the icon to create a new e*Way.
4. Enter the name of the new e*Way, then click **OK**.
5. Select the new component, then click the Properties icon to edit its properties.

   The **e*Way Properties** dialog box opens
6. Click **Find** beneath the **Executable File** field, and select an executable file (**stceway.exe** is located in the **bin** directory).
7. Under the **Configuration File** field, click **New**.

   The e*Way Configuration Editor window opens.
8. When the **Settings** page opens, set the configuration parameters for this e*Way's configuration file (see **"JVM Settings" on page 22** and **"General Settings" on page 25** for details).
9. After selecting the desired parameters, click **Save** on the **File** menu to save the configuration (**.cfg**) file.
10. Close the **.cfg** file and e*Way Configuration Editor.
11. Set the properties for the e*Way in the **e*Way Properties** dialog box.
12. Click **OK** to close the dialog box and save the properties.

3.2 JVM Settings

To correctly configure the e*Way Intelligent Adapter for Odette FTP, you must configure the Java Virtual Machine (JVM) settings. This section explains the configuration parameters in the e*Way Configuration Editor window, which control these settings.

JNI DLL Absolute Pathname

Description

Specifies the absolute path name to where the JNI **.dll** (Windows) or shared library (UNIX) file is installed by the *Java 2 SDK*, on the Participating Host.

Required Values

A valid path name.

Additional Information

The JNI **.dll** or shared library file name varies, depending on the current operating system (OS). The following table lists the file names by OS:

Operating System	Java 2 JNI .dll or Shared Library Name
Windows systems	jvm.dll
AIX	libjvm.a

The value assigned can contain a reference to an environment variable, by enclosing the variable name within a pair of "%" symbols, for example:

```
%MY_JNIDLL%
```

Such variables can be used when multiple Participating Hosts are used on different OS/platforms.

Caution: *To ensure that the JNI **.dll** file loads successfully, the Dynamic Load Library search path environment variable must be set appropriately to include all the directories under the Java 2 SDK (or JDK) installation directory, which contain shared library or **.dll** files.*

CLASSPATH Prepend

Description

Specifies the paths to be prepended to the CLASSPATH environment variable for the JVM.

Required Values

An absolute path or an environmental variable. This parameter is optional.

Additional Information

If left unset, no paths are prepended to the CLASSPATH environment variable.

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of "%" symbols, for example:

```
%MY_PRECLASSPATH%
```

CLASSPATH Override

Description

Specifies the complete CLASSPATH variable to be used by the JVM. This parameter is optional. If left unset, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) is set.

Note: *All necessary* ***.jar*** *and* ***.zip*** *files needed by both e*Gate and the JVM must be included. It is advised that the* ***CLASSPATH Prepend*** *parameter should be used.*

Required Values

An absolute path or an environment variable. This parameter is optional.

Additional Information

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of "%" symbols, for example:

```
%MY_CLASSPATH%
```

CLASSPATH Append From Environment Variable

Description

Specifies whether the path is appended for the CLASSPATH environmental variable to **.jar** and **.zip** files needed by the JVM.

Required Values

YES or **NO**. The configured default is **YES**.

Initial Heap Size

Description

Specifies the value for the initial heap size in bytes. If this value is set to 0 (zero), the preferred value for the initial heap size of the JVM is used.

Required Values

An integer from **0** to **2147483647**. This parameter is optional.

Maximum Heap Size

Description

Specifies the value of the maximum heap size in bytes. If this value is set to 0 (zero), the preferred value for the maximum heap size of the JVM is used.

Required Values

An integer from **0** to **2147483647**. This parameter is optional.

Maximum Stack Size for Native Threads

Description

Specifies the value of the maximum stack size in bytes for native threads. If this value is set to 0 (zero), the default value is used.

Required Values

An integer from **0** to **2147483647**. This parameter is optional.

Maximum Stack Size for JVM Threads

Description

Specifies the value of the maximum stack size in bytes for JVM threads. If this value is set to 0 (zero), the preferred value for the maximum heap size of the JVM is used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Disable JIT

Description

Specifies whether the Just-In-Time (JIT) compiler is disabled.

Required Values

YES or **NO**.

Remote debugging port number

Description

Specifies whether to allow remote debugging of the JVM.

Required Values

YES or **NO**.

Suspend Option for Debugging

Description

Indicates whether to suspend the Option for Debugging on JVM startup.

Required Values

YES or **NO**.

3.3 General Settings

This section contains the parameters for rollback wait and IQ messaging priority.

Note: *For more information on the* ***General Settings*** *configuration parameters see the* ***e*Gate Integrator User's Guide****.*

3.3.1 Rollback Wait Interval

Description

Specifies the time interval to wait before rolling back the transaction.

Required Values

A number within the range of 0 to 99999999, representing the time interval in milliseconds.

3.3.2 Standard IQ FIFO

Description

Specifies whether the highest priority messages from all SeeBeyond Standard IQs are delivered in the first-in-first-out (FIFO) order.

Required Values

Select **Yes** or **No**. **Yes** indicates that the e*Way retrieves messages from all SeeBeyond Standard IQs in the first-in-first-out (FIFO) order. **No** indicates that this feature is disabled; **No** is the default.

Chapter 4

e*Way Connection Configuration

This chapter explains how to configure e*Way Connections for the e*Way Intelligent Adapter for Odette FTP.

4.1 Configuring e*Way Connections

Set up e*Way Connections using the e*Gate Schema Designer graphical user interface (GUI).

To create and configure e*Way Connections

1. In the Schema Designer's **Navigation** pane**,** select the **Component** tab.
2. Select the **e*Way Connections** folder.
3. On the palette, click on the icon to create a new **e*Way Connection**.

   The **New e*Way Connection Component** dialog box appears.
4. Enter a name for the **e*Way Connection**, then click **OK**.

   An icon for your new e*Way Connection appears in the Navigation pane.
5. Double-click on the new **e*Way Connection** icon.

   The **e*Way Connection Properties** dialog box appears.
6. From the **e*Way Connection Type** drop-down box, select **Odette FTP**.
7. From the **e*Way Connection Configuration File**, click **New** to open the e*Way Editor GUI.

Note: *To use an existing file, click* ***Find****.*

8. Use the e*Way Editor to create a new configuration file for this e*Way Connection. Do this operation by selecting the appropriate configuration parameters available in the GUI.
9. When you are finished, close the e*Way Editor and save the new configuration file.

Note: *It is recommended that you configure one e*Way Connection for each e*Way component. Avoid sharing one e*Way Connection among different e*Ways.*

The rest of this chapter explains the Odette FTP e*Way Connection configuration parameters as follows:

- **"Connector" on page 27**
- **"Odette FTP ETD Configuration" on page 28**
- **"Return Code Configuration" on page 32**

4.2 Configuration Parameters

This section explains the configuration parameters for the Odette FTP e*Way Connection.

4.2.1 Connector

The parameters in the **Connector** section allow the Collaboration engine to identify the e*Way Connection.

Type

Description

Specifies the type of e*Way Connection.

Required Values

ODEX. The value defaults to **ODEX**.

Class

Description

Specifies the class name of the ODEX connector object.

Required Values

A valid package name. The default is **com.stc.eways.oftp.ODEXConnector**.

Property.Tag

Description

Identifies the data source. This parameter is required by the current **EBobConnectorFactory**.

Required Values

A valid data source package name.

4.2.2 Odette FTP ETD Configuration

The parameters in the **Odette FTP ETD Configuration** section allow you to configure characteristics of the ODEX Event Type Definition (ETD) file (based on **oftp.xsc**), that is, the default values needed when scheduling an outgoing file to ODEX or extracting an incoming file from ODEX.

ODEX

Description

This e*Way is based on the ODEX batch interface. There are slight batch syntax differences between the two ODEX products, ODEX Professional (Windows) and ODEX Plus (UNIX). You need to indicate which you are using.

Required Values

ODEX Plus or ODEX Professional (default).

ODEX Command Processor

Description

The full path on a Participating Host, pointing to the **odexbat.exe** executable file (ODEX batch processor).

Required Values

Valid path location on a Participating Host.

Batch File

Description

The full path pointing to the **odexbat.exe** file that can also be invoked by the ETD method **invokeBatch**. For more information on the e*Way's methods, see **Chapter 6**.

Required Values

Optional; if used, enter the valid path location of the **odexbat.exe** file, on a Participating Host.

Outbound Box

Description

The full path on a local system where all outgoing files can be located by any ODEX ETD and its Collaboration. This is path for a local directory where outbound files to be scheduled for transmission by ODEX are located. In ODEX, this source file is referred to as the **PCFile** in the **SNDFILE** batch command. See the appropriate ODEX user's guide for details.

Required Values

A valid path location on a local system.

To Partner

Description

An ID for one of your trading partners to whom you sent data. This ID must be a preconfigured trading partner in the ODEX User Directory. The ID must correspond to a **Local Code** in ODEX. This ID is used to refer to a trading partner in your Collaboration and never goes out to your trading partner in any exchanged message. See the appropriate ODEX user's guides for a detailed description of the **Local Code**.

Required Values

A valid ODEX **Local Code** ID.

Orig Local Node

Description

The **Local Code** ID for the **FileNode** level originator in the ODEX ETD. When a file is sent, the originator can be identified using the ETD's **Internal Node** information (see **"User Directory Configuration" on page 64**). Your internal node is your own electronic data interchange (EDI) code.

However, for large organizations, departmental or sectional information may be needed to further identify the originator of a file. See the appropriate ODEX user's guides for more information.

Required Values

A valid ODEX **Local Code** ID.

Outbound File

Description

The name of a file outbound to ODEX (to be scheduled by ODEX to send out to a trading partner). Note that it is a file name only. It does not include the path name of the parent directory. The parent directory must be given in the **Outbound Box** parameter.

Required Values

A valid file name.

Priority

Description

An ODEX priority code used by ODEX to prioritize a file for sending. The codes range from 1 to 9, with 1 as the highest priority.

Required Values

An integer from 1 to 9.

Format

Description

The format of a file to be sent. The file can contain unformatted, text, fixed-length, or variable-length records.

Required Values

U for unformatted, **T** for text, **F** for fixed length, and **V** for variable length.

Record Size

Description

If the format is fixed length (**F**), then this parameter gives the record size of the file transferred.

Required Values

An integer from 0 to 99999; optional, only use this parameter for fixed-length files.

Outbound Virtual File

Description

The default OFTP virtual file (VF) name for an outgoing file. This file name is usually agreed upon between trading partners. For example, **INVOICE101** could be the file name your trading partner expects for a specific type of invoice data-bearing file. The names for virtual files must conform to OFTP specifications.

Note: *See OFTP specifications for information on naming conventions for virtual files.*

Required Values

A valid OFTP virtual file name.

Inbound Box

Description

A directory (full path) on a Participating Host, where all incoming files are stored by the file-extracting operation by a Collaboration.

Required Values

A valid directory on an e*Gate Participating Host.

From Partner

Description

The **Local Code** from your ODEX User Directory (see **"User Directory Configuration" on page 64**), for a trading partner from whom you receive data. This **Local Code** ID must correspond to a preconfigured trading partner in your ODEX User Directory. See the appropriate ODEX user's guides for a detailed explanation of the **Local Code**.

Required Values

A valid ODEX **Local Code**.

Inbound File

Description

The name of a file inbound from ODEX. Note that it is a file name only. It does not include the path name of the parent directory. The parent directory must be given in the **Inbound Box** parameter.

Required Values

A valid file name.

Inbound Virtual File

Description

An OFTP virtual file (VF) name associated with a file received from a trading partner. It is also the default value of the corresponding ETD node, **InboundVFN**. The Collaboration uses this name to extract the appropriate inbound file (incoming from ODEX).

Required Values

A valid file name.

UserData

Description

Refers to **OFTP UserData**, a character string whose specific usage is agreed upon between you and your trading partner. See the appropriate ODEX user's guide for details.

Required Values

An eight-character string; optional.

Log Response

Description

Indicates whether the response file is added to the e*Way's component log file in e*Gate. If it is set to **Yes**, the response file information is added, if set to **No**, the file information for each batch execution is overwritten.

Note: *You must turn on the TRACE flag for the e*Way to be sure the* ***odexbat.exe*** *response file is added to the e*Way's log file.*

Required Values

Yes or **No**. The default is **Yes**.

Temp Directory

Description

A directory (full path) location on a Participating Host. This directory is used for temporarily depositing intermediate files, such as the working file for holding the contents of the payload nodes in the ETD.

Required Values

A valid directory on a Participating Host. You must create this directory before configuring the e*Way.

Max DataIn Size

Description

The maximum size for incoming data, that is, the inbound payload node (**DataIn**) in the ETD. This node is used to hold the extracted inbound file from ODEX. The node can be copied to another destination, for example, to an Intelligent Queue (IQ).

This parameter is a safeguard to prevent a huge payload from using up too much memory.

Required Values

An integer from 1024 to 864000.

Max DataOut Size

Description

The maximum size for outgoing data, that is, the outbound payload node (**DataOut**) in the ETD. This node is used to hold the data routed into a Collaboration, for example, from a subscribed topic. Its contents is usually placed in a local file, and the ODEX batch interface is used to schedule the file to ODEX.

This is a safeguard to prevent huge payload using up memory.

Required Values

An integer from 1024 to 864000.

4.2.3 Return Code Configuration

The parameters in the **Connector** section allow you to set the numbers for commonly used conditions that **odexbat.exe** returns to the calling application.

Table 2 shows the most common return conditions and the codes ODEX batch processor (**odexbat.exe**) uses for these conditions. The e*Way configuration parameters default to these codes.

Table 2 ODEX Batch Processor Return Codes

Condition	Return Code
Success	0
Error	4
FatalError	8
NotAttempted	12
Terminated	1

Success

Description

The return code indicating that the batch command has been executed successfully.

Required Values

0 (the default).

Error

Description

The return code indicating that the batch command has been executed but with an error or warning from ODEX.

Required Values

4 (the default).

FatalError

Description

The return code indicating that the batch command failed because of a fatal error in ODEX.

Required Values

8 (the default).

NotAttempted

Description

The return code indicating that the batch command was not attempted. Usually this return results from a file syntax error.

Required Values

12 (the default).

Terminated

Description

The return code indicating that the batch command processor has been terminated for some reason, for example, stopped by an operating-system level command.

Required Values

1 (the default).

Chapter 5

Implementation

This chapter explains sample implementations and a sample schema to help you understand how to implement the e*Way Intelligent Adapter for Odette FTP in a production environment.

5.1 Implementation: Introduction

This chapter explains implementation examples and provides additional information on the Odette FTP e*Way sample included on your installation CD-ROM.

Each of these e*Gate Integrator schema examples allows you to observe and/or create end-to-end data-exchange scenarios involving e*Gate, the e*Way, the ODEX application.

Additional sections in this chapter explain the Odette FTP (OFTP) Event Type Definition (ETD) structure, ODEX configuration, and other information you need to know in implementing this e*Way.

5.2 Sample Implementations

All the schema examples in this section use one Odette FTP e*Way Connection configuration that is consistent with the corresponding ODEX configuration. This section provides an overview of some sample schema implementations.

Note: *See* **"ODEX Configuration" on page 63** *for information on configuring ODEX with the e*Way. For complete configuration information, see the appropriate ODEX user's guides.*

5.2.1 ODEX Configuration for Examples

There are two ODEX servers configured on a LAN for demonstration purposes.

Note: *See* **"ODEX Configuration" on page 63** *for information on where you enter these settings and what they mean.*

All the examples exchange data between a current site (10.1.191.48) with a Windows operating system (OS) and a remote site (10.1.201.27) with a UNIX OS. The complete ODEX configuration is not explained here. However, you do need the following ODEX configuration information to understand the ETD configuration:

Current Site

- SELF (INTERNAL NODE):
- DISPLAY NAME: SeeBeyond
- SSID: O093120780410000000SBYN01
- LOCALCODE: DEFAULT INTERNAL NODE

Trading Partner (external network node):

- DISPLAY NAME: ABC
- SSID: O093120780420000000SBYN02
- LOCALCODE: TRAD0001
- CALLED ADDRESS: 10.1.201.27

Remote Site

- SELF (INTERNAL NODE):
- DISPLAY NAME: ABC
- SSID: O093120780420000000SBYN02
- LOCALCODE: DEFAULT INTERNAL NODE

Trading Partner (external network node):

- DISPLAY NAME: SeeBeyond
- SSID: O093120780410000000SBYN01
- LOCALCODE: TRAD0002 (has nothing to do with LOCALCODE in the other site)
- CALLED ADDRESS: 10.1.191.48

5.2.2 ETD Configuration

The examples mimic a situation where the Odette FTP e*Ways run on the current site and exchange data with a remote peer/trading partner.

Note: *See* **Chapter 4** *for details on how to configure the e*Way parameters.*

Choose applicable configuration parameters for **Odette FTP ETD Configuration** (see **"Odette FTP ETD Configuration" on page 28**) from the following list:

- **ODEX: ODEX Professional**
- **Odex Batch Processor: C:\EDI32\ODEXBAT.EXE**
- **Batch File: C:\WORK_AREA\ODEX_TEST\SCHFILE.CMD**
- **Outbound Box: C:\WORK_AREA\ODEX_TEST\OUTGOING_BOX**
- **To Partner: TRAD0001**
- **Outbound File: INVOICE101.DAT**
- **Outbound Virtual File: INVOICE101**
- **Inbound Box: C:\WORK_AREA\ODEX_TEST\INCOMING_BOX**
- **From Partner: TRAD0001**
- **Inbound File: ORDER101.DAT**
- **Inbound Virtual File: ORDER101**

5.2.3 Schema Example 1

In this schema, the Odette FTP e*Way tells ODEX to make a call to a trading partner then publishes the return code to the e*Gate JMS Intelligent Queue (IQ) Manager. In turn, a file e*Way picks up the return data and writes it into a local file.

In ODEX, making a call is like probing the remote partner to see whether there is anything to receive. If so, an exchange is triggered. Then after a call, because some files could be received by ODEX, the Collaboration can check for any newly received files. If there are such files, the Collaboration can extract them from ODEX.

Figure 4 on page 38 shows a Network View of this schema in the e*Gate Schema Designer graphical user interface (GUI).

Figure 4 Example 1: Network View

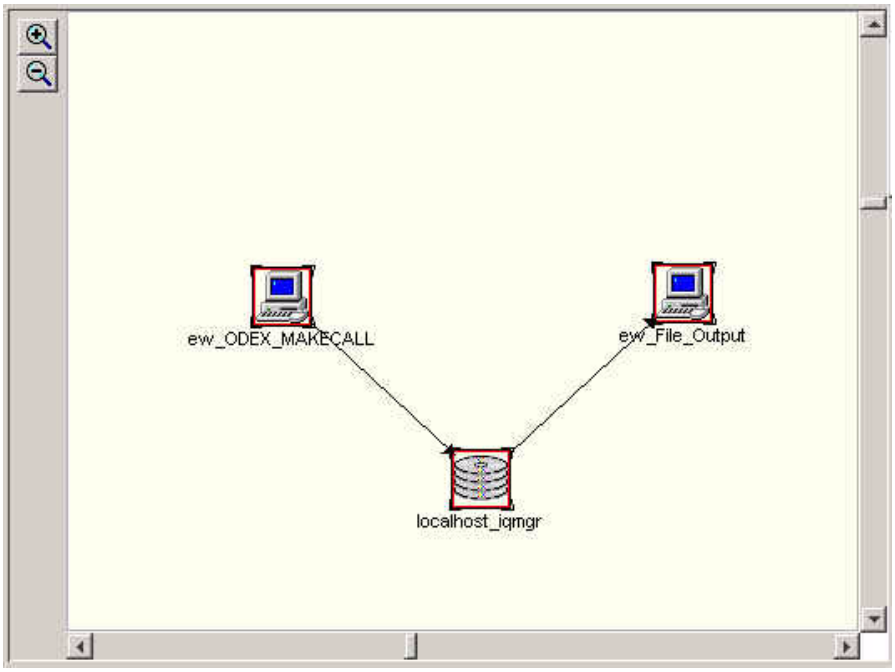


Figure 5 on page 39 shows the setup of the ETD nodes in the e*Gate Collaboration Rules Editor GUI. Only nodes concerned with making a call are expanded.

Figure 5 Example 1: ETD Nodes in Collaboration Rules Editor

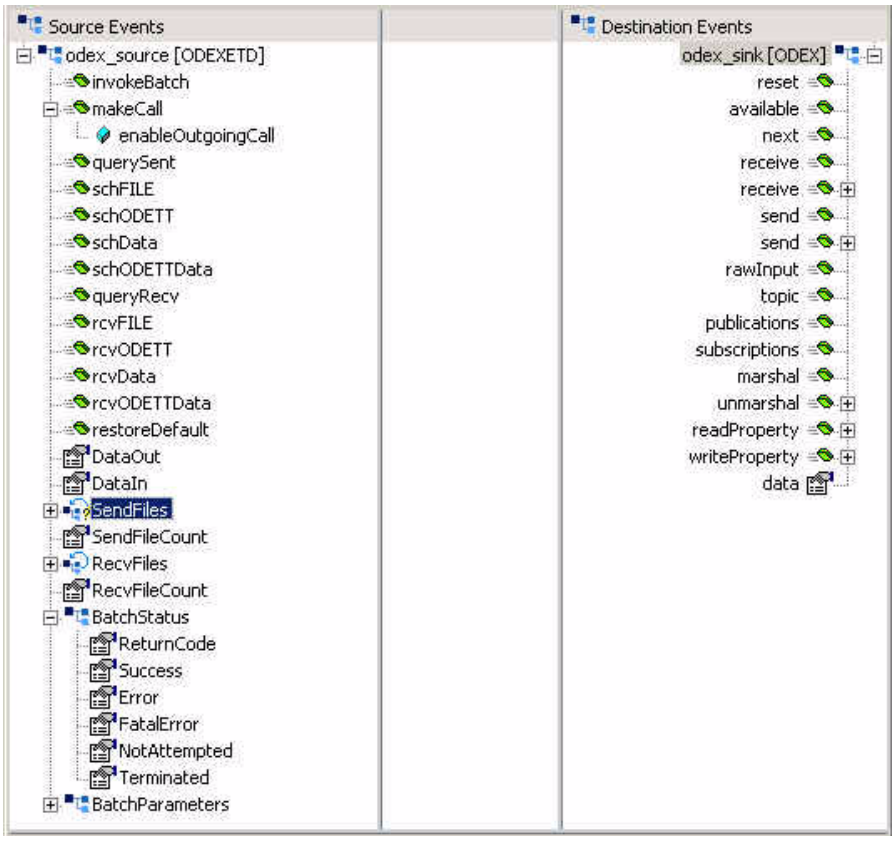
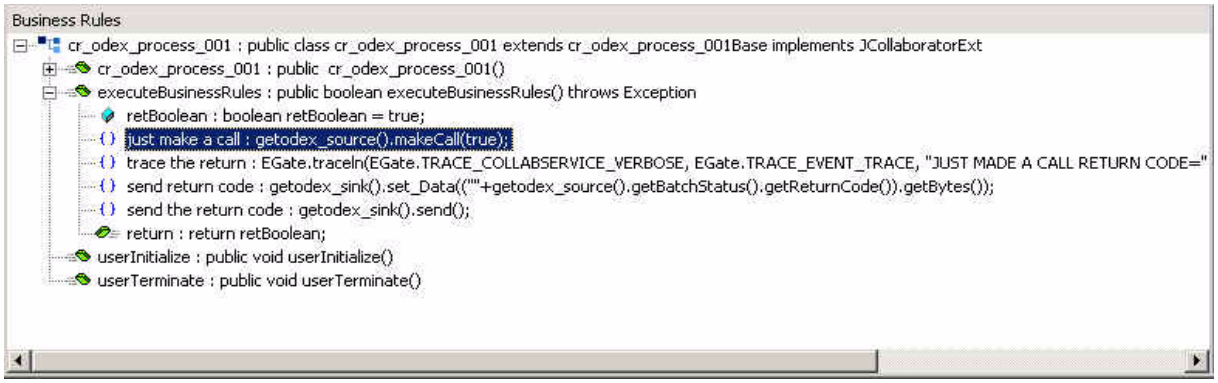


Figure 6 shows the applicable Business Rules in the same GUI.

Figure 6 Example 1: Business Rules



5.2.4 Schema Example 2

In this schema, the Odette FTP e*Way uses the method **invokeBatch** to call a user-written batch file. Such a file allows you to do whatever you want using the ODEX batch language.

In using this feature you can:

- Configure the **Batch File** parameter to point to the batch file
- Set the path pointing to a batch file by setting the value of the ETD node **BatchFile**, then call the **invokeBatch** method

If you have written the batch file, the ETD cannot know where your extracted files go and where to locate files to be sent. In such cases, the writer of the batch file must specify this information in the file.

In this example, the return data is published to JMS IQ Manager and, in turn, is picked up by a file e*Way and written to a local file.

Figure 7 shows a Network View of this schema in the e*Gate Schema Designer.

Figure 7 Example 2: Network View

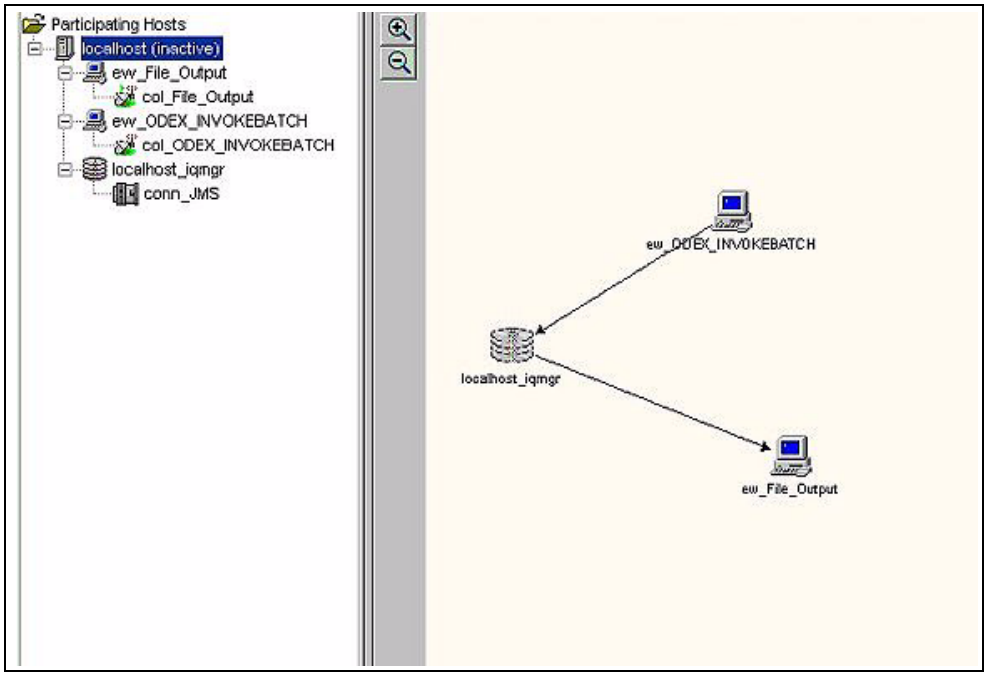


Figure 8 shows the setup of the ETD nodes in the e*Gate Collaboration Rules Editor. The parameter that corresponds to the **invokeBatch** method is **BatchFile**.

Figure 8 Example 2: ETD Nodes in Collaboration Rules Editor

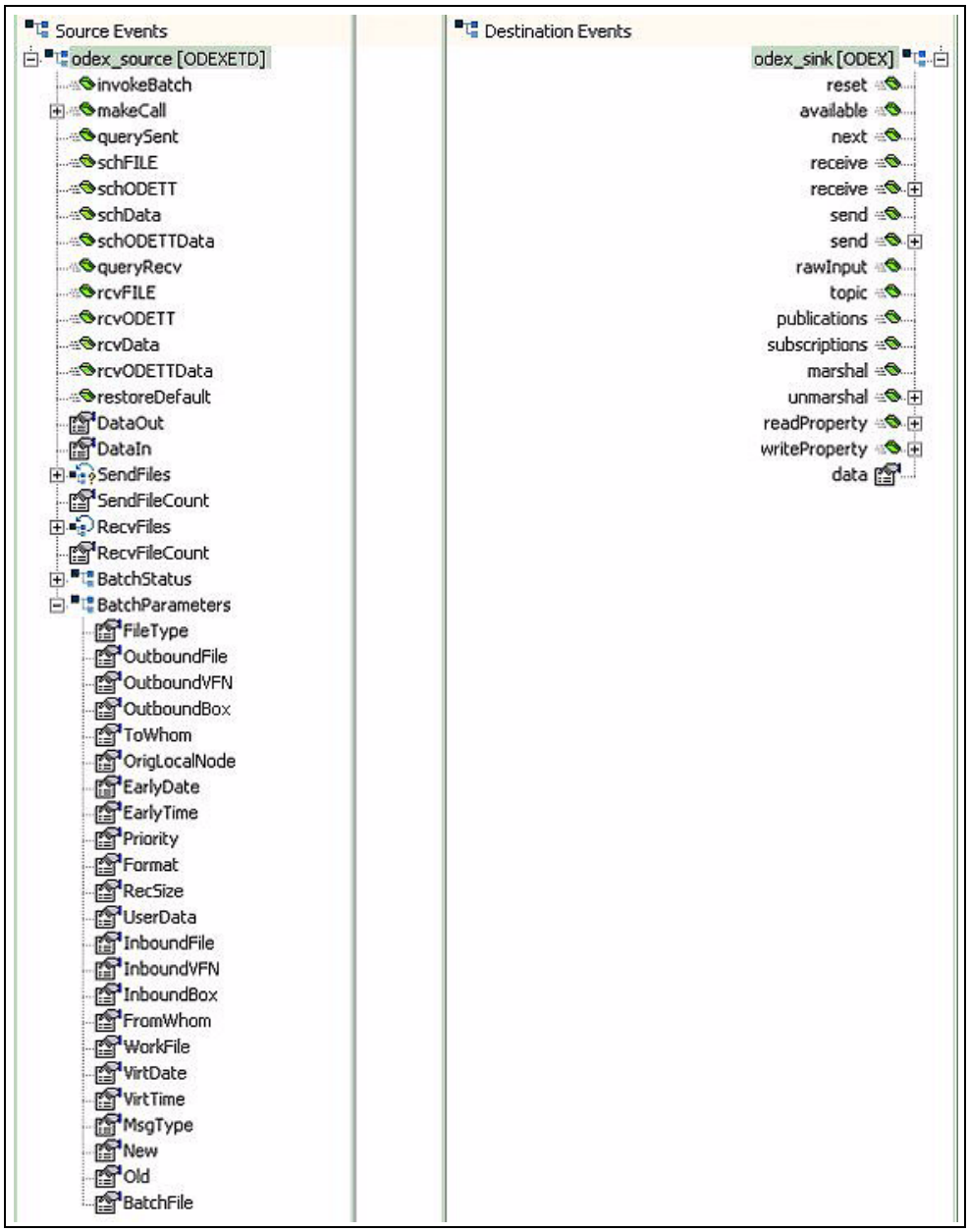
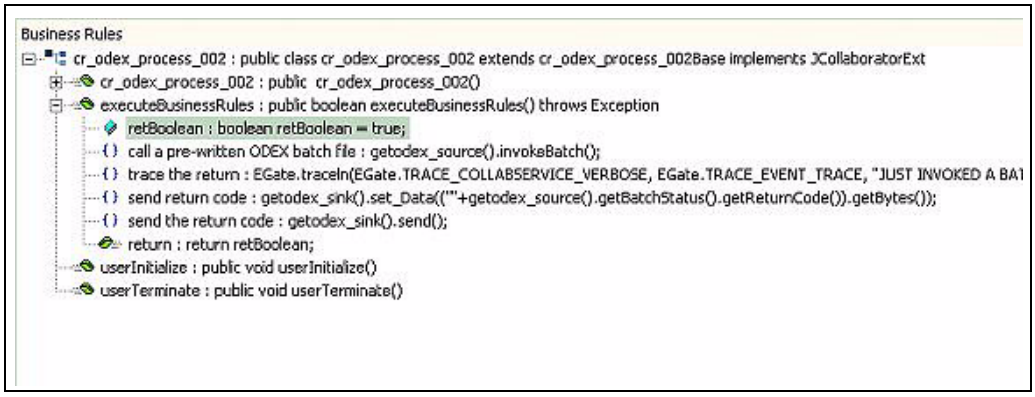


Figure 9 shows the applicable Business Rules in the same GUI.

Figure 9 Example 2: Business Rules



5.2.5 Schema Example 3

In this example, the e*Way queries ODEX for status information on files sent and scheduled (outgoing files), for example, scheduled time, sent time, end-to-end response (EERP) received time, failure, and retriability. For the sending and sent files, the summary of the query result is published to JMS IQ Manager and a file e*Way in turn picks up the result and writes it to a local file.

Figure 10 shows a Network View of this schema in the e*Gate Schema Designer.

Figure 10 Example 3: Network View

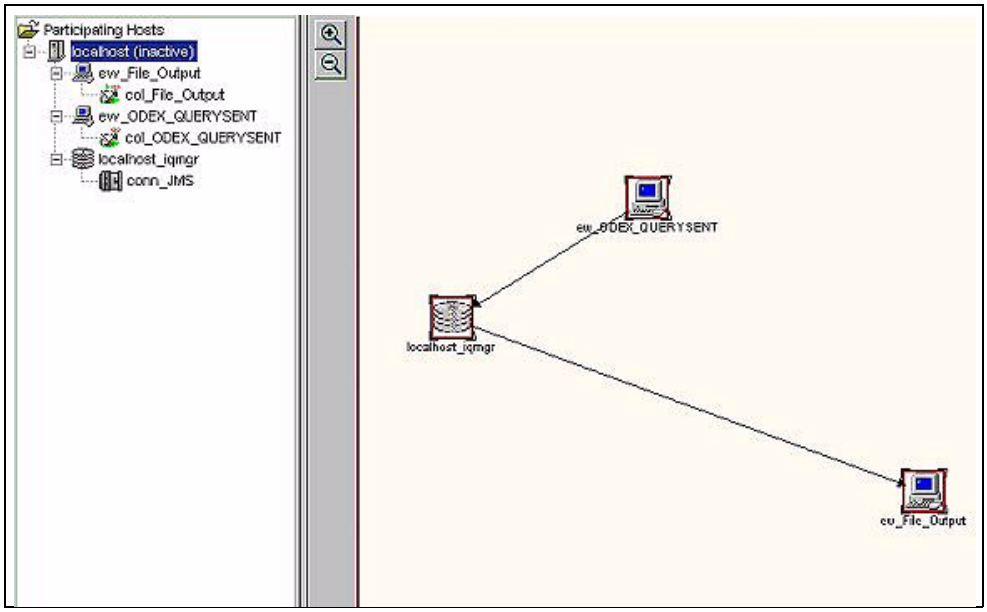
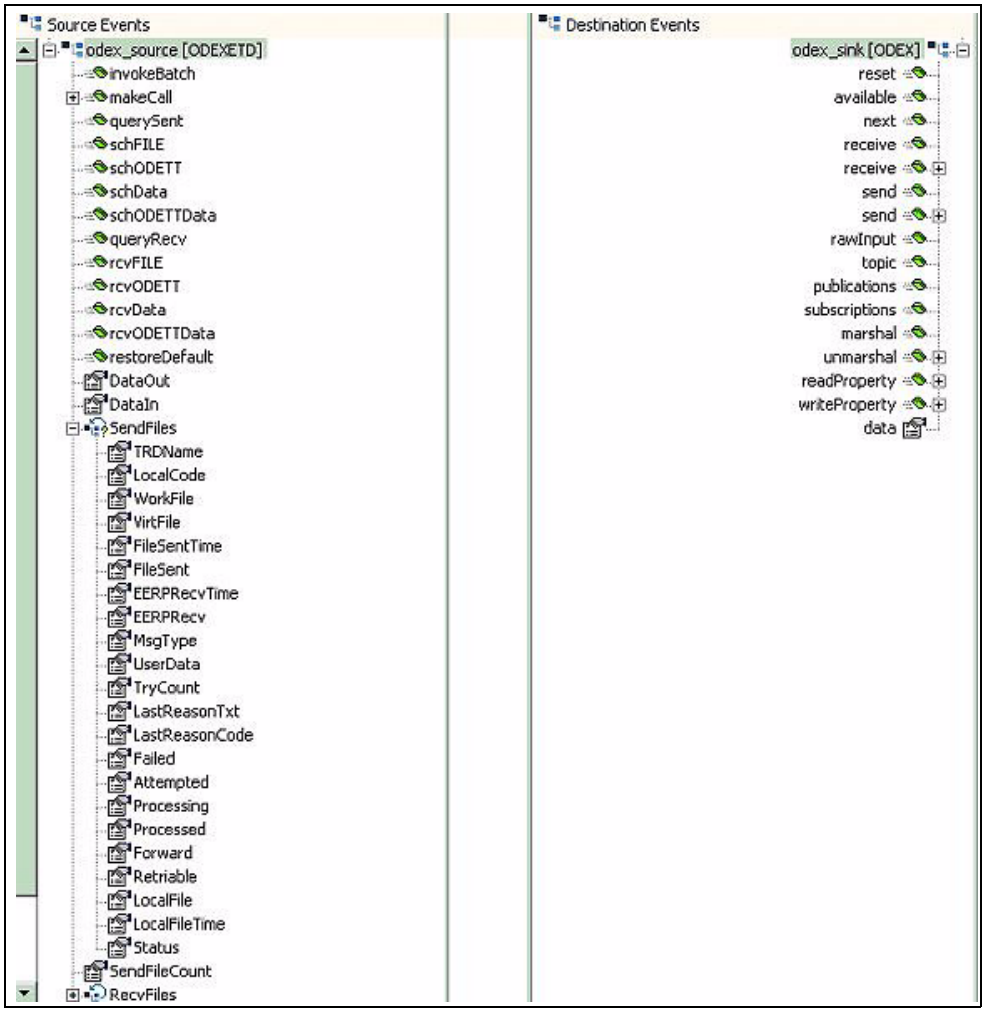


Figure 11 shows the setup of the ETD nodes in the e*Gate Collaboration Rules Editor. The desired information is contained in the **SendFiles** node.

Figure 11 Example 3: ETD Nodes in Collaboration Rules Editor

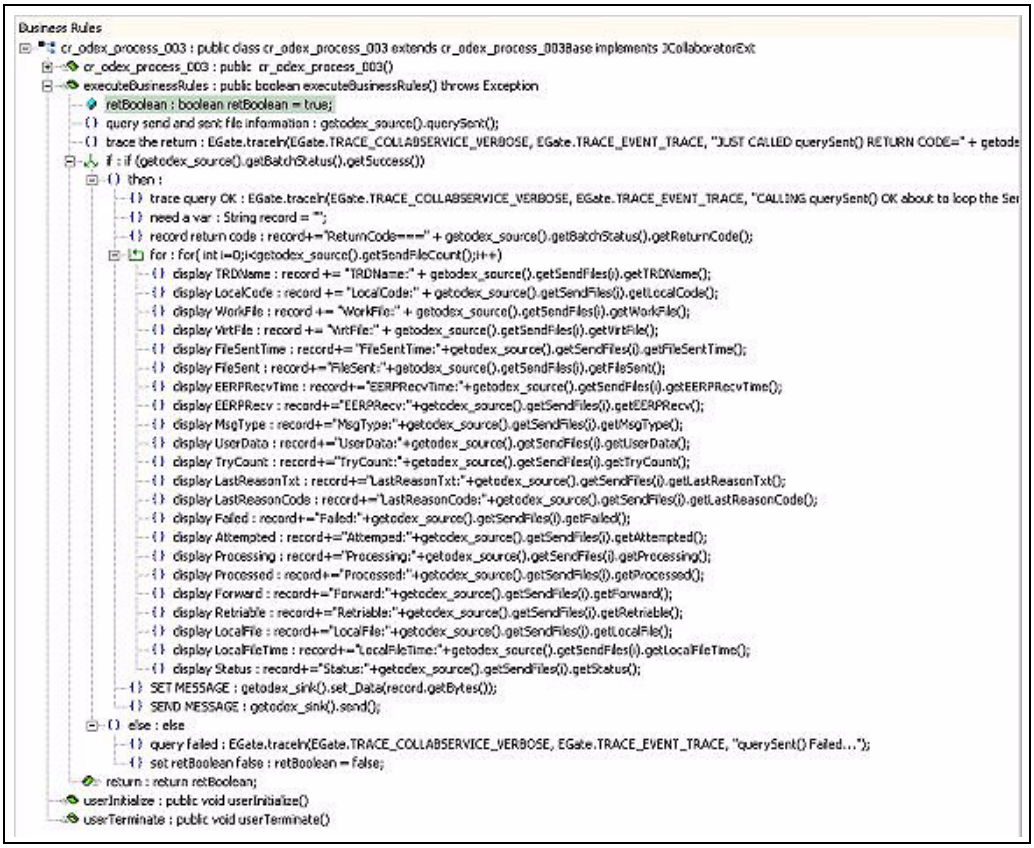


In Figure 11, only the **SendFiles** node is expanded. It is an enumeration node containing a list of file status objects that meet the query criteria. The method **querySent** uses the following parameters for the file type: C for current files, O for old files, and B for both current and old files.

Note: *See the appropriate ODEX user's guides for detailed an explanation of these categories. See* **Chapter 6** *for details on the e*Way's methods.*

Figure 12 shows the applicable Business Rules in the same GUI.

Figure 12 Example 3: Business Rules



The purpose of **querySent** and the enumeration node **SendFiles** is to monitor electronic data interchange (EDI) jobs scheduled to ODEX. You can also use these nodes to check the status of files sent and/or scheduled.

Note: *The Collaboration does not care whether the files are EDI or non-EDI and only queries the result then publishes it.*

The sub-nodes in **SendFiles**, for example, **FileSentTime**, **EERPRecvTime**, and **LastReasonCode**, can be used by the Collaboration to do sophisticated monitoring tasks and time-related scheduling.

Note that **FileSentTime** is a long integer representing milliseconds since January 1970. In addition, you can use this value with **FileRecvTime** and **EERPSentTime**

5.2.6 Schema Example 4

In this example (as in example 3), the e*Way queries ODEX for the status information such as scheduled time, sent time, EERP received time, failure, and retriability. For the sending and sent files, the summary of the query result is published to JMS IQ Manager and a file e*Way in turn picks up the result and writes it to a local file.

Figure 13 shows a Network View of this schema in the e*Gate Schema Designer.

Figure 13 Example 4: Network View

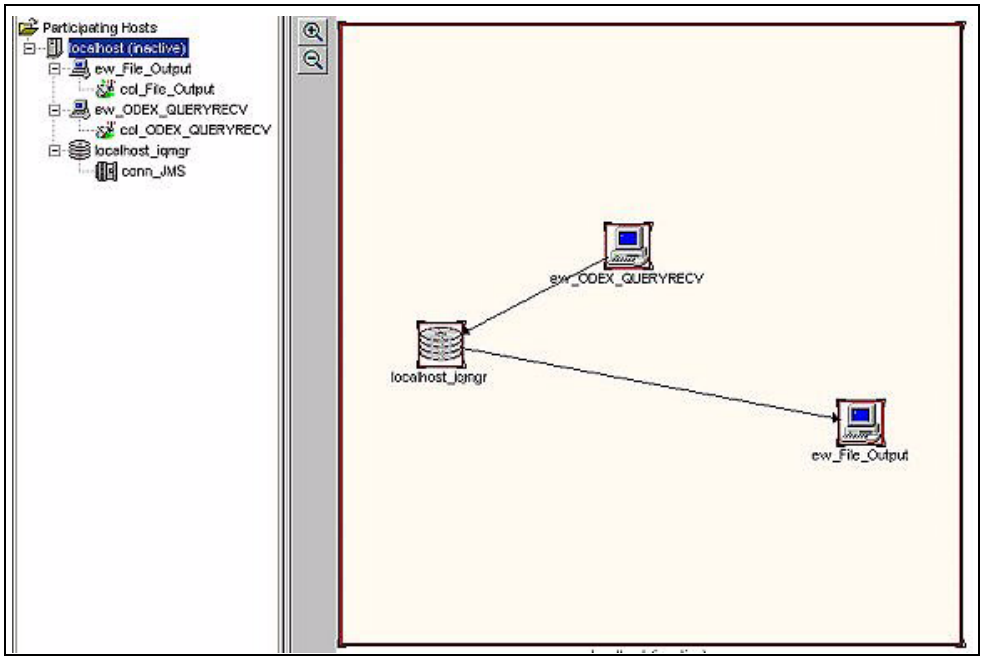
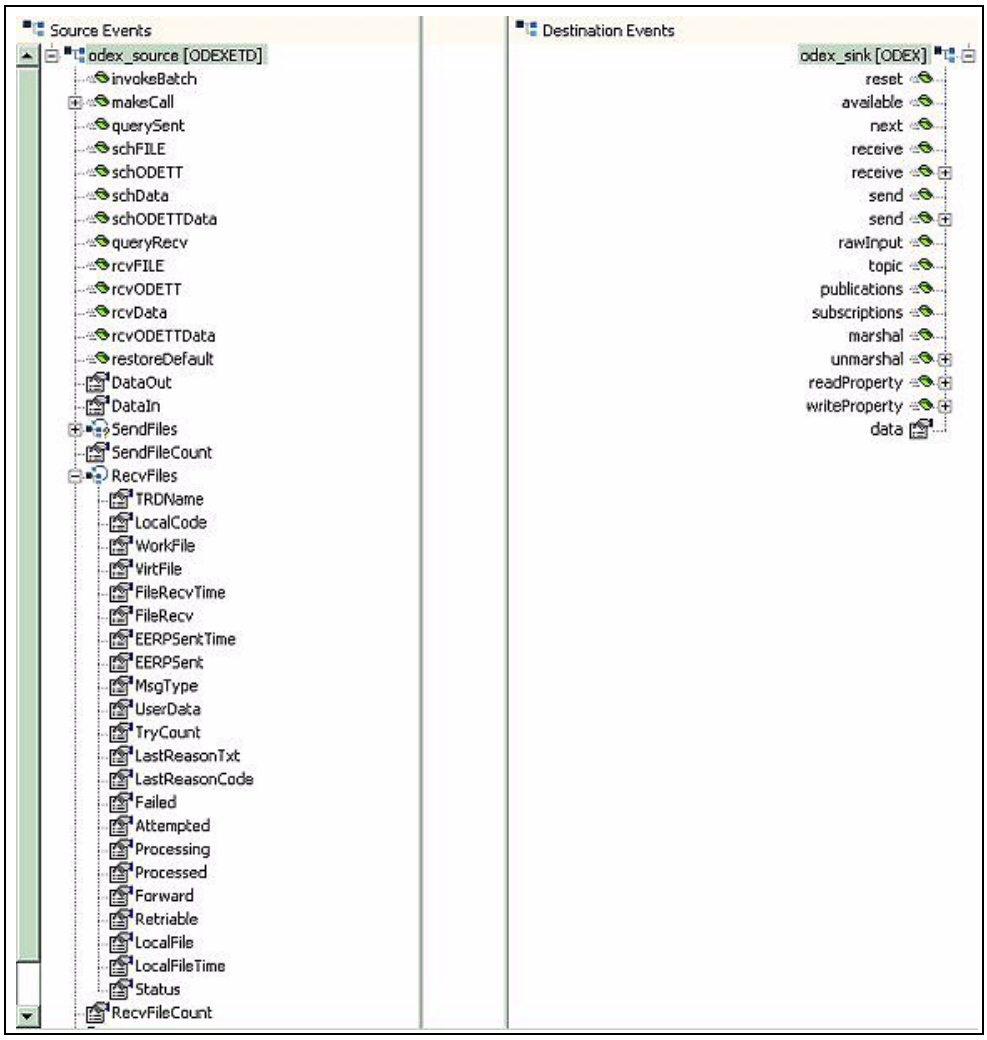


Figure 14 shows the setup of the ETD nodes in the e*Gate Collaboration Rules Editor.

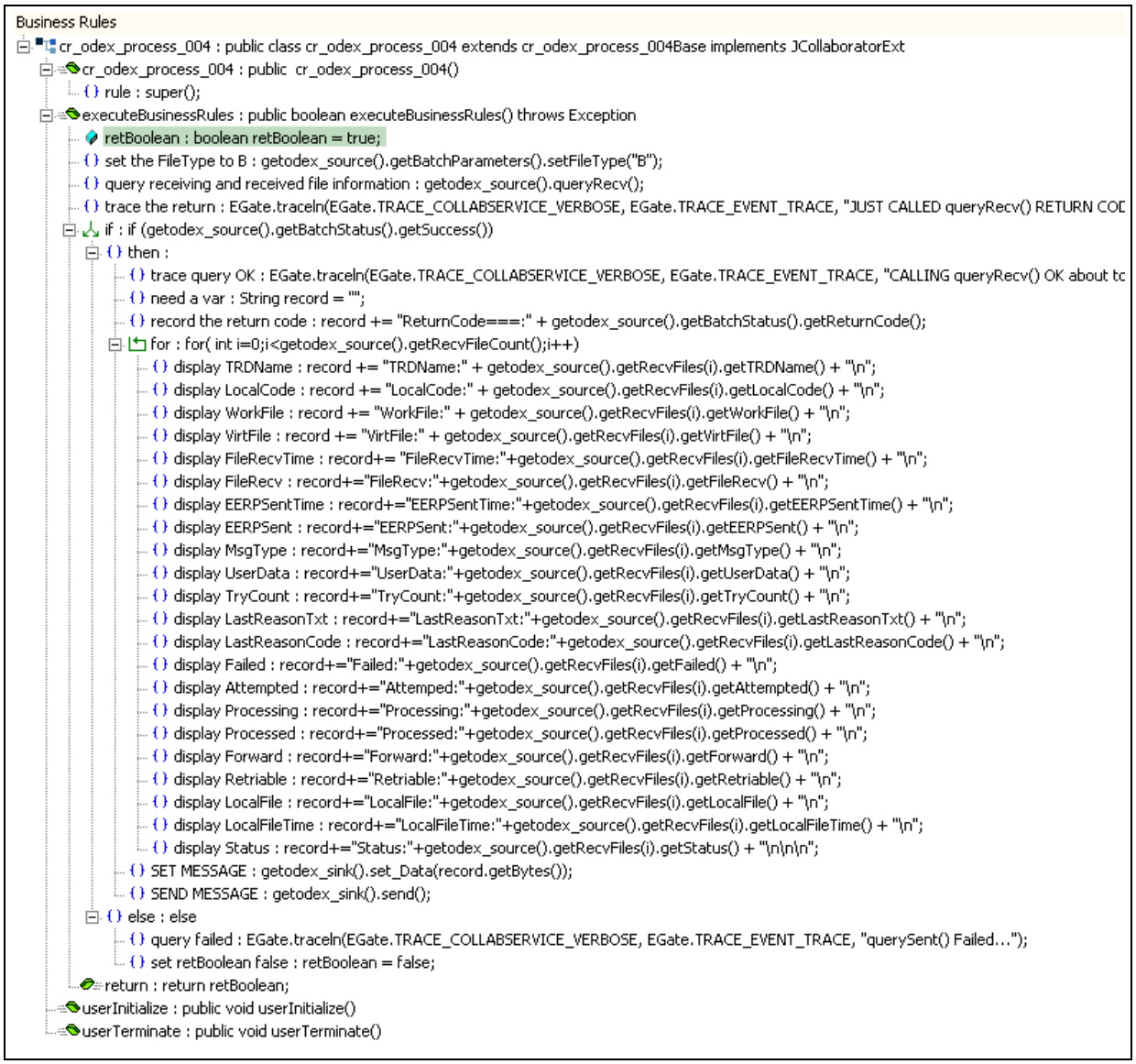
Figure 14 Example 4: ETD Nodes in Collaboration Rules Editor



In Figure 14, only the **SendFiles** node is expanded. It is an enumeration node containing a list of file status objects that meet the query criteria. The method **queryRecv** uses the following parameters for the file type: C for current files, O for old files, and B for both current and old files.

Figure 15 shows the applicable Business Rules in the same GUI.

Figure 15 Example 4: Business Rules



The purpose of **queryRecv** and the enumeration node **RecvFiles** is to monitor EDI jobs scheduled to ODEX. The Collaboration does not care whether the files are EDI or non-EDI and only queries the result then publishes it.

One important use for this query-and-iterate feature is to obtain the **WorkFile** name, an ODEX-internal unique file name that identifies a received file. Once the e*Way has this name, it can use a method, for example, **rcvFILE** or **rcvODETT**, to extract the file from ODEX.

The sub-nodes in **RecvFiles**, for example, **FileRecvTime**, **EERPSentTime**, and **LastReasonCode**, are used by the Collaboration to do sophisticated monitoring tasks and time-related scheduling.

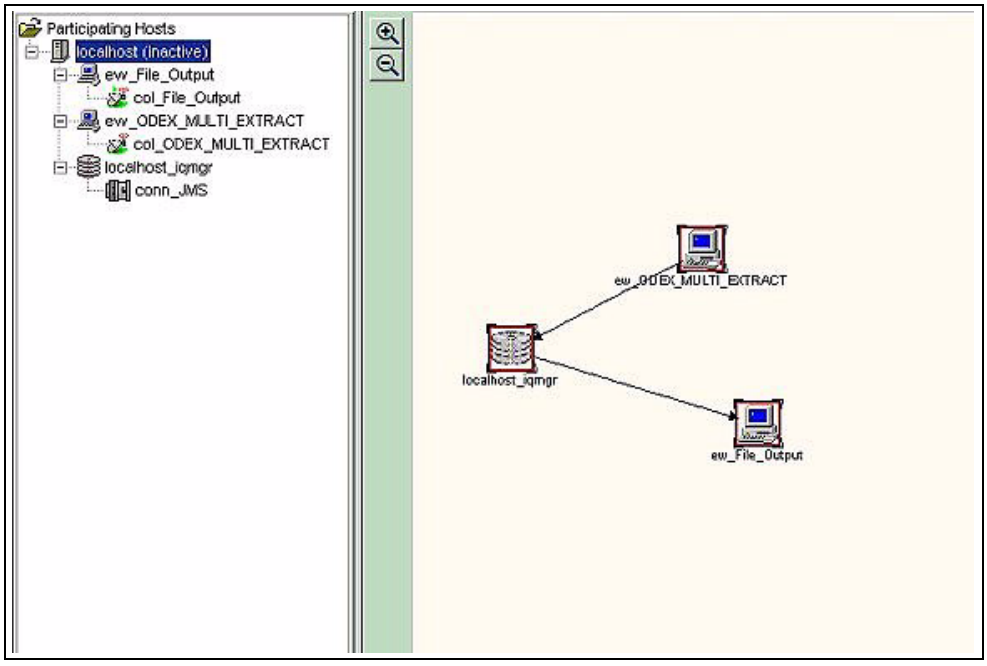
5.2.7 Schema Example 5

In this schema, the Odette FTP e*Way queries ODEX (using the method **queryRecv**) for received files and extracts then into the **DataIn** location (using the method **rcvData**) The Collaboration then publishes each file to the JMS IQ Manager. A file e*Way, in turn, picks up the files and writes them to local file.

As an alternative, you can replace **rcvData** with **rcvFile**. In this case the received files are extracted into the **InboundBox** location (a local folder), where another e*Way can be further route them through the e*Gate system.

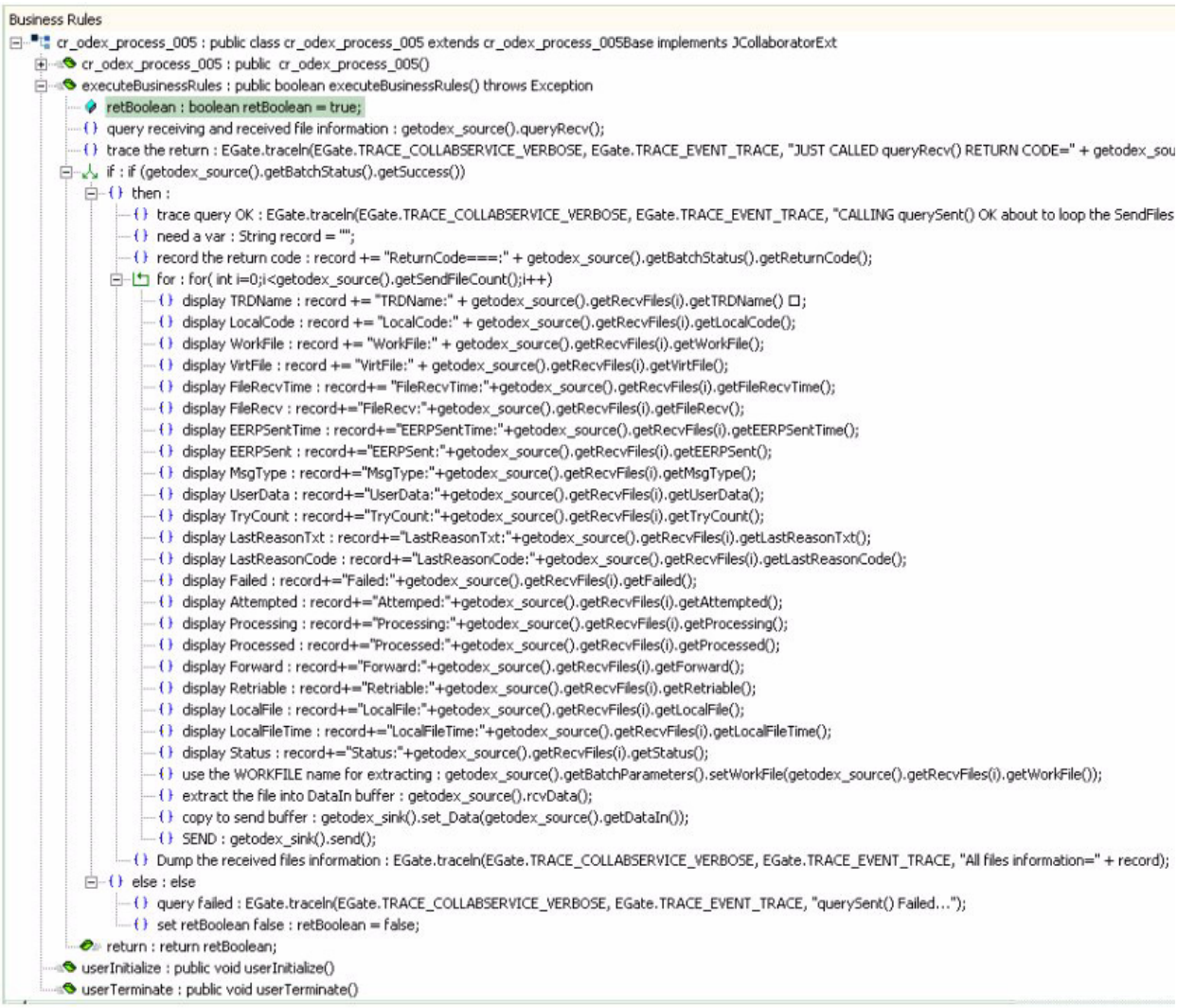
Figure 16 shows a Network View of this schema in the e*Gate Schema Designer.

Figure 16 Example 5: Network View



See **Figure 14 on page 46** for an illustration of the setup of the ETD nodes in the e*Gate Collaboration Rules Editor. **Figure 17 on page 49** shows the applicable Business Rules in the same GUI.

Figure 17 Example 5: Business Rules



In this schema, the Collaboration also accumulates the appropriate file status information and writes it to a log file.

5.3 Schema Sample

This section introduces you to the sample schema for the Odette FTP e*Way. Find this sample on the installation CD-ROM at the following path location:

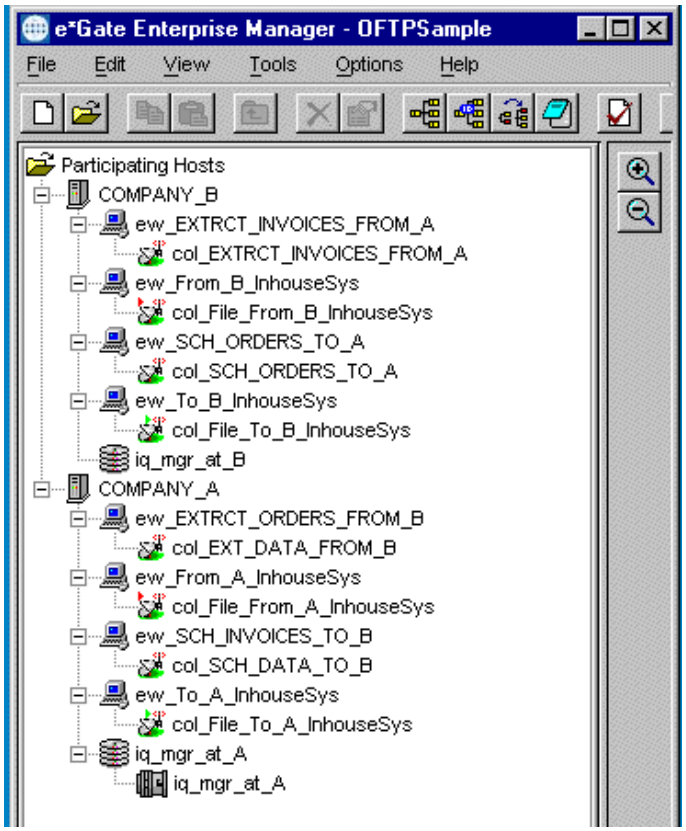
\samples\ewoftp

5.3.1 Schema Setup and Components

In this schema sample (called OFTPSample), Company A and Company B are trading partners. The schema's setup uses basic components and operations explained in the previous examples. Here, they are combined and work together to show how to create an end-to-end scenario involving two companies exchanging information.

Figure 18 shows the basic components contained in this schema, using the e*Gate Schema Designer's Network View.

Figure 18 OFTPSample: Basic Components



For detailed information on this schema and its operation, see the **Readme.txt** file that accompanies the sample.

5.3.2 Sample Schema: General Implementation Steps

This section explains how to import the Odette FTP e*Way sample schema and how to run the schema.

Before Schema Implementation

To use and implement a sample schema, the Odette FTP e*Way and sample schema must be installed, all of the necessary files and scripts must be located in the default location, ODEX must be installed and accessible to the e*Way, and you must also have access to an Odette FTP protocol-based network.

Importing a Schema

To import a sample schema

1. Copy the desired .**zip** file from the **Samples** directory in the install CD-ROM to your desktop or to a temporary directory, then unzip the file.
2. Start the e*Gate Schema Designer GUI.
3. On the **Open Schema from Registry Host** dialog box, click **New**.
4. On the **New Schema** dialog box, click **Create from export**, and then click **Find**.
5. On the **Import from File** dialog box, browse to the directory that contains the sample schema.
6. Click the .**zip** file then click **Open**.

   The sample schema is installed.

Running a Schema

To run a sample schema

- From the command line prompt, enter:

```
stccb -rh hostname -rs schemaname -un username
-up user password -ln hostname_cb
```

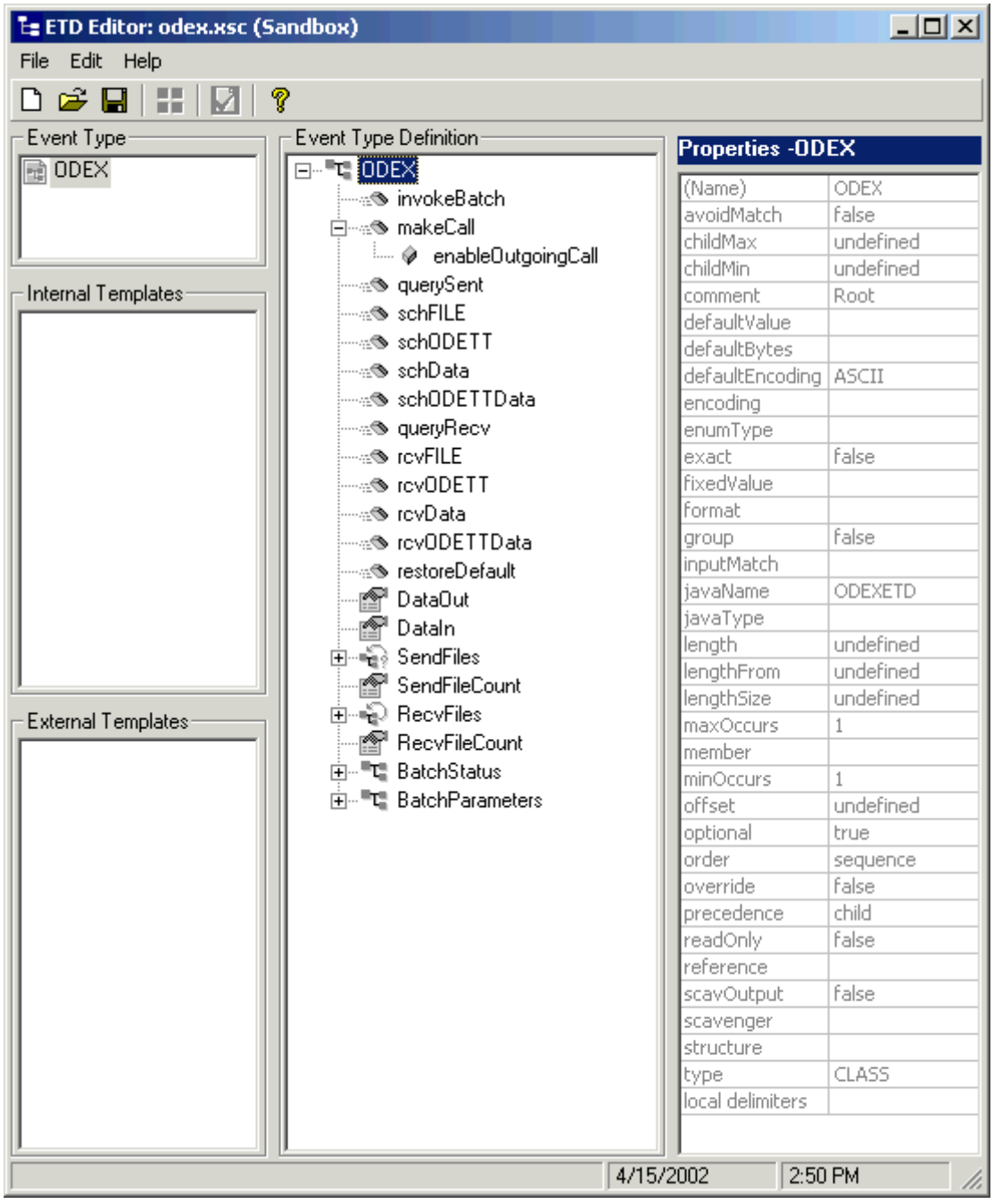
  Substitute ***hostname***, ***username***, ***schemaname***, and ***user password*** as appropriate.

The schema components start automatically. When there are no more run-time messages, check the output file. If the schema is operating correctly, the final output directory contains the payload data.

5.4 ODEX ETD Structure

This section explains the structure and layout of the ODEX ETD (**oftp.xsc**). Figure 19 shows an example of the ODEX ETD in the Collaboration Editor Main window.

Figure 19 ODEX ETD in ETD Editor



Note: *The nodes shown in Figure 19, with a "+" (plus sign) to the left contain sub-nodes. See* **"ETD Attributes" on page 54** *for a list of these sub-nodes, with details.*

5.4.1 ETD Exposed Methods

As shown in **Figure 19 on page 52**, the ODEX ETD exposes important methods available to the e*Way. As a result, ODEX attribute-related nodes exposed through the ODEX ETD are from ETD node structure.

Note: *See* **Chapter 6** *for details on these methods.*

The ODEX ETD exposes the following methods:

- **invokeBatch** starts the ODEX batch processor (**odexbat.exe**) as a native process, and waits on it until it sends the return code (blocking mode). The return code is from the batch file that is being executed. This code can be accessed via the **ReturnCode** node in **InvokeStatus**. This method is introduced so the user can write a customized batch command file and run it by calling **invokeBatch**. The actual batch file executed is in the **BatchFile** node, and it defaults to the configuration parameter set under **Batch File**.

Note: *See* **Chapter 4** *for details on the e*Way's configuration parameters.*

Before an e*Gate Collaboration can call **invokeBatch**, it can optionally change the actual batch file executed by setting the **BatchFile** node to a full path pointing to a batch file on the Participating Host. This feature provides more flexibility for the Collaboration.

- **makeCall** generates a **DIRUPDAT** command and runs it. This command can make ODEX generate an outgoing call even if there is nothing to send out. This method is only needed when the trading partner does not initiate a session.
- **schFILE** generates a **SNDFILE** command and runs it. This method relays a file to ODEX so ODEX can send it. This operation is done asynchronously, that is, when the method returns a "success" code, it means the job and the data has been handed over to ODEX for ODEX to schedule. ODEX also copies the data into its own repository, ensuring its safety.

  ODEX schedules the date and time it actually sends the data, as determined by how you have configured ODEX and/or the commands it receives. The data could be sent immediately or as specified by a command, for example, one hour later.
- **schODETT** generates a **SNDODETT** command and runs it. This command is the same as **schFILE** except the data must be in a specific EDI format.
- **schData** schedules the content of the outbound payload, the **DataOut** node, to ODEX to be sent. The content must be in a non-EDI format.
- **schData** schedules the content of the outbound payload, the **DataOut** node, to ODEX to be sent. The content must be in a non-EDI format.
- **schODETTData** schedules the content of the outbound payload, the **DataOut** node, to ODEX to be sent. The content must be in a specific EDI format.
- **EERPRecvTime** returns the time stamp when the EERP is received for a sent virtual file (VF). The method returns a null if the EERP has not been received yet (see **"EERP Handling" on page 69**).

- **fileSentTime** returns the time stamp when the VF is sent by ODEX and returns a null if the VF has not been sent yet.
- **rcvFILE** generates a **RCVFILE** command and run it. This method extracts the data from ODEX's repository of received files. The actual data could have been received previously, so the extracting and receiving of the file are both asynchronous.
- **rcvODETT** generates a **RCVODETT** command and runs it. This command is the same as **rcvFILE** except the data must be in a specific EDI format.
- **queryRecv** queries ODEX for all receiving and received file information, for example who sent the file, when it was received, and the EERP send time.
- **rcvFILE** generates a **RCVFILE** command and runs it. This method extracts the data from ODEX's repository of received files. The actual data could have been received previously, so the extracting and receiving of the file is asynchronous.
- **rcvODETT** generates a **RCVODETT** command and runs it. This method is the same as **rcvFILE** except the data must be in a specific EDI format.
- **rcvData** is the same as **rcvFILE** except the data is loaded in the inbound payload, the **DataIn** node, which is convenient for further routing.
- **rcvODETTData** is the same as **rcvODETTData** except the content must be in a specific EDI format.
- **EERPSentTime** returns the time stamp when the EERP is sent for a VF received from the trading partner. The method returns a null if the EERP has not been sent yet.
- **fileRecvTime** returns the time stamp when the VF is received from the trading partner and returns a null if the VF has not been received yet.
- **querySent** returns the time stamp for when the EERP is sent for a received VF from a trading partner. This method returns a null if the file has not been sent yet.
- **restoreDefault** restores the default values of all the configuration parameters in the group **BatchParameters**.

5.4.2 ETD Attributes

The ODEX ETD has the following attribute nodes:

- **DataOut**: The outgoing payload; this is a buffer. This node can receive its contents from another ETD, for example, an Event in an IQ. This ETD can then be sent to your trading partner (by calling **schFILE** or **schODETT**).
- **DataIn**: The incoming payload; this is a buffer. This node can receive its contents from a file extracted from ODEX (by calling **schFILE** or **schODETT**). These contents can then be copied to another ETD for further routing, for example, being sent to an e*Gate IQ.

Note: *Transferring send or receive payloads into and out of ODEX is implemented via a local file on the Participating Host. This file resides in a folder indicated by the configuration parameter* ***Temp Directory.*** *The ODEX batch command interface is used to facilitate these operations.*

- **SendFiles**: This is a repeating node, and its entries are loaded by invoking **querySent**. These entries contain status information for the outbound/sent files in the ODEX repository. The status information sub-nodes contained in this node are:
  - **TRDName**: The trading partner's name, extracted from the ODEX User Directory using **LocalCode**.
  - **LocalCode**: The local code ID for the trading partner to whom the file is sent.
  - **WorkFile**: The FTP file name in the ODEX repository, for the scheduled file.
  - **VirtFile**: The VFN for the scheduled file.
  - **FileSentTime**: The time when the file was sent; -1 means it has not been sent yet.
  - **FileSent**: This Boolean return indicates whether the file has been sent.
  - **EERPRecvTime**: The time when the file's EERP has been returned by the destination; -1 means it has not been returned yet.
  - **EERPRecv**: This Boolean return indicates the file's EERP has been returned.
  - **MsgType**: The type of the first message in a file. The type is **UNKNOWN** if no valid message found.
  - **UserData**: The OFTP **userdata**.
  - **TryCount**: The number of attempts made by the current operation.
  - **LastReasonTxt**: The text showing the last reason for the current operation on the file (see **Table 3 on page 60**).
  - **LastReasonCode**: This numeric code indicates the last reason for the current operation on the file (see **Table 3 on page 60**).
  - **SchTime**: The time the file is scheduled to be sent.
  - **Failed**: This Boolean return indicates whether the current operation on the file has failed (and either is not allowed to retry or has exceeded a predefined retry limit).
  - **Attemped**: This Boolean return indicates whether the current operation on the file has been attempted and failed (the operation can still be retried).
  - **Processing**: This Boolean return indicates the file is being processed.
  - **Processed**: This Boolean return indicates the file processing is complete.
  - **Forward**: This Boolean return indicates the file needs to be forwarded.
  - **Retriable**: This Boolean return indicates the current operation is retriable.
  - **LocalFile**: The local file name that has been loaded into ODEX to be scheduled for sending.
  - **LocalFileTime**: The time the local file became scheduled.
  - **Status**: The ODEX file status code (see **Table 4 on page 61**).

- **SendFileCount**: This node represents the count of entries in **SendFiles**.
- **RecvFiles**: This is a repeating node, and its entries are loaded by invoking **queryRecv**. These entries contain status information for the inbound/received files in the ODEX repository. The status information sub-nodes contained in this node are:
  - **TRDName**: The trading partner's name, extracted from the ODEX User Directory using **LocalCode**.
  - **LocalCode**: The local code ID for the trading partner to whom the file is sent.
  - **WorkFile**: The FTP file name in the ODEX repository, for the scheduled file.
  - **VirtFile**: The VFN for the scheduled file.
  - **FileRecvTime**: The time when the file was received; -1 means it has not been received yet.
  - **FileRecv**: This Boolean return indicates whether the file has been received.
  - **EERPSentTime**: The time when the file's EERP has been sent by the destination; -1 means it has not been sent yet.
  - **EERPSent**: This Boolean return indicates the file's EERP has been sent.
  - **MsgType**: The type of the first message in a file. The type is **UNKNOWN** if no valid message found.
  - **UserData**: The OFTP **userdata**.
  - **TryCount**: The number of attempts made by the current operation.
  - **LastReasonTxt**: The text showing the last reason for the current operation on the file (see **Table 3 on page 60**).
  - **LastReasonCode**: This numeric code indicates the last reason for the current operation on the file (see **Table 3 on page 60**).
  - **Failed**: This Boolean return indicates whether the current operation on the file has failed (and either is not allowed to retry or has exceeded a predefined retry limit).
  - **Attemped**: This Boolean return indicates whether the current operation on the file has been attempted and failed (the operation can still be retried).
  - **Processing**: This Boolean return indicates the file is being processed.
  - **Processed**: This Boolean return indicates the file processing is complete.
  - **Forward**: This Boolean return indicates the file needs to be forwarded.
  - **Retriable**: This Boolean return indicates the current operation is retriable.
  - **LocalFile**: The local file name that has been created for a file extracted from ODEX.
  - **LocalFileTime**: The time the local file was extracted.
  - **Status**: The ODEX file status code (see **Table 4 on page 61**).
- **RecvFileCount**: This node represents the count of entries in **RecvFiles**.

BatchStatus: This node contains all the return conditions for batch command execution, in the following sub-nodes:

- **ReturnCode** (integer) is the return code from **odexbat.exe**.
- **Success** (Boolean) is true if the return code is 0.
- **Error** (Boolean) is true if the return code is 4.
- **FatalError** (Boolean) is true if the return code is 8.
- **NotAttempted** (Boolean) is true if the return code is 12.
- **Terminated** (Boolean) is true if the return code is 1.

Note: *See* **Table 2 on page 33** *for a list and explanation of the ODEX batch processor return codes.*

- **BatchParameters**: This node contains all the parameters needed for invoking batch commands. These commands are made by ETD methods such as **makeCall**, **schFILE**, **schODETT**, **rcvFILE**, **rcvODETT**. For any given method, only some of the parameters apply, so check each method for detail. The sub-nodes contained in this node are:
  - **ToWhom**: The unique name for the trading partner to whom data is sent or a call is going to be made. This ID must be a valid **LocalCode** in your ODEX User Directory (see **"User Directory Configuration" on page 64**).

    The initial (default) value is taken from the configuration parameter **To Partner**.

Note: *When **restoreDefault** is called, the current value of this node is set to the value of the configuration parameter **To Partner**. See* **Chapter 4** *for details on the e*Way's configuration parameters.*

  - **OutboundFile**: The local file name to be sent to your trading partner. This file must be located in a local directory indicated by **OutboundBox**.

    The initial (default) value is taken from the configuration parameter **Outbound File.**
  - **OutboundVFN**: The VF name (VFN) used when ODEX sends the local file. The file name's syntax must comply with the restrictions for a VFN, as defined by RFC2044 guidelines.

Note: *See the appropriate ODEX user's guide for a detailed explanation of RFC2044 guidelines.*

    The initial (default) value is taken from the configuration parameter **Outbound Virtual File**.

- **OutboundBox**: The full path pointing to a local system directory where an outbound file (indicated by **OutboundFile**) is located. The implementation of the ETD methods **schFILE** and **schODETT** locate the physical file to be sent there.

  The initial (default) value is taken from the configuration parameter **Outbound Box**.

- **OrigLocalNode**: A second-level **LocalCode** identifying the departmental originator, in addition to Network Node level information.

Note: *See the appropriate ODEX user's guide for a detailed explanation of the Network Node, and* ***LocalCode****.*

  The initial (default) value is taken from the configuration parameter **Orig Local Node**.

- **EarlyDate**: A second-level **LocalCode** identifying the departmental originator, in addition to Network Node level information.

  The initial (default) value is taken from the configuration parameter **Orig Local Node**.

- **EarlyTime**: The earliest date that ODEX can send a file. Use this command for scheduling a file to ODEX to be sent. The format is YYMMDD.

  The default value is the current date.

- **Format**: The format of the transfer. The valid values are **U**, **T**, **F**, and **V** (see **"Format" on page 30**).

  The default value is **U**, unformatted.

- **RecSize**: When the format of the transfer is **F**, **RecSize** is the size of the record.

- **Priority**: The priority for ODEX to schedule a file to be sent (see **"Priority" on page 29**).

- **UserData**: The OFTP **UserData** is an eight-character string whose usage is agreed between two trading partners exchanging data over OFTP (see **"UserData" on page 31**).

  The initial (default) value is taken from the configuration parameter **UserData**.

- **FromWhom**: The unique name for the trading partner from whom data is received and extracted. This ID must be a valid **LocalCode** in your ODEX User Directory.

  The initial (default) value is taken from the configuration parameter **From Partner.**

- **InboundFile**: The local file name to be used to hold incoming files extracted from ODEX. This file must be located in a local directory indicated by **InboundBox**.

  The initial (default) value is taken from the configuration parameter **Inbound File**.

- **InboundVFN**: The VF name (VFN) the e*Way looks for to extract received files from ODEX. For example, a typical VFN could be INVOIVE101.

Note: *VFN syntax must comply with the restrictions for VFNs, as defined by the RFC2044 guidelines. It is a good idea to agree upon VFNs, with your trading partners and/ members, so the file names can have easily recognizable meanings. See the appropriate ODEX user's guide for details.*

The initial (default) value is taken from the configuration parameter **Inbound Virtual File**.

- **InboundBox**: A full path pointing to a local system directory where an inbound file (indicated by **InboundFile**) is to be created. The implementation of the ETD methods **rcvFILE** and **rcvODETT** create the physical file in that directory, to hold data extracted from ODEX.

  The initial (default) value is taken from the configuration parameter **Inbound Box**.

- **VirtDate**: The date stamp of the VF to be extracted. This date is given in the format YYMMDD.
- **VirtTime**: The time stamp of the VF to be extracted. This time is given in the format HHMMSS.

Note: *Every VF carries its date-transferred and time-of-transfer information.*

- **MsgType**: If a file contains EDI data, then a specified message of **MsgType** can be extracted from it. The default is UNKNOWN indicating a non-EDI file.
- **New**: The flag indicating the latest non-EDI file extracted from ODEX. This value only applies only to **rcvFILE**.
- **Old**: The flag indicating the oldest non-EDI file extracted from ODEX. This value only applies only to **rcvFILE**.
- **BatchFile**: The full path pointing to a pre written ODEX batch command file on the Participating Host. This command/file is executed when an e*Gate Collaboration calls the ETD method **invokeBatch**.

  The initial (default) value is taken from the configuration parameter **Batch File**.

ODEX Last Reason Codes

Table 3 shows a list of the ODEX last reason codes.

Table 3 ODEX Last Reason Codes

Code	Name	Description
00	File successfully processed	No problem has occurred.
01	Invalid filename	The virtual file name (VFN) does not contain valid Odette characters. Ensure that only upper-case alphabetic characters, numbers 0 to 9 and the characters & ()-./ are used.
02	Invalid destination	The destination EDI code (file or message node) is not profiled in the ODEX User Directory.
03	Invalid origin	The origin EDI code (file or message node) is not profiled on the ODEX User Directory.
04	Storage record format not supported	The storage format for the VF cannot be supported by the destination. This problem would occur if, for example, a variable-length record was sent to a machine that could only handle fixed-length records.
05	Maximum record length not supported	The maximum record length that the destination machine can handle has been exceeded.
06	File size is too large	The file is too large for either the OFTP or the remote site to handle.
10	Invalid record count	The number of records field, stored on the EFID, does not match the number counted during the transmission.
11	Invalid byte count	The number of bytes field, stored on the EFID, does not match the number counted during the transmission.
12	Access method failure	The remote site has had a problem storing the virtual file. This code is often caused by the receiver when it is running out of disk space.
13	Duplicate file identity	The VFN, date, and time must be unique. This message states that a second VF has been transmitted with exactly the same name, date, and time as an existing file.
99	Unspecified reason code	There has been an error that does not fit into any of the above categories.

ODEX File Status Flags

Table 4 shows a list of the ODEX file status flags.

Table 4 ODEX File Status Flags

Flag	Description
0x80	Receiving/received file
0x40	Sending/sent file
0x20	Forwarding file
0x10	EERP sent/received status
0x08	Processing complete
0x04	Being processed
0x02	Failed but retriable
0x01	Failed; no retry allowed

5.4.3 ODEX ETD and ODEX Batch Jobs

If it is operating correctly, the ODEX server is always up and running. However, **odexbat.exe**, the ODEX batch processor, does not necessarily run continually. An e*Gate Collaboration starts and monitors **odexbat.exe** via the ODEX ETD.

The ODEX ETD exposes attributes and methods that allow a Collaboration to do the following ODEX batch (**odexbat.exe**) operations:

1. Set necessary parameters for starting a batch job. The Collaboration schedules a file (EDI or non-EDI) to ODEX for sending out, making an outgoing call to see whether the trading partner has something to send in, extracting received files from the ODEX repository. The Collaboration can specify files according to predefined criteria, for example, files from certain trading partners or files having certain VFNs.
2. Start **odexbat.exe**.
3. Receive the **odexbat.exe** return code. This code is another important way for the ETD (and the Collaboration) to check on a job's status.

Note: *See* **Table 2 on page 33** *for a list and explanation of the* ***odexbat.exe*** *return codes.*

Additional detailed return information can be extracted by parsing the response file (see step 4).

4. Log the batch job execution in the **odexbat.exe** response file. This information includes the error code and any applicable error messages. The Collaboration parses this file for detailed error information based on the error-level return code.
5. Support the payload. For the convenience of the Collaboration developer, the payload nodes are exposed in the ETD so the Collaboration can route data from e*Gate to external systems (then to trading partners) and from external systems into e*Gate.

5.5 Collaboration Implementation

This section contains examples of how to set up Collaborations to operate with the Odette FTP e*Way. These samples show several typical implementation use cases.

5.5.1 Example A: Send and Receive Payload

A basic Collaboration operation is to send a payload to a trading partner (as set in the ETD configuration under the parameter **To Partner**) and receive a payload from the trading partner (as set in the ETD configuration under the parameter **From Partner**). The payload sent and received is further routed into the e*Gate system (though IQs, for example).

Simple Cases

The ETD is designed so that, for simple cases, the least coding is needed for Collaboration development. For a more complex Collaboration, the ETD is flexible enough to let the developer customize the Collaboration to meet any desired needs.

Best Practices

It is best to configure an e*Way Connection to serve one trading partner for two kinds of data to be exchanged between you and your trading partner as follows:

- Set the **To Partner** and **From Partner** configuration parameters as the same company.
- For example, your customer (you are the supplier) expects the **Outbound Virtual File** value to be **INVOICE**, that is, what you need to send to your trading partner. Then, the **Inbound Virtual File** value can be **ORDER**, that is, what you are expecting from your trading partner.
- All you need to do in the Collaboration is to call **schFILE** (or **schODETT**) at a agreed-upon time to send the **INVOICE** data.
- Similarly, call **rcvFILE** (or **rcvODETT**) at an agreed-upon time to extract the **ORDER** data from ODEX.

Complex Cases

You can handle more complex cases in a similar way, for example, exchanging more than one type of data with more than one trading partner. The difference is that the Collaboration, before the actual scheduling and extraction are executed, has to set specific information such as the trading partner and data to be sent. You can do this operation easily using the **BatchParameters** sub-nodes exposed in the ETD.

The Collaboration can also check whether a file you have sent has been received by a trading partner by calling the EERP status-checking methods in the ETD. Similarly, a trading partner can discover whether the received file has been acknowledged by calling the same methods.

5.5.2 Example B: Using the Outbound Box

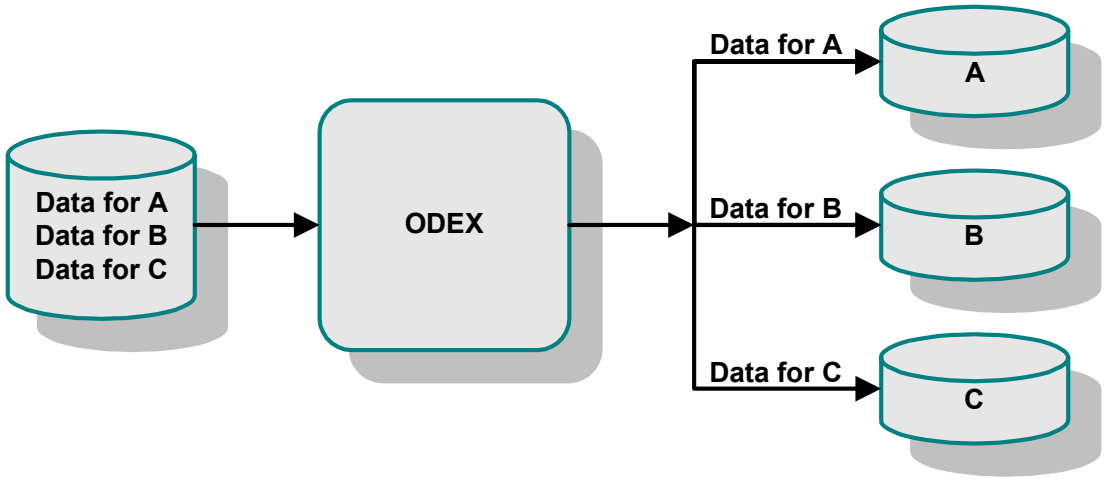
You can send an outbound file (configured in the **Outbound Box** parameter) to a trading partner (configured in the ETD as the parameter **To Partner**) and receive files from that trading partner (configured in the ETD as the parameter **From Partner**). You can then have the file stored in the **Inbound Box** configured location.

Files in the **Outbound Box** or **Inbound Box** locations can be further picked up by other e*Ways in the same schema. For example, you can use a file ETD to pick up the files or then send them to an ODEX ETD for more routing.

Routing EDI Files

When you are routing EDI files, keep in mind that they contain their own information necessary for data routing to one or more trading partners members. In these cases, the Collaboration does not need to set the destination, and ODEX can do the data splitting as shown in Figure 20.

Figure 20 ODEX Data Splitting



5.6 ODEX Configuration

Before you can use the e*Way, ODEX must be correctly installed and configured on the appropriate Participating Host. The e*Way does not configure ODEX, and it can only keep necessary information that allows it to generate and start the operation of ODEX batch files. Then, ODEX does the file transfer over OFTP to and from trading partner s.

This section explains essential steps you must do and considerations to keep in mind when configuring ODEX to operate with e*Gate and the Odette FTP e*Way.

Note: *For details on how to configure the e*Way Connection for the Odette FTP e*Way, see* **Chapter 4**.

5.6.1 User Directory Configuration

ODEX has its own User Directory feature, where you can enter your trading partner information. The ODEX User Directory dialog box is the GUI where you set up your User Directory.

When First Running ODEX

When you first fun ODEX, it prompts you for your own user information. It is especially important that you enter your own SSID, communication method, and other EDI-related information. In this way, you allow ODEX to have information necessary for your trading partners to communicate with you.

EDI Requirements

Before any EDI job can be defined or carried out, your trading partners members must be entered and organized in the ODEX User Directory. This feature becomes your EDI "phone book." You must configure your trading partners correctly in the User Directory.

This configuration includes entering their correct SSID, communication method, and all other information required for EDI transmission. ODEX calls this configuring *trading partner profiling*.

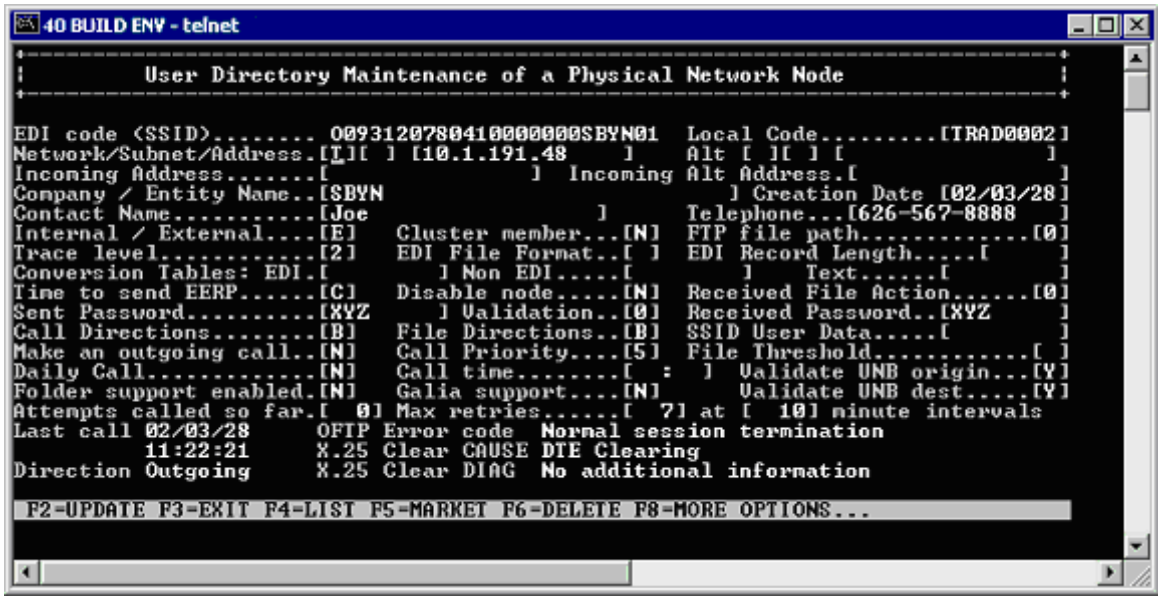
ODEX Plus

Figure 21 shows an example of the User Directory dialog box in ODEX Plus.

Figure 21 ODEX Plus User Directory Dialog Box

Figure 22 shows a list of trading partners s already profiled. You can use this dialog box add, delete, and update their information.

Figure 22 Updated ODEX Plus User Directory Dialog Box



A typical profile of a trading partner () includes:

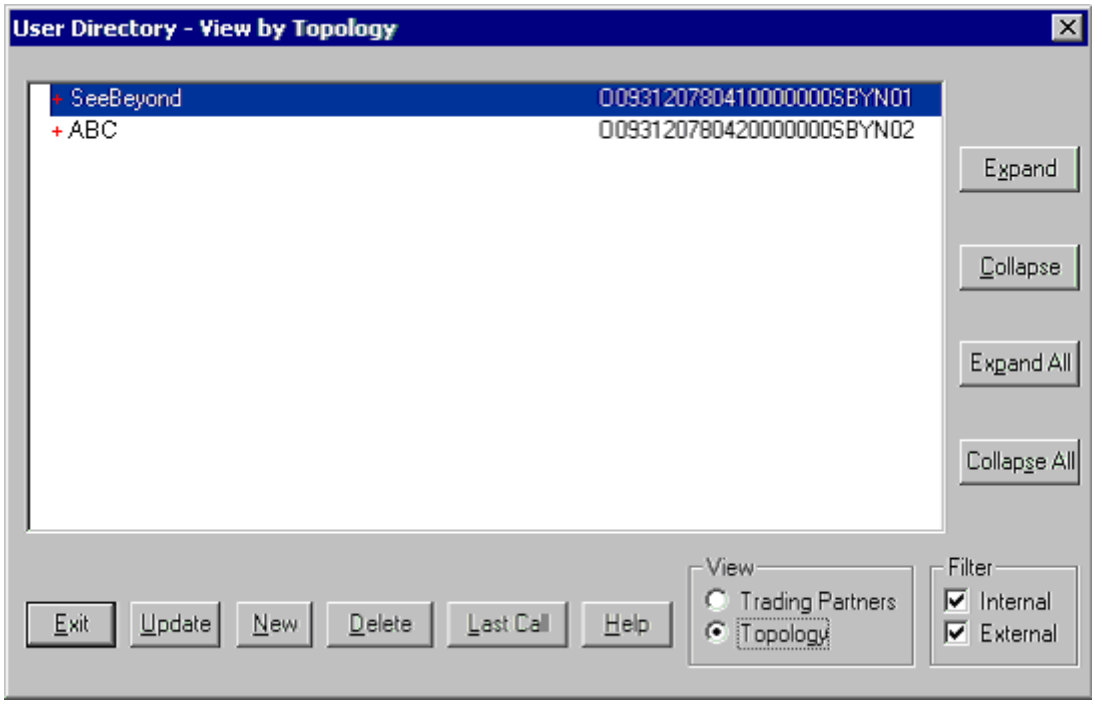
- **Display Name:** SBYN
- **SSID**: O093120780410000000SBYN01
- **Local Code**: TRAD0002
- **Network/Subnet/Address**: [T] indicating TCP/IP and the IP of the trading partner's host (here you define the communication details)

The **Local Code** ID is important in ODEX, because, after a trading partner is profiled, ODEX refers to it with this unique name. For example, in using the batch command, you must specify the trading partner you want to send file to by passing its **Local Code** ID to the batch command.

ODEX Professional

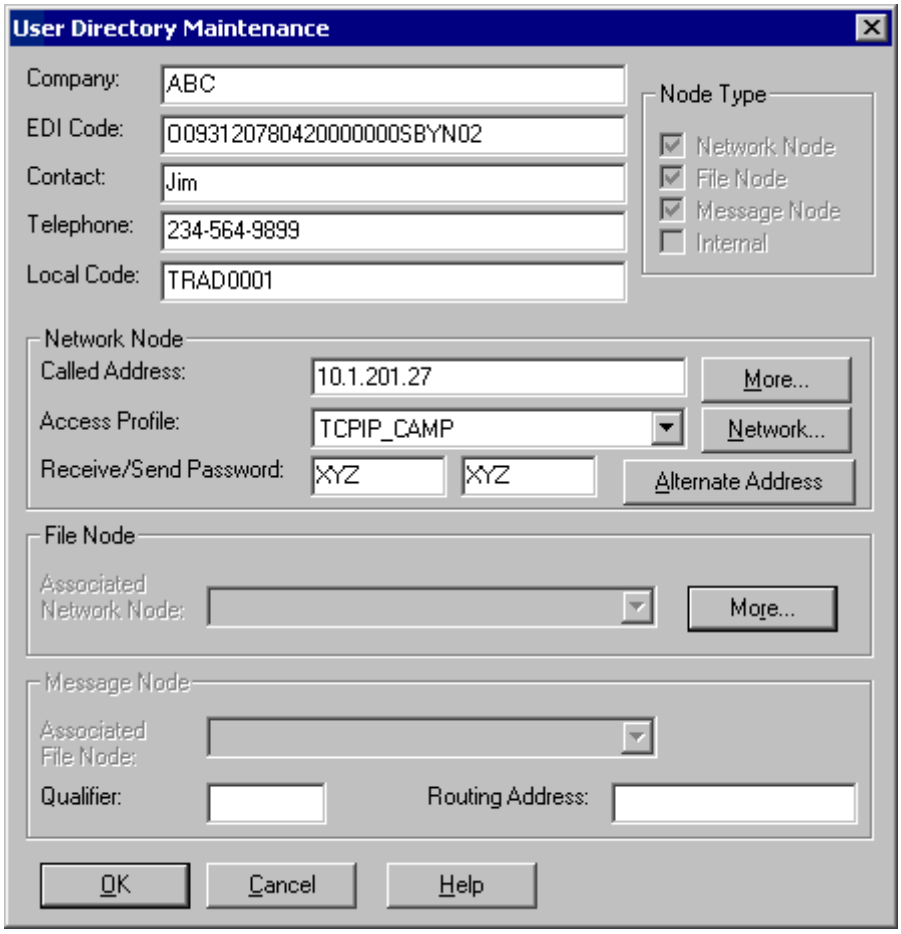
Figure 23 shows the User Directory dialog box for ODEX Professional.

Figure 23 ODEX Professional User Directory - View by Topology Dialog Box



Double-click on the name of the desired trading partner in this dialog box to display the complete User Directory entries for a given partner (see **Figure 24 on page 67**) in the User Directory Maintenance dialog box.

Figure 24 ODEX Professional User Directory Maintenance Dialog Box



Compare the User Directory features in ODEX Plus and ODEX Professional. In Figure 23 there are data communication peers on the network, and they profile each other as trading partner s.

Also note, in Figure 23, that the called IP address is 10.1.201.27. This address must be where your trading partner's EDI host is running.

5.6.2 Basic ODEX Configurations

This section explains additional basic parameters of ODEX, which you must configure.

Communication Configuration

You must specify in ODEX which communication protocols you are using (see Figure 25). You also need to specify input parameters, for example, for X.25, ISDN, or TCP/IP, as agreed upon between you and your trading partners.

Figure 25 ODEX TCP Communications Address Point Details Dialog Box

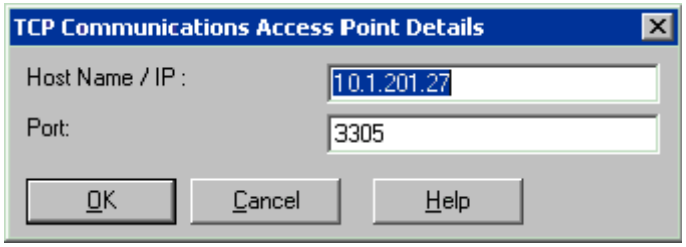


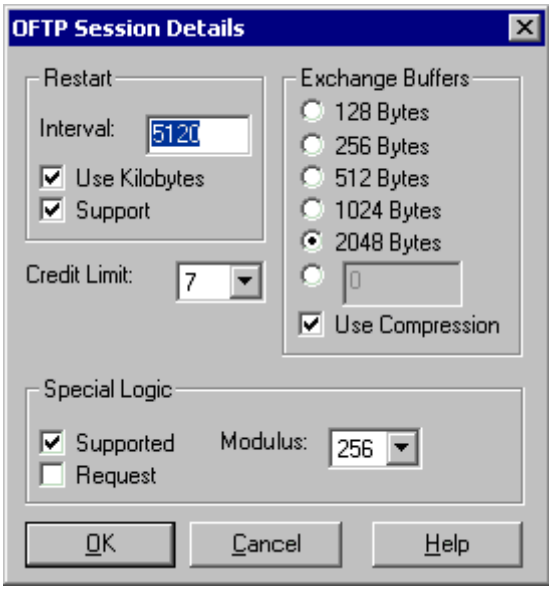
Figure 25 shows a TCP/IP communication configuration. For X.25 and ISDN, you need to enter additional information. See the appropriate ODEX user's guides for details.

OFTP Configuration

OFTP parameters such as **SEND PASSWORD**, **RECEIVE PASSWORD**, **SSID**, and **SFID** are needed by the OFTP stack to start file transfer sessions and other operations. Configure this information as part of trading partner profiling.

Also use the dialog box shown in Figure 26. You can configure this information once, and you do not have to change it again unless your trading partners change or alter their system parameters.

Figure 26 ODEX Session Details Dialog Box



Additional Configuration

Keep in mind that ODEX configurations such as EDI and Communication configurations can be extremely complex. ODEX handles these operations well and provides user-friendly GUIs for their configuration. ODEX also provides excellent online Help and documentation.

Basically, the e*Way only keeps a minimum set of configuration parameters in its own configuration file, so it can launch an EDI or non-EDI job exchanging data with one or more trading partners. Note that most of the elements in an EDI job are preconfigured, for example, trading partners a user can exchange data with and underlying communication protocols to be used. However, once you configure this information, data transmission and scheduling by ODEX become automatic.

5.6.3 EERP Handling

You must pay special attention to EERP support in ODEX. EERP is the OFTP method of acknowledgement to the originator (sender) of a file, that the file is received by its destination (receiver).

EERP is activated in the Odette FTP e*Way provided that you have configured ODEX correctly. For a given trading partner, you can configure its EERP to happen:

- When the file is received by ODEX
- When the file is copied out (extracted)
- On request (manually)

When you configure ODEX to work with the Odette FTP e*Way, your trading partner's EERP must be configured either on file receipt or when it is copied out. If your trading partner has one of these configurations, EERP is activated.

The ETD does not expose methods to do EERP operation. Instead, you must configure ODEX as explained in this section, so EERP can be sent at the proper time. The Collaboration can then use the **queryRecv** and **querySent** methods to query status information about files received or sent. This information includes the EERP status for a given file.

See **Chapter 6** for details on the e*Way's Java methods.

Note: *See the appropriate ODEX user's guides for complete information on how ODEX handles EERP.*

5.7 Recovery, Data Integrity, and Security

OFTP and ODEX adequately address the need for data recovery and integrity, as well as security.

ODEX supports the full the OFTP protocol, which includes OFTP restart functionality. In the ODEX configuration, if the **restart** option is enabled, an interrupted transfer is picked up during the next data exchange session, at the exact point where it was left off.

In terms of security, ODEX is a public X.25 system. These systems are protected by a very rigorous security regime. It is virtually impossible to "hack" into an X.25 server and change essential attributes and information.

Note: *ODEX describes its support of these features in its own documentation. For details, see the appropriate ODEX user's guides.*

Chapter 6

e*Way Java Methods

This chapter explains the Java classes and methods contained in the e*Way Intelligent Adapter for Odette FTP, which are used to extend the functionality of the e*Way.

6.1 Odette FTP e*Way Methods and Classes: Overview

For any e*Way, communication takes place both on the e*Gate system and the external system side. Communication between the e*Way and the e*Gate environment is common to all e*Ways, while the communication between the e*Way and the external system is different for each e*Way.

For the Odette FTP e*Way, the **stceway.exe** file (Multi-Mode e*Way, see **Chapter 3**) is used to communicate between the e*Way and e*Gate, and a Java Collaboration is utilized to keep the communication open between the e*Way and the ODEX application. ODEX, in turn, handles the actual Odette FTP (OFTP) file transfers.

Using Java Methods

Java methods have been added to make it easier to set information in the ODEX Event Type Definition (ETD), as well as get information from it. The nature of this data transfer depends on the configuration parameters (see **Chapter 4**) you set for the e*Way in the e*Gate Schema Designer's e*Way Editor graphical user interface (GUI). These parameters in turn determine parameters used by ODEX and its job scripts.

Note: *For more information on the ODEX ETD structure, its nodes, and attributes (based on the e*Gate **oftp.xsc** ETD file), see* **"ODEX ETD Structure" on page 52**.

The Schema Designer's Collaboration Editor GUI allows you to call Java methods by dragging and dropping an ETD node into the **Rules** dialog box.

Note: *The node name can be different from the Java method name.*

After you drag and drop, the actual conversion takes place in the **.xsc** file. To view the **.xsc** file, use the Schema Designer's ETD Editor GUI.

For example, if the node name is **SendFiles**, the associated **javaName** is **SendFiles**. If you want to get the node value, use the Java method called **getSendFiles**. If you want to set the node value, use the Java method called **setSendFiles**.

Java Classes

The Java methods for the Odette FTP e*Way are contained in the following classes:

- **ODEXConfig Class** on page 72
- **ODEXETD Class** on page 99
- **ODEXRecvFileDesc Class** on page 112
- **ODEXSendFileDesc Class** on page 122

6.2 ODEXConfig Class

The **ODEXConfig** class is defined as:

```
public class ODEXConfig
```

These methods retrieve information and set configuration parameters in the ODEX ETD sub-nodes of the top-level nodes **BatchStatus** and **BatchParameters**, as well as retrieving/setting additional configuration information.

The **ODEXConfig** class extends **java.lang.Object**.

The **ODEXConfig** methods are:

getSendFiles on page 74
getSendFileCount on page 74
getRecvFiles on page 75
getRecvFileCount on page 75
restoreDefault on page 76
getWorkFile on page 76
setWorkFile on page 76
getToWhom on page 79
getOrigLocalNode on page 80
setOrigLocalNode on page 81
getEarlyDate on page 81
setEarlyDate on page 82
getEarlyTime on page 82
setEarlyTime on page 82
getFormat on page 83
setFormat on page 83
getRecSize on page 84
setRecSize on page 84
getPriority on page 85

setPriority on page 85
getUserData on page 86
setUserData on page 86
getMsgType on page 86
setMsgType on page 87
getFileType on page 87
setFileType on page 88
getBatchFile on page 88
setBatchFile on page 88
getNew on page 89
setNew on page 89
getOld on page 90
setOld on page 90
getInboundFile on page 91
setInboundFile on page 91
getInboundVFN on page 91
setInboundVFN on page 92
getInboundBox on page 92
setInboundBox on page 93
getFromWhom on page 93
setFromWhom on page 94
getVirtDate on page 94
setVirtDate on page 94
getVirtTime on page 95
setVirtTime on page 95
getReturnCode on page 96
getSuccess on page 96
getTerminated on page 97
getError on page 97
getFatalError on page 98
getNotAttempted on page 98

getSendFiles

Description

Retrieves information contained in the **SendFiles** ETD node.

Syntax

```
public ODEXSendFileDesc getSendFiles(int index)
```

Parameters

Name	Type	Description
index	Integer	Index number of the desired SendFiles node.

Returns

Object
Information contained in the **SendFiles** node, that is, the current value of **SendFiles**.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

getSendFileCount

Description

Retrieves information contained in the **SendFileCount** ETD node, that is, the number of sent file.

Syntax

```
public int getSendFileCount()
```

Parameters

None.

Returns

Integer
The number of sent files.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

getRecvFiles

Description

Retrieves information contained in the **RecvFiles** ETD node.

Syntax

```
public ODEXRecvFileDesc getRecvFiles(int index)
```

Parameters

Name	Type	Description
index	Integer	Index number of the desired SendFiles node.

Returns

Object
Information contained in the **RecvFiles** node, that is, the current value of **RecvFiles**.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

getRecvFileCount

Description

Retrieves information contained in the **getRecvFileCount** ETD node, that is, the number of received files.

Syntax

```
public int getRecvFileCount()
```

Parameters

None.

Returns

Integer
The number of received files.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

restoreDefault

Description

Sets the ETD nodes with the default values from e*Way's configuration parameter settings.

Syntax

```
public void restoreDefault()
```

Parameters

None.

Returns

Void.

Throws

None.

getWorkFile

Description

Retrieves information contained in the **WorkFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **WorkFile** node, that is, the current value of **WorkFile**..

Throws

None.

setWorkFile

Description

Sets configuration parameters for the **WorkFile** ETD node.

Syntax

```
public void setWorkFile(java.lang.String workfile)
```

Parameters

Name	Type	Description
workfile	Java.lang.String	The file name for the ODEX work file.

Returns

Void.

Throws

None.

getOutboundFile

Description

Retrieves information contained in the **OutboundFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **OutboundFile** node, that is, the current value of **OutboundFile**.

Throws

None.

setOutboundFile

Description

Sets configuration parameters for the **OutboundFile** ETD node.

Syntax

```
public void setOutboundFile(java.lang.String outboundfile)
```

Parameters

Name	Type	Description
outboundfile	Java.lang.String	The file name for the outbound file.

Returns

Void.

Throws

None.

getOutboundVFN

Description

Retrieves information contained in the **OutboundVFN** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **OutboundVFN** node, that is, the current value of **OutboundVFN**.

Throws

None.

setOutboundVFN

Description

Sets configuration parameters for the **OutboundVFN** ETD node.

Syntax

```
public void setOutboundVFN(java.lang.String outboundvfn)
```

Parameters

Name	Type	Description
outboundvfn	Java.lang.String	The virtual file name (VFN) for the outbound file.

Returns

Void.

Throws

None.

getOutboundBox

Description

Retrieves information contained in the **OutboundBox** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **OutboundBox** node, that is, the full path pointing to a local file system directory where outgoing files are located by the ETD.

Throws

None.

setOutboundBox

Description

Sets configuration parameters for the **OutboundBox** ETD node.

Syntax

```
public void setOutboundBox(java.lang.String outboundbox)
```

Parameters

Name	Type	Description
outboundbox	Java.lang.String	The full path pointing to a local file system directory where outgoing files are located by the ETD.

Returns

Void.

Throws

None.

getToWhom

Description

Retrieves information contained in the **ToWhom** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **ToWhom** node, that is, the LocalCode of the default outgoing trading partner.

Throws

None.

setToWhom

Description

Sets configuration parameters for the **ToWhom** ETD node.

Syntax

```
public void setToWhom(java.lang.String towhom)
```

Parameters

Name	Type	Description
towhom	java.lang.String	The LocalCode for the default outbound trading partner ().

Returns

Void.

Throws

None.

getOrigLocalNode

Description

Retrieves information contained in the **OrigLocalNode** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **OrigLocalNode** node, that is, the LocalCode for the originator of the outgoing data.

Throws

None.

setOrigLocalNode

Description

Sets configuration parameters for the **OrigLocalNode** ETD node.

Syntax

```
public void setOrigLocalNode(java.lang.String origlocalnode)
```

Parameters

Name	Type	Description
origlocalnode	java.lang.String	The LocalCode for the originator of the outgoing data.

Returns

Void.

Throws

None.

getEarlyDate

Description

Retrieves information contained in the **EarlyDate** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **EarlyDate** node, that is, the earliest date (in the format YYMMDD) that the outgoing data can be sent to the trading partner.

Throws

None.

setEarlyDate

Description

Sets configuration parameters for the **EarlyDate** ETD node.

Syntax

```
public void setEarlyDate(java.lang.String earlydate)
```

Parameters

Name	Type	Description
earlydate	java.lang.String	The earliest date (expressed as YYMMDD) when ODEX can send the outgoing data.

Returns

Void.

Throws

None.

getEarlyTime

Description

Retrieves information contained in the **EarlyTime** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **EarlyTime** node, that is, the earliest time, in the format HHMMSS, when ODEX can sent scheduled outgoing data to the trading partner.

Throws

None.

setEarlyTime

Description

Sets configuration parameters for the **EarlyTime** ETD node.

Syntax

```
public void setEarlyTime(java.lang.String earlytime)
```

Parameters

Name	Type	Description
earlytime	java.lang.String	The earliest time (expressed as HHMMSS) when ODEX can send the outgoing data.

Returns

Void.

Throws

None.

getFormat

Description

Retrieves information contained in the **Format** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
The format of the data to be sent; its valid values are U (unformatted), T (text), F (fixed-length), or V (variable-length).

Throws

None.

setFormat

Description

Sets configuration parameters for the **Format** ETD node.

Syntax

```
public void setFormat(java.lang.String format)
```

Parameters

Name	Type	Description
format	java.lang.String	The format of the data to be sent out as follows: ▪ U: Unformatted ▪ T: Text ▪ F: Fixed-length ▪ V: Variable-length

Returns

Void.

Throws

None.

getRecSize

Description

Retrieves information contained in the **RecSize** ETD node.

Syntax

```
public int getRecSize()
```

Parameters

None.

Returns

Integer
Information contained in the **RecSize** node, that is, the current value of **RecSize**.

Throws

None.

setRecSize

Description

Sets configuration parameters for the **RecSize** ETD node.

Syntax

```
public void setRecSize(int recsize)
```

Parameters

Name	Type	Description
recsize	Integer	The record size.

Returns

Void.

Throws

None.

getPriority

Description

Retrieves information contained in the **Priority** ETD node.

Syntax

```
public int getPriority()
```

Parameters

None.

Returns

Integer
The information contained in the **Priority** node, that is, the priority for scheduling an outbound file, from 1 (highest) to 9 (lowest).

Throws

None.

setPriority

Description

Sets configuration parameters for the **Priority** ETD node.

Syntax

```
public void setPriority(int priority)
```

Parameters

Name	Type	Description
priority	Integer	The priority for scheduling an outbound file, from 1 (highest) to 9 (lowest).

Returns

Void.

Throws

None.

getUserData

Description

Retrieves information contained in the **UserData** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **UserData** node, that is, the **UserData** value.

Throws

None.

setUserData

Description

Sets configuration parameters for the **UserData** ETD node.

Syntax

```
public void setUserData(java.lang.String userdata)
```

Parameters

Name	Type	Description
userdata	java.lang.String	An eight-character string whose usage is agreed upon between you and your trading partner (), used for exchanging data over OFTP.

Returns

Void.

Throws

None.

getMsgType

Description

Retrieves information contained in the **MsgType** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **MsgType** node, that is, the current value of **MsgType**.

Throws

None.

setMsgType

Description

Sets configuration parameters for the **MsgType** ETD node.

Syntax

```
public void setMsgType(java.lang.String msgtype)
```

Parameters

Name	Type	Description
msgtype	java.lang.String	The value of MsgType.

Returns

Void.

Throws

None.

getFileType

Description

Retrieves information contained in the **FileType** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **FileType** node, that is, the current value of **FileType**.

Throws

None.

setFileType

Description

Sets configuration parameters for the **FileType** ETD node.

Syntax

```
public void setFileType(java.lang.String filetype)
```

Parameters

Name	Type	Description
filetype	java.lang.String	The value of FileType.

Returns

Void.

Throws

None.

getBatchFile

Description

Retrieves information contained in the **BatchFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **BatchFile** node, that is, the current value of **BatchFile**.

Throws

None.

setBatchFile

Description

Sets configuration parameters for the **BatchFile** ETD node.

Syntax

```
public void setBatchFile(java.lang.String batchfile)
```

Parameters

Name	Type	Description
batchfile	java.lang.String	The value of BatchFile.

Returns

Void.

Throws

None.

getNew

Description

Retrieves information contained in the **New** ETD node.

Syntax

```
public boolean getNew()
```

Parameters

None.

Returns

Boolean
Information contained in the **New** node, that is, the current value of **New**.

Throws

None.

setNew

Description

Sets configuration parameters for the **New** ETD node.

Syntax

```
public void setNew(boolean newflag)
```

Parameters

Name	Type	Description
newflag	Boolean	The new value for the New ETD node.

Returns

Void.

Throws

None.

getOld

Description

Retrieves information contained in the **Old** ETD node.

Syntax

```
public boolean getOld()
```

Parameters

None.

Returns

Boolean
Information contained in the **Old** node, that is, the current value of **Old**.

Throws

None.

setOld

Description

Sets configuration parameters for the **Old** ETD node.

Syntax

```
public void setOld(boolean oldflag)
```

Parameters

Name	Type	Description
oldflag	Boolean	The new value for the Old ETD node.

Returns

Void.

Throws

None.

getInboundFile

Description

Retrieves information contained in the **InboundFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **InboundFile** node, that is, the name of the inbound file.

Throws

None.

setInboundFile

Description

Sets configuration parameters for the **InboundFile** ETD node.

Syntax

```
public void setInboundFile(java.lang.String inboundfile)
```

Parameters

Name	Type	Description
inboundfile	java.lang.String	The file name of the inbound file.

Returns

Void.

Throws

None.

getInboundVFN

Description

Retrieves information contained in the **InboundVFN** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **InboundVFN** node, that is, the virtual file name (VFN) of the inbound virtual file (VF).

Throws

None.

setInboundVFN

Description

Sets configuration parameters for the **InboundVFN** ETD node.

Syntax

```
public void setInboundVFN(java.lang.String inboundvfn)
```

Parameters

Name	Type	Description
inboundvfn	java.lang.String	The virtual file name (VFN) for the inbound file.

Returns

Void.

Throws

None.

getInboundBox

Description

Retrieves information contained in the **InboundBox** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **InboundBox** node, that is, the full path pointing to a local file system directory where incoming files are created and stored by the ETD.

Throws

None.

setInboundBox

Description

Sets configuration parameters for the **InboundBox** ETD node.

Syntax

```
public void setInboundBox(java.lang.String inboundbox)
```

Parameters

Name	Type	Description
inboundbox	java.lang.String	The full path pointing to a local file system directory, where incoming files are created and stored by the ETD.

Returns

Void.

Throws

None.

getFromWhom

Description

Retrieves information contained in the **FromWhom** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **FromWhom** node, that is, the LocalCode of the default incoming trading partner.

Throws

None.

setFromWhom

Description

Sets configuration parameters for the **FromWhom** ETD node.

Syntax

```
public void setFromWhom(java.lang.String fromwhom)
```

Parameters

Name	Type	Description
fromwhom	java.lang.String	The LocalCode for the default inbound trading partner.

Returns

Void.

Throws

None.

getVirtDate

Description

Retrieves information contained in the **VirtDate** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String

Information contained in the **VirtDate** node, that is, the date (in format YYMMDD) used to extract a received file with a virtual date tag matching this value; in OFTP, a file is transferred between partners as a virtual file (VF), and every VF bears a virtual date and a virtual time, which in combination identify the file.

Throws

None.

setVirtDate

Description

Sets configuration parameters for the **VirtDate** ETD node.

Syntax

```
public void setVirtDate(java.lang.String virtdate)
```

Parameters

Name	Type	Description
virtdate	java.lang.String	The date (expressed as YYMMDD) used to extract a received file with a virtual date tag matching this value.

Returns

Void.

Throws

None.

getVirtTime

Description

Retrieves information contained in the **VirtTime** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **VirtTime** node, that is, the time used to extract a received file with a virtual time tag matching this value; in OFTP, a file is transferred between partners as a virtual file (VF), and every VF bears a virtual date and a virtual time, which in combination identify the file.

Throws

None.

setVirtTime

Description

Sets configuration parameters for the **VirtTime** ETD node.

Syntax

```
public void setVirtTime(java.lang.String virttime)
```

Parameters

Name	Type	Description
virttime	java.lang.String	The time used to extract a received file with a virtual time tag matching this value.

Returns

Void.

Throws

None.

getReturnCode

Description

Retrieves information contained in the **ReturnCode** ETD node.

Syntax

```
public int getReturnCode()
```

Parameters

None.

Returns

Integer
Information contained in the **ReturnCode** node, that is, the return value sent by the ODEX the batch command processor.

Throws

None.

getSuccess

Description

Retrieves information contained in the **Success** ETD node.

Syntax

```
public boolean getSuccess()
```

Parameters

None.

Returns

Boolean
true if the ODEX return code matches the defined configuration parameter **Success**, and **false** if not.

Throws

None.

---

getTerminated

Description

Retrieves information contained in the **Terminated** ETD node.

Syntax

```
public boolean getTerminated()
```

Parameters

None.

Returns

Boolean
true if the ODEX return code matches the defined configuration parameter **Terminated**, and **false** if not.

Throws

None.

---

getError

Description

Retrieves information contained in the **Error** ETD node.

Syntax

```
public boolean getError()
```

Parameters

None.

Returns

Boolean
true if the ODEX return code matches the defined configuration parameter **Error**, and **false** if not.

Throws

None.

getFatalError

Description

Retrieves information contained in the **FatalError** ETD node.

Syntax

```
public boolean getFatalError()
```

Parameters

None.

Returns

Boolean
true if the ODEX return code matches the defined configuration parameter **FatalError**, and **false** if not.

Throws

None.

getNotAttempted

Description

Retrieves information contained in the **NotAttempted** ETD node.

Syntax

```
public boolean getNotAttempted()
```

Parameters

None.

Returns

Boolean
true if the ODEX return code matches the defined configuration parameter **NotAttempted**, and **false** if not.

Throws

None.

6.3 ODEXETD Class

The **ODEXETD** class is defined as:

```
public class ODEXETD
```

The **ODEXETD** class represents a configurable ODEX job. An ODEX ETD starts and monitors the ODEX process. These methods, as well as attributes (ETD nodes), are exposed through the ODEXETD class to allow a Collaboration to set parameters for scheduling and controlling an ODEX job.

The **ODEXETD** class extends **java.lang.Object**.

The **ODEXETD** methods are:

initialize on page 100
reset on page 100
terminate on page 101
getBatchParameters on page 101
getBatchStatus on page 102
getDataOut on page 102
setDataOut on page 103
getSendFileCount on page 103
getRecvFileCount on page 103
getSendFiles on page 104
getRecvFiles on page 104
getDataIn on page 105
setDataIn on page 105
invokeBatch on page 106
makeCall on page 106
querySent on page 107
schFILE on page 108
schODETT on page 108
schData on page 109
schODETTData on page 109
queryRecv on page 110
rcvFILE on page 110
rcvODETT on page 111
rcvData on page 111
rcvODETTData on page 112

initialize

Description

Initializes (starts) the (ODEX) ETD and overrides the **initialize** method in the class **com.stc.jcsre.SimpleETDImpl**.

Syntax

```
public void initialize(com.stc.common.collabService.
    JCollabController jcollabController,
    java.lang.String key, int mode)
```

Parameters

Name	Type	Description
jcollabController	Object	The JCollabController object used to access IQs, e*Way Connections, and other e*Gate components.
key	java.lang.String	The key to one of the JMsgObject objects.
mode	Integer	The mode for the ETD (IN_MODE, OUT_MODE, or IN_OUT_MODE).

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

reset

Description

Resets (clears) the data content of the (ODEX) ETD and overrides the method **reset** in the class **com.stc.jcsre.SimpleETDImpl**.

Syntax

```
public boolean reset()
```

Parameters

None.

Returns

Boolean
false if the ETD does not have a meaningful implementation of the method reset (in this case, check and correct the ETD); otherwise, **true** means that **reset** has cleared the data content of the ETD.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

terminate

Description

Terminates the (ODEX) ETD, stops the server, and closes and cleans up all the connections. This method overrides the method **terminate** in the class **com.stc.jcsre.SimpleETDImpl**.

Syntax

```
public void terminate()
```

Parameters

None.

Returns

Void.

Throws

com.stc.common.collabService.CollabConnException if there is an external connection problem.

getBatchParameters

Description

Retrieves the information contained in the **BatchParameters** ETD node. This node contains sub-nodes needed for calling ODEX-invoking methods. See the entry under **BatchParameters** under **"ETD Attributes" on page 54** for details on these sub-nodes.

Syntax

```
public ODEXConfig getBatchParameters()
```

Parameters

None.

Returns

Object
Returns the requested **BatchParameters** node with its sub-nodes.

Throws

None.

getBatchStatus

Description

Retrieves the information contained in the **BatchStatus** ETD node. This node contains sub-nodes with ODEX operation status information. See the entry under **BatchStatus** under **"ETD Attributes" on page 54** for details on these sub-nodes.

Syntax

```
public ODEXConfig getBatchStatus()
```

Parameters

None.

Returns

Object
Returns the requested **BatchStatus** node with its sub-nodes.

Throws

None.

getDataOut

Description

Retrieves the outgoing payload buffer **DataOut** ETD node.

Syntax

```
public byte[] getDataOut()
```

Parameters

None.

Returns

Byte array
The blob in the **DataOut** node.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

setDataOut

Description

Sets information for the **DataOut** ETD node.

Syntax

```
public void setDataOut(byte[] data)
```

Parameters

Name	Type	Description
data	Byte array	The blob to be set into the DataOut node.

Returns

Void.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

getSendFileCount

Description

Retrieves the numeric count contained in the **SendFiles** ETD node.

Syntax

```
public int getSendFileCount()
```

Parameters

None.

Returns

Integer
The count of the file-description entries in the **SendFiles** node.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

getRecvFileCount

Description

Retrieves the numeric count contained in the **RecvFiles** ETD node.

Syntax

```
public int getRecvFileCount()
```

Parameters

None.

Returns

Integer
The count of the file-description entries in the **RecvFiles** node.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

getSendFiles

Description

Retrieves the **SendFiles** ETD node.

Syntax

```
public ODEXSendFileDesc getSendFiles(int index)
```

Parameters

Name	Type	Description
index	Integer	The index number of the desired SendFiles node.

Returns

Object
The desired **SendFiles** node with the given index number.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

getRecvFiles

Description

Retrieves the **RecvFiles** ETD node.

Syntax

```
public ODEXRecvFileDesc getRecvFiles(int index)
```

Parameters

Name	Type	Description
index	Integer	The index number of the desired RecvFiles node.

Returns

Object
The desired **RecvFiles** node with the given index number.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

getDataIn

Description

Retrieves the incoming payload buffer **DataIn** ETD node.

Syntax

```
public byte[] getDataIn()
```

Parameters

None.

Returns

Byte array
The blob in the **DataIn** (buffer) node.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

setDataIn

Description

Sets information for the **DataIn** ETD node.

Syntax

```
public void setDataIn(byte[] data)
```

Parameters

Name	Type	Description
data	Byte array	The blob to be set into the DataIn node.

Returns

Void.

Throws

com.stc.common.collabService.CollabDataException if there is a data problem, for example, unmarshaling.

invokeBatch

Description

Invokes an ODEX batch command file. The method **invoke** blocks the requesting Collaboration until the batch operation sends a return or is terminated for whatever reason. The exact operation done is controlled by the batch file. You can check the **ReturnCode** ETD node for the status of the method request.

Syntax

```
public void invokeBatch()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

makeCall

Description

Allows ODEX to make an outgoing call to the trading partner according to the name given in the **ToWhom** ETD node. This method is only necessary when the other partner only receives incoming calls and could have data for you. In such cases, your system must make a call even when there is nothing to send out. You can check the **ReturnCode** node for the status of the method request.

Syntax

```
public void makeCall(boolean enableOutgoingCall)
```

Parameters

Name	Type	Description
enableOutgoingCall	Boolean	true enables the makeCall feature; false disables it.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

---

querySent

Description

Queries the file status information on sent files in ODEX. You can find the result in the **SentFiles** ETD node. Check the **ReturnCode** node for the status of the method call. You can iterate the **SentFiles** node for the status information of sent files that satisfy the query criteria. Use the following parameters to filter the result: **ToWhom**, **VirtDate**, **VirtTime**, **VirtFile**, and **MsgType**. See the entry under **BatchParameters** under **"ETD Attributes" on page 54** for details on these parameters (sub-nodes).

Syntax

```
public void querySent()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

schFILE

Description

Schedules the VF by the name contained in the **OutboundVFN** ETD node. This file's destination is the trading partner whose name is contained in the **ToWhom** node. The actual file is copied from a local file named by **OutboundFile** in the local directory **OutboundBox**. *This file must be in a non-EDI format*. You can check the **ReturnCode** node for the status of the method request.

Syntax

```
public void schFILE()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

schODETT

Description

Schedules the VF by the name contained in the **OutboundVFN** ETD node. This file's destination is the trading partner whose name is contained in the **ToWhom** node. The actual file is copied from a local file named by **OutboundFile** in the local directory **OutboundBox**. *This file must be in an EDI format*. You can check the **ReturnCode** node for the status of the method request.

Syntax

```
public void schODETT()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

schData

Description

Schedules the VF by the name contained in the **OutboundVFN** ETD node. This file's destination is the trading partner whose name is contained in the **ToWhom** node. The actual data is contained in the **DataOut** node. *This file must be in a non-EDI format*. You can check the **ReturnCode** node for the status of the method request.

Syntax

```
public void schData()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

schODETTData

Description

Schedules the VF by the name contained in the **OutboundVFN** ETD node. This file's destination is the trading partner whose name is contained in the **ToWhom** node. The actual data is contained in the **DataOut** node. *This file must be in an EDI format*. You can check the **ReturnCode** node for the status of the method request.

Syntax

```
public void schODETTData()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

queryRecv

Description

Queries the file status information on sent files in ODEX. You can find the result in the **RecvFiles** node. Check the **ReturnCode** node for the status of the method call. You can iterate the **RecvFiles** node for the status information of sent files that satisfy the query criteria. Use the following parameters to filter the result: **ToWhom**, **VirtDate**, **VirtTime**, **VirtFile**, and **MsgType**. See the entry under **BatchParameters** under **"ETD Attributes" on page 54** for details on these parameters (sub-nodes).

Syntax

```
public void queryRecv()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

rcvFILE

Description

Extracts the VF specified by the **InboundVFN** ETD node from the trading partner specified by the **FromWhom** node. The method also saves the VF to the file named by **InboundFile** in the local directory **InboundBox**. *The file must be in a non-EDI format*. You can check the **ReturnCode** node for the status of the method request.

Syntax

```
public void rcvFILE()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

rcvODETT

Description

Extracts the VF specified by the **InboundVFN** ETD node from the trading partner specified by the **FromWhom** node. The method also saves the VF to the file named by **InboundFile** in the local directory **InboundBox**. *The file must be in an EDI format.* You can check the **ReturnCode** node for the status of the method request.

Syntax

```
public void rcvODETT()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

rcvData

Description

Extracts the VF specified by the **InboundVFN** ETD node from the trading partner specified by the **FromWhom** node. The method also saves the VF to a temporary file in the local file system then copies into **DataIn**. *The file must be in a non-EDI format.* You can check the **ReturnCode** node for the status of the method request.

Syntax

```
public void rcvData()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

rcvODETTData

Description

Extracts the VF specified by the **InboundVFN** ETD node from the trading partner specified by the **FromWhom** node. The method also saves the VF to a temporary file in the local file system then copies into **DataIn**. *The file must be in an EDI format.* You can check the **ReturnCode** node for the status of the method request.

Syntax

```
public void rcvODETTData()
```

Parameters

None.

Returns

Void.

Throws

- **com.stc.common.collabService.CollabConnException** if there is an external connection problem.
- **com.stc.common.collabService.CollabDataException** if there is a data problem, for example, unmarshaling.

6.4 ODEXRecvFileDesc Class

The **ODEXRecvFileDesc** class is defined as:

```
public class ODEXRecvFileDesc
```

These methods retrieve information in the ODEX ETD sub-nodes of the top-level node **RecvFiles**. For a complete explanation of this node and its sub-nodes, see **"ETD Attributes" on page 54**.

The **ODEXRecvFileDesc** class extends **java.lang.Object**.

The **ODEXRecvFileDesc** methods are:

getTRDName on page 113
getLocalCode on page 113
getVirtFile on page 114
getTryCount on page 114
getLocalFile on page 115
getWorkFile on page 115
getUserData on page 115
getMsgType on page 116

getEERPSentTime on page 116
getLocalFileTime on page 117
getFileRecvTime on page 117
getFileRecv on page 117
getEERPSent on page 118
getRetriable on page 118
getStatus on page 119
getLastReasonCode on page 119
getAttempted on page 120
getFailed on page 120
getProcessing on page 121
getProcessed on page 121
getForward on page 121

getTRDName

Description

Retrieves information contained in the **TRDName** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **TRDName** ETD node.

Throws

None.

getLocalCode

Description

Retrieves information contained in the **LocalCode** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **LocalCode** ETD node.

Throws

None.

getVirtFile

Description

Retrieves information contained in the **VirtFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **VirtFile** ETD node.

Throws

None.

getTryCount

Description

Retrieves information contained in the **TryCount** ETD node.

Syntax

```
public int getTryCount()
```

Parameters

None.

Returns

Integer
Information contained in the **TryCount** ETD node.

Throws

None.

getLocalFile

Description

Retrieves information contained in the **LocalFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **LocalFile** ETD node.

Throws

None.

getWorkFile

Description

Retrieves information contained in the **WorkFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **WorkFile** ETD node.

Throws

None.

getUserData

Description

Retrieves information contained in the **UserData** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **UserData** ETD node.

Throws

None.

getMsgType

Description

Retrieves information contained in the **MsgType** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **MsgType** ETD node.

Throws

None.

getEERPSentTime

Description

Retrieves information contained in the **EERPSentTime** ETD node.

Syntax

```
public long getEERPSentTime()
```

Parameters

None.

Returns

Long
Information contained in the **EERPSentTime** ETD node.

Throws

None.

getLocalFileTime

Description

Retrieves information contained in the **LocalFileTime** ETD node.

Syntax

```
public long getLocalFileTime()
```

Parameters

None.

Returns

Long
Information contained in the **LocalFileTime** ETD node.

Throws

None.

getFileRecvTime

Description

Retrieves information contained in the **FileRecvTime** ETD node.

Syntax

```
public long getFileRecvTime()
```

Parameters

None.

Returns

Long
Information contained in the **FileRecvTime** node.

Throws

None.

getFileRecv

Description

Retrieves information contained in the **FileRecv** ETD node.

Syntax

```
public boolean getFileRecv()
```

Parameters

None.

Returns

Boolean
Information contained in the **FileRecv** node.

Throws

None.

getEERPSent

Description

Retrieves information contained in the **EERPSent** ETD node.

Syntax

```
public boolean getEERPSent()
```

Parameters

None.

Returns

Boolean
Information contained in the **EERPSent** node.

Throws

None.

getRetriable

Description

Retrieves information contained in the **Retriable** ETD node.

Syntax

```
public boolean getRetriable()
```

Parameters

None.

Returns

Boolean
Information contained in the **Retriable** node.

Throws

None.

getStatus

Description

Retrieves information contained in the **Status** ETD node.

Syntax

```
public int getStatus()
```

Parameters

None.

Returns

Integer
Information contained in the **Status** node.

Throws

None.

getLastReasonCode

Description

Retrieves information contained in the **LastReasonCode** ETD node.

Syntax

```
public int getLastReasonCode()
```

Parameters

None.

Returns

Integer
Information contained in the **LastReasonCode** node.

Throws

None.

getLastReasonTxt

Description

Retrieves information contained in the **LastReasonTxt** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **LastReasonTxt** node.

Throws

None.

getAttempted

Description

Retrieves information contained in the **Attempted** ETD node.

Syntax

```
public boolean getAttempted()
```

Parameters

None.

Returns

Boolean
Information contained in the **Attempted** node.

Throws

None.

getFailed

Description

Retrieves information contained in the **Failed** ETD node.

Syntax

```
public boolean getFailed()
```

Parameters

None.

Returns

Boolean
Information contained in the **Failed** node.

Throws

None.

getProcessing

Description

Retrieves information contained in the **Processing** ETD node.

Syntax

```
public boolean getProcessing()
```

Parameters

None.

Returns

Boolean
Information contained in the **Processing** node.

Throws

None.

getProcessed

Description

Retrieves information contained in the **Processed** ETD node.

Syntax

```
public boolean getProcessed()
```

Parameters

None.

Returns

Boolean
Information contained in the **Processed** node.

Throws

None.

getForward

Description

Retrieves information contained in the **Forward** ETD node.

Syntax

```
public boolean getForward()
```

Parameters

None.

Returns

Boolean
Information contained in the **Forward** node.

Throws

None.

6.5 ODEXSendFileDesc Class

The **ODEXSendFileDesc** class is defined as:

```
public class ODEXRecvFileDesc
```

These methods retrieve information in the ODEX ETD sub-nodes of the top-level node **SendFiles**. For a complete explanation of this node and its sub-nodes, see **"ETD Attributes" on page 54**.

The **ODEXSendFileDesc** class extends **java.lang.Object**.

The **ODEXSendFileDesc** methods are:

getTRDName on page 123
getLocalCode on page 123
getVirtFile on page 123
getLocalFile on page 124
getWorkFile on page 124
getTryCount on page 125
getUserData on page 125
getMsgType on page 125
getEERPRecvTime on page 126
getFileSentTime on page 126
getLocalFileTime on page 127
getFileSent on page 127
getEERPRecv on page 127
getRetriable on page 128
getStatus on page 128
getLastReasonCode on page 129
getLastReasonTxt on page 129
getAttempted on page 129
getFailed on page 130
getProcessing on page 130

getProcessed on page 131
getForward on page 131

getTRDName

Description

Retrieves information contained in the **TRDName** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **TRDName** node.

Throws

None.

getLocalCode

Description

Retrieves information contained in the **LocalCode** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **LocalCode** node.

Throws

None.

getVirtFile

Description

Retrieves information contained in the **VirtFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **VirtFile** node.

Throws

None.

---

getLocalFile

Description

Retrieves information contained in the **LocalFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **LocalFile** node.

Throws

None.

---

getWorkFile

Description

Retrieves information contained in the **WorkFile** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **WorkFile** node.

Throws

None.

getTryCount

Description

Retrieves information contained in the **TryCount** ETD node.

Syntax

```
public int getTryCount()
```

Parameters

None.

Returns

Integer
Information contained in the **TryCount** node.

Throws

None.

getUserData

Description

Retrieves information contained in the **UserData** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **UserData** node.

Throws

None.

getMsgType

Description

Retrieves information contained in the **MsgType** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **MsgType** node.

Throws

None.

getEERPRecvTime

Description

Retrieves information contained in the **EERPRecvTime** ETD node.

Syntax

```
public long getEERPRecvTime()
```

Parameters

None.

Returns

Long
Information contained in the **EERPRecvTime** node.

Throws

None.

getFileSentTime

Description

Retrieves information contained in the **FileSentTime** ETD node.

Syntax

```
public long getFileSentTime()
```

Parameters

None.

Returns

Long
Information contained in the **FileSentTime** node.

Throws

None.

getLocalFileTime

Description

Retrieves information contained in the **LocalFileTime** ETD node.

Syntax

```
public long getLocalFileTime()
```

Parameters

None.

Returns

Long
Information contained in the **LocalFileTime** node.

Throws

None.

getFileSent

Description

Retrieves information contained in the **FileSent** ETD node.

Syntax

```
public boolean getFileSent()
```

Parameters

None.

Returns

Boolean
Information contained in the **FileSent** node.

Throws

None.

getEERPRecv

Description

Retrieves information contained in the **EERPRecv** ETD node.

Syntax

```
public boolean getEERPRecv()
```

Parameters

None.

Returns

Boolean
Information contained in the **EERPRecv** node.

Throws

None.

getRetriable

Description

Retrieves information contained in the **Retriable** ETD node.

Syntax

```
public boolean getRetriable()
```

Parameters

None.

Returns

Boolean
Information contained in the **Retriable** node.

Throws

None.

getStatus

Description

Retrieves information contained in the **Status** ETD node.

Syntax

```
public int getStatus()
```

Parameters

None.

Returns

Integer
Information contained in the **Status** node.

Throws

None.

getLastReasonCode

Description

Retrieves information contained in the **LastReasonCode** ETD node.

Syntax

```
public int getLastReasonCode()
```

Parameters

None.

Returns

Integer
Information contained in the **LastReasonCode** node.

Throws

None.

getLastReasonTxt

Description

Retrieves information contained in the **LastReasonTxt** ETD node.

Syntax

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

Parameters

None.

Returns

java.lang.String
Information contained in the **LastReasonTxt** node.

Throws

None.

getAttempted

Description

Retrieves information contained in the **Attempted** ETD node.

Syntax

```
public boolean getAttempted()
```

Parameters

None.

Returns

Boolean
Information contained in the **Attempted** node.

Throws

None.

getFailed

Description

Retrieves information contained in the **Failed** ETD node.

Syntax

```
public boolean getFailed()
```

Parameters

None.

Returns

Boolean
Information contained in the **Failed** node.

Throws

None.

getProcessing

Description

Retrieves information contained in the **Processing** ETD node.

Syntax

```
public boolean getProcessing()
```

Parameters

None.

Returns

Boolean
Information contained in the **Processing** node.

Throws

None.

getProcessed

Description

Retrieves information contained in the **Processed** ETD node.

Syntax

```
public boolean getProcessed()
```

Parameters

None.

Returns

Boolean
Information contained in the **Processed** node.

Throws

None.

getForward

Description

Retrieves information contained in the **Forward** ETD node.

Syntax

```
public boolean getForward()
```

Parameters

None.

Returns

Boolean
Information contained in the **Forward** node.

Throws

None.

Index

B

basic functions **9**, **11**
batch operation **14**

C

classes, Java
  ODEXConfig **72**
  ODEXETD **99**
  ODEXRecvFileDesc **112**
  ODEXSendFileDesc **122**
CLASSPATH Append From Environment Variable **23**
Classpath Override **23**
Classpath Prepend **22**
Collaboration
  example 1 **62**
  example 2 **63**
components, e*Way overview **16**
configuration overview **11**
configuration parameters
  Connector **27**
  Odette FTP ETD Configuration **28**
  Return Code Configuration **32**

D

design overview **11**
Disable JIT **24**

E

e*Way Connection
  configuration overview **26**
e*Way connection
  configuration parameters **27**
e*Way operation, general **10**
e*Way purpose **10**
EERP handling **69**
ETD attributes **54**
ETD configuration **36**
ETD exposed methods **53**
ETD, drag and drop **71**
external system requirements **16**

G

getAttempted **120**, **129**
getBatchFile **88**
getBatchParameters **101**
getBatchStatus **102**
getDataIn **105**
getDataOut **102**
getEarlyDate **81**
getEarlyTime **82**
getEERPRecv **127**
getEERPRecvTime **126**
getEERPSent **118**
getEERPSentTime **116**
getError **97**
getFailed **120**, **130**
getFatalError **98**
getFileRecv **117**
getFileRecvTime **117**
getFileSent **127**
getFileSentTime **126**
getFileType **87**
getFormat **83**
getForward **121**, **131**
getFromWhom **93**
getInboundBox **92**
getInboundFile **91**
getInboundVFN **91**
getLastReasonCode **119**, **129**
getLastReasonTxt **119**, **129**
getLocalCode **113**, **123**
getLocalFile **115**, **124**
getLocalFileTime **117**, **127**
getMsgType **86**, **116**, **125**
getNew **89**
getNotAttempted **98**
getOld **90**
getOrigLocalNode **80**
getOutboundBox **79**
getOutboundFile **77**
getOutboundVFN **78**
getPriority **85**
getProcessed **121**, **131**
getProcessing **121**, **130**
getRecSize **84**
getRecvFileCount **75**, **103**
getRecvFiles **75**, **104**
getRetriable **118**, **128**
getReturnCode **96**
getSendFileCount **74**, **103**
getSendFiles **104**
getSendfiles **74**
getStatus **119**, **128**
getSuccess **96**

getTerminated **97**
getToWhom **79**
getTRDName **113**, **123**
getTryCount **114**, **125**
getUserData **86**, **115**, **125**
getVirtDate **94**
getVirtFile **114**, **123**
getVirtTime **95**
getWorkFile **76**, **115**, **124**

I

implementation
  introduction **35**
  sample overview **35**
  schema sample 1 **37**
  schema sample 2 **40**
  schema sample 3 **42**
  schema sample 4 **44**
  schema sample 5 **48**
  schema sample 6 **50**
  use with Collaborations **62**
Initial Heap Size **23**
initialize **100**
installation
  files/directories created **20**
  UNIX **19**
  Windows **18**
intended reader **9**
invokeBatch **106**

J

Java Methods **71–131**
Java methods and classes, overview **71**
Java methods, using **71**

M

makeCall **106**
Maximum Heap Size **24**
methods, Java
  getAttempted **120**, **129**
  getBatchFile **88**
  getBatchParameters **101**
  getBatchStatus **102**
  getDataIn **105**
  getDataOut **102**
  getEarlyDate **81**
  getEarlyTime **82**
  getEERPRecv **127**
  getEERPRecvTime **126**
  getEERPSent **118**
  getEERPSentTime **116**
  getError **97**
  getFailed **120**, **130**
  getFatalError **98**
  getFileRecv **117**
  getFileRecvTime **117**
  getFileSent **127**
  getFileSentTime **126**
  getFileType **87**
  getFormat **83**
  getForward **121**, **131**
  getFromWhom **93**
  getInboundBox **92**
  getInboundFile **91**
  getInboundVFN **91**
  getLastReasonCode **119**, **129**
  getLastReasonTxt **119**, **129**
  getLocalCode **113**, **123**
  getLocalFile **115**, **124**
  getLocalFileTime **117**, **127**
  getMsgType **86**, **116**, **125**
  getNew **89**
  getNotAttempted **98**
  getOld **90**
  getOrigLocalNode **80**
  getOutboundBox **79**
  getOutboundFile **77**
  getOutboundVFN **78**
  getPriority **85**
  getProcessed **121**, **131**
  getProcessing **121**, **130**
  getRecSize **84**
  getRecvFileCount **75**, **103**
  getRecvFiles **75**, **104**
  getRetriable **118**, **128**
  getReturnCode **96**
  getSendFileCount **74**, **103**
  getSendFiles **74**, **104**
  getStatus **119**, **128**
  getSuccess **96**
  getTerminated **97**
  getToWhom **79**
  getTRDName **113**, **123**
  getTryCount **114**, **125**
  getUserData **86**, **115**, **125**
  getVirtDate **94**
  getVirtFile **114**, **123**
  getVirtTime **95**
  getWorkFile **76**, **115**, **124**
  initialize **100**
  invokeBatch **106**
  makeCall **106**
  queryRecv **110**
  querySent **107**

rcvData **111**
rcvFILE **110**
rcvODETT **111**
rcvODETTData **112**
reset **100**
restoreDefault **76**
schData **109**
schFILE **108**
schODETT **108**
schODETTData **109**
setBatchFile **88**
setDataIn **105**
setDataOut **103**
setEarlyDate **82**
setEarlyTime **82**
setFileType **88**
setFormat **83**, **84**
setFromWhom **94**
setInboundBox **93**
setInboundFile **91**
setInboundVFN **92**
setMsgType **87**
setNew **89**
setOld **90**
setOrigLocalNode **81**
setOutboundBox **79**
setOutboundVFN **78**
setPriority **85**
setToWhom **80**
setUserData **86**
setVirtDate **94**
setVirtTime **95**
setWorkFile **76**, **77**
terminate **101**

O

ODEX
configuration **17**, **35**, **63**
installation **17**
Plus version **15**, **17**
Professional version **15**, **17**
ODEX and ETD overview **14**
ODEX communication **10**
ODEX components overview **13**
ODEX ETD and batch jobs **61**
ODEX ETD structure **52**
ODEX setup and configuration **10**
ODEX versions overview **14**
ODEX-specific e*Way **10**

Q

queryRecv **110**
querySent **107**

R

rcvData **111**
rcvFILE **110**
rcvODETT **111**
rcvODETTData **112**
recovery, data integrity, and security **69**
reset **100**
restoreDefault **76**

S

sample schemas, importing and running **50**
schData **109**
schFILE **108**
schODETT **108**
schODETTData **109**
setBatchFile **88**
setDataIn **105**
setDataOut **103**
setEarlyDate **82**
setEarlyTime **82**
setFileType **88**
setFormat **83**, **84**
setFromWhom **94**
setInboundBox **93**
setInboundFile **91**
setInboundVFN **92**
setMsgType **87**
setNew **89**
setOld **90**
setOrigLocalNode **81**
setOutboundBox **79**
setOutboundVFN **78**
setPriority **85**
setToWhom **80**
setUserData **86**
setVirtDate **94**
setVirtTime **95**
setWorkFile **76**, **77**
supported operating systems **16**
Suspend Option for Debugging **25**
system requirements **16**

T

terminate **101**