SeeBeyond ICAN Suite

Cobol Copybook Converter User's Guide

Release 5.0.5

Version 20050428144229.

Contents

Chapter 1

Introduction

This user's guide describes how to use the Cobol Copybook Converter to convert input data to COBOL copybook specifications.

What's in This Chapter

1.1 About Cobol Copybooks

Copybooks are common fragments of code that are typically distributed throughout a software application. Functionally similar to the #include file of a C or C++ application, mainframes reference these books, which are usually stored in a source library file, and call structures as needed. When integrating mainframe applications with other platforms, it is necessary to retrieve and generate the data structure of the copybook. Without the copybook's data structure, your disparate applications are not able to communicate with each other and are not capable of transferring data between applications and platforms.

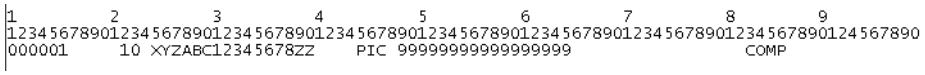
1.1.1 Copybooks with content beyond column 72

The content of the Cobol Copybook, compliant with the IBM Cobol Reference standard, does not go past column 72. To process a copybook that contains data beyond column 72 (e.g., content that is not line numbering or comments, which should be ignored), deselect the **Ignore copybook content beyond column 72** option.

Caution: *It is still possible for a copybook with data beyond the 72th column to process successfully--but not correctly--if the latter option is selected.*

Figure 1 demonstrates copybook content beyond column 72 that may be incorrectly processed.

Figure 1 Copybook content beyond column 72.



If you disable content past column 72, the word "COMP" that begins in column 73 is ignored. Even without this word, the content that appears within the first 72 columns composes a correct (but now misinterpreted) description entry. With the option selected, the entry describes *XYZABC12345678ZZ* as a 18-character alpha-numeric item, using 18 bytes of storage (implicit USAGE is DISPLAY). With the option disabled, the entry describes a 18-digit numeric item using 8 bytes of storage (USAGE is COMP).

1.2 About the Cobol Copybook Converter

The Cobol Copybook Converter converts copybook descriptions, and creates OTDs designed to encapsulate data conforming to the description. The generated OTD is a model, containing a user-friendly abstraction of the data. Cobol Copybook Converter OTDs enable you to handle the data, which is COBOL/EBCDIC in form, as objects of the Java programming language.

The Cobol Copybook Converter reads the copybook specification from a flat file. The converter feature uses the 01 segment of the Cobol copybook as the root node of the OTD. For example, if you are using a CICS eWay, after you have generated an OTD file, the eGate Project can populate the file and present it into the COMM AREA for CICS calls. Similarly, the system can parse the output COMM AREA from CICS into OTDs created by the Cobol Copybook Converter.

Note: *The Cobol Copybook Converter must have valid COBOL syntax to complete an accurate conversion. The Cobol Copybook Converter performs limited syntax validation on an input copybook. To ensure a functional OTD conversion, verify that the copybook supplied to the converter is well-formed* ***and*** *valid.*

1.2.1 Unsupported Features

The following Cobol Copybook features are not supported by the Cobol Copybook Converter:

- **Cobol Copy Statements** — Cobol copy statements that are embedded within the Cobol Copybook are not supported.
- **Usage Pointer** — Usage pointer statements are not supported. To accommodate these elements, you must change the statement to **PIC X(4)**. The Cobol Copybook Converter interprets this and creates a node of the correct length with the subsequent nodes as siblings instead of child nodes.

- **Complete COBOL programs** — these contain non-working storage and non-linkage areas (such as an Environment Division area). The Cobol Copybook Converter processes COBOL files with working-storage and linkage-section record entries only.

1.3 What's New in This Release

This release provides support for Japanese Data.

1.4 About This Document

This guide explains how to install, configure, and operate the SeeBeyond® Integrated Composite Application Network Suite™ (ICAN) Cobol Copybook Converter.

1.4.1 What's in This Document

This guide contains the following information:

- **Chapter 2, "Installing the Cobol Copybook Converter" on page 10** describes how to install the Cobol Copybook Converter and its sample Project.
- **Chapter 3, "Using the OTD Wizard" on page 12** describes how to use the OTD wizard to create and configure Object Type Definitions.
- **Chapter 4, "Locating, Importing, and Using the Sample Projects" on page 26** describes how to use the Cobol Copybook Converter. The chapter also includes procedures for importing and using the Cobol Copybook sample Project.

1.4.2 Scope

This document describes the process of installing, configuring, and running the Cobol Copybook Converter.

1.4.3 Intended Audience

This guide is intended for experienced computer users who have the responsibility of helping to set up and maintain a fully functioning ICAN Suite system. This person must also understand any operating systems on which the ICAN Suite will be installed (Windows, UNIX, and/or HP NonStop Server), and must be thoroughly familiar with Windows-style GUI operations.

1.4.4 Document Conventions

The following conventions are observed throughout this document.

Table 1 Document Conventions

Text	Convention	Example
Names of buttons, files, icons, parameters, variables, methods, menus, and objects	**Bold** text	▪ Click **OK** to save and close. ▪ From the **File** menu, select **Exit**. ▪ Select the **logicalhost.exe** file. ▪ Enter the **timeout** value. ▪ Use the **getClassName()** method. ▪ Configure the **Inbound** File eWay.
Command line arguments, code samples	`Fixed` font. Variables are shown in ***`bold italic`***.	`bootstrap -p` ***`password`***
Hypertext links	**Blue** text	See **Document Conventions** on page 7
Hypertext links for Web addresses (URLs) or email addresses	**Blue underlined** text	**http://www.seebeyond.com** **docfeedback@seebeyond.com**

1.4.5 Screenshots

Depending on what products you have installed, and how they are configured, the screenshots in this document may differ from what you see on your system.

1.5 Related Documents

The following SeeBeyond documents provide additional information about the ICAN Suite:

- *SeeBeyond Integrated Composite Application Network Suite Primer*
- *SeeBeyond ICAN Suite Installation Guide*
- *eGate Integrator User's Guide*
- *eGate Integrator Tutorial*
- *SeeBeyond ICAN Suite Deployment Guide*

1.6 SeeBeyond Web Site

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

http://www.seebeyond.com

1.7 Feedback

If you have any feedback on any SeeBeyond documentation, please send an e-mail to:

docfeedback@seebeyond.com

Chapter 2

Installing the Cobol Copybook Converter

This chapter describes how to install the Cobol Copybook Converter.

What's in This Chapter

2.1 Supported Operating Systems

The Cobol Copybook Converter supports Japanese data and is available for the following operating systems:

- Windows 2000
- Windows XP
- Windows 2000 Japanese
- Windows XP Japanese

2.2 System Requirements

The system requirements for the Cobol Copybook Converter are the same as for eGate Integrator. For information, refer to the *eGate Integrator Installation Guide*. Additional system requirements include:

- The system where the Cobol Cobybook Converter is installed needs approximately 20 MB of free disk space for the application and its configuration, library, and script files.
- Cobol Copybook Converter 5.0.4 requires a 5.0.4 or higher version of the logical host.

2.3 Installing the Product Files

During the eGate Integrator installation process, the Enterprise Manager, a web-based application, is used to select and upload products as .sar files from the eGate installation CD-ROM to the Repository.

The installation process includes installing the following components:

- Installing the Repository
- Uploading products to the Repository
- Downloading components (such as Enterprise Designer and Logical Host)
- Viewing product information home pages

Follow the instructions for installing the eGate Integrator in the *SeeBeyond ICAN Suite Installation Guide*, and include the following steps:

1. After uploading the **eGate.sar** file (using Enterprise Manager) select and upload the following files as described in the *SeeBeyond ICAN Suite Installation Guide*:
   - **CobolCopyBook.sar** (to install the Cobol Copybook Converter)
   - **FileeWay.sar** (to install the File eWay, used in the sample Projects)
   - **CobolCopyBookDocs.sar** (to install the user's guide and the sample Projects)
2. In the Enterprise Manager, click the **DOCUMENTATION** tab.
3. Click **Cobol Copybook Converter**.
4. In the right-hand pane, click **Download Sample**, and select a location for the .zip file to be saved.

   For information about importing and using the sample, refer to **Locating, Importing, and Using the Sample Projects** on page 26.

2.4 After You Install

Upon successful completion of the required files and documentation installation, open the readme file included with the Cobol Copybook Converter documentation SAR file. This file contains information about patches or ESRs that may be required to run the Cobol Copybook Converter or to run project sample files that use the Cobol Copybook Converter.

After ensuring you have all the required Cobol Copybook Converter patches or ESRs, you must then incorporate the Cobol Copybook Converter into an eGate Project and Environment in Enterprise Designer. The next chapters describe how to incorporate the Cobol Copybook Converter into an eGate Project and an eGate Environment, as well as configuring it and building the necessary OTDs.

Chapter 3

Using the OTD Wizard

This chapter describes how to build the business logic for Cobol Copybook Converter Projects. Project business logic is contained in Business Processes for eInsight, and in Collaborations for eGate Integrator used without eInsight.

To build Cobol Copybook Project business logic, you use the Cobol Copybook wizard to create the Cobol Copybook Converter OTD. You then create the Business Processes or Collaborations, and the Connectivity Maps.

What's in This Chapter

3.1 About the Cobol Copybook Wizard

You use the Cobol Copybook wizard to create copybook converter OTDs. These OTDs can then later be used in Collaboration Definitions to create the business logic behind the Collaborations.

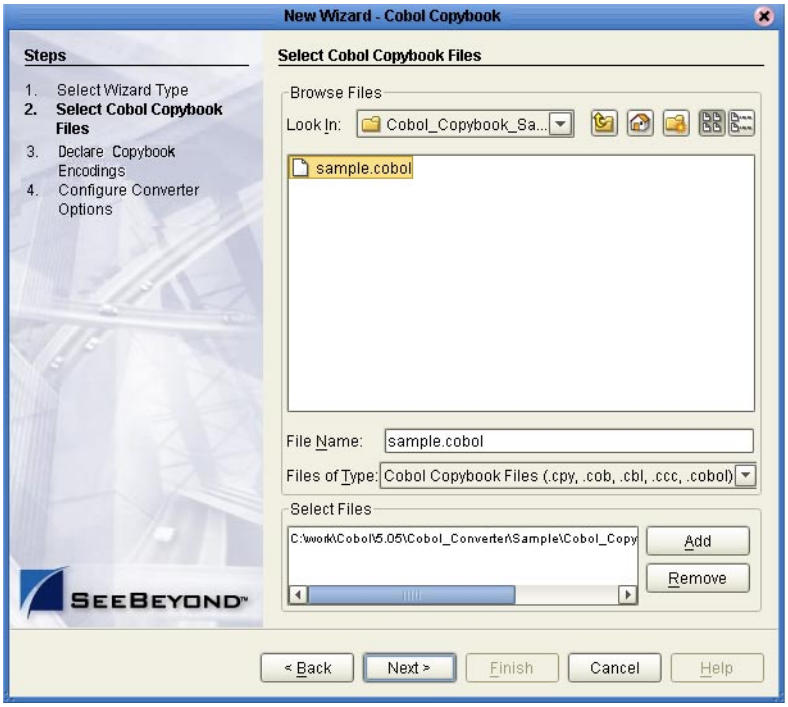
3.2 Creating Cobol Copybook OTDs

You create Cobol Copybook Converter OTDs with the Cobol Copybook wizard in the Enterprise Designer.

To create Cobol Copybook OTDs

1. In the Explorer tab of the Enterprise Designer, right-click **%Project Name% > New > Object Type Definition**. The **New Object Type Definition Wizard** dialog box appears.
2. Click **Cobol Copybook** and click **Next**. The **Select Cobol Copybook Files** page appears.

Figure 2 Cobol Copybook Wizard—Cobol Copybook Selection



3 Browse for the desired Cobol Copybook file and highlight it.

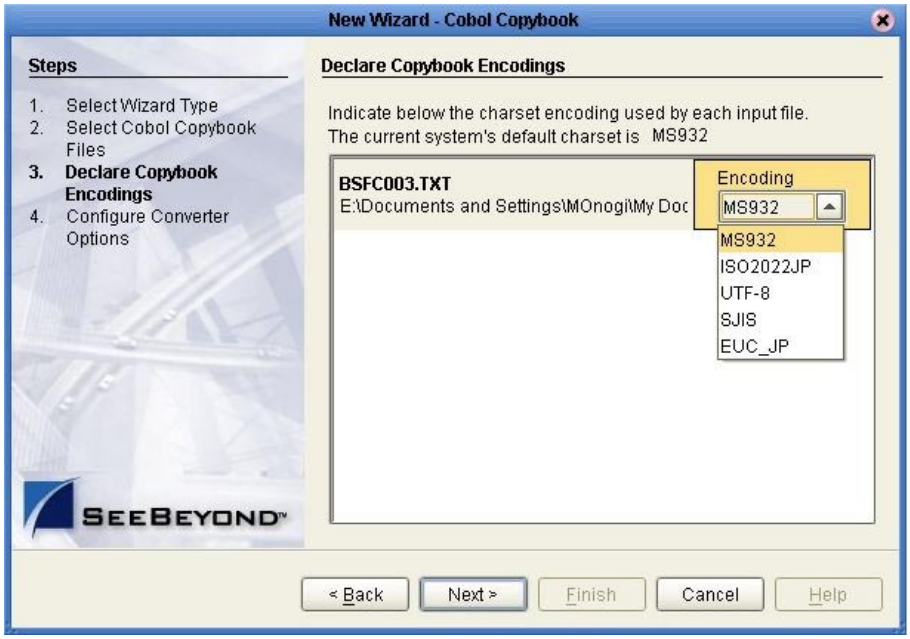
4 Click the **Add** button to include a copybook file in a project.

5 Repeat Steps 3 and 4 for each file to include in the project.

6 To remove a copybook file from the project, highlight the file name in the **Select Files** container and click **Remove**.

7 Click **Next**. The **Declare Copybook Encodings** page appears only if the **Extended Language** Option, in the **Options** menu, has been selected.

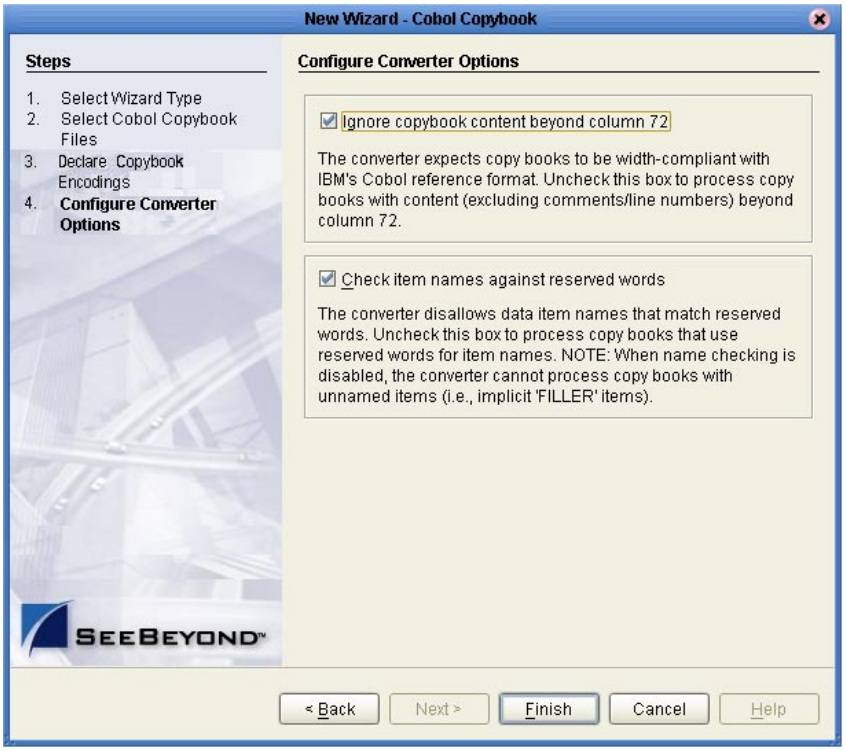
Figure 3 Cobol Copybook Wizard—Declare Copybook Encodings



8 Select the charset encoding, from the drop-down list, to be used by each input (copybook) file.

9 Click **Next**. The **Configure Converter Options** page appears.

Figure 4 Cobol Copybook Wizard—Configure Converter Options



10 Optionally, add/remove checks from boxes to enable/disable options:

- **Ignore copybook content beyond column 72** -- The converter expects copybooks to be width-compliant with IBM's Cobol reference format. Uncheck this box to process books with content (excluding comments/line numbers) beyond column 72. Default: enabled (box is checked).
- **Check Item names against reserved words** -- The converter disallows data item names that match reserved words. Uncheck this box to process copy books that use reserve words for item names. When name checking is disabled, the converter cannot process copy books with unnamed items (i.e., implicit 'FILLER' items). Default: enabled (box is checked).

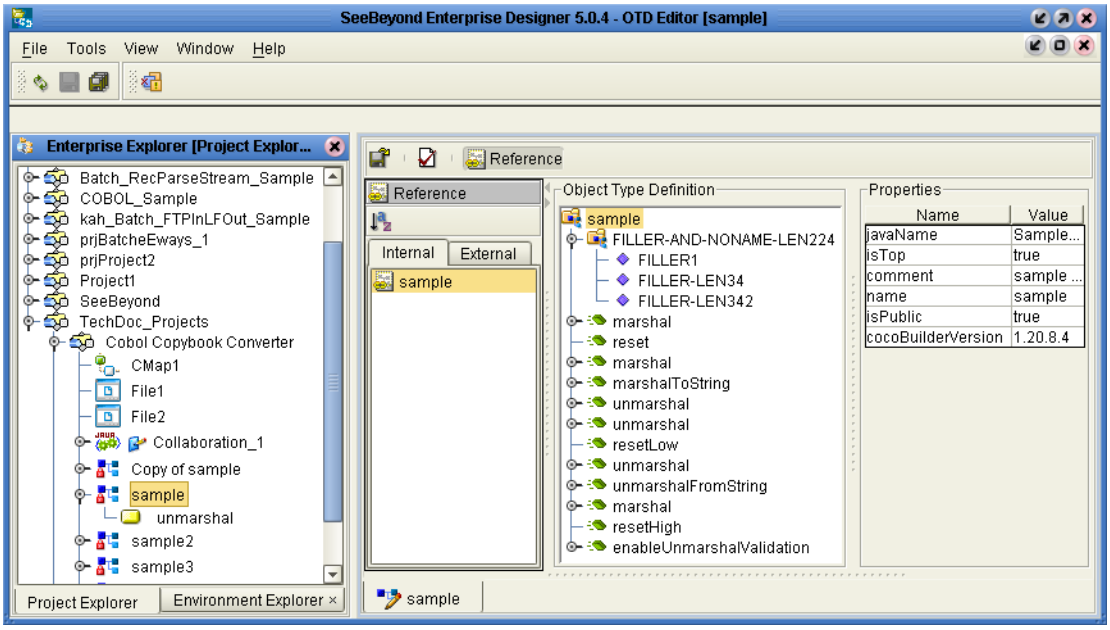
11 Click **Finish**. The **OTD Editor** window appears, displaying the OTD.

The section below describes the cobol copybook methods (operations) that are available for you to use in the source code for the Collaborations or Business Activities.

3.3 Cobol Copybook OTD

When an OTD is built from a copybook file (as is the Sample copybook file) it creates an OTD which contains methods that may be used with the converted contents of the copybook business object.

Figure 5 Sample Copybook OTD



The figure above shows the copybook converter OTD. The OTD has a node for each of the business processes that may be performed on the converted copybook. The unmarshal method allows business processes to flow data into the copybook OTDs and access contents field-by-field.

3.4 Cobol Copybook OTD Methods

The Object Type Definitions (OTDs) created by the Cobol Copybook Converter provide the method that you can use to extract or insert content into OTDs.

3.4.1 OTD Method Guidelines

This section addresses the concerns of global behavior, effects, and assumptions inherent to most methods.

Encoding Behavior for Redefinitions

The unmarshal and marshal methods of a Cobol Copybook OTD (with the exception of the marshalToString and unmarshalFromString) have been reimplemented to heed the OTD structure's data type information. When data flows into or out of the OTD, character set encoding is applied only to the portions of the data that fall on or draw from OTD fields corresponding to items in the Copybook specification that store character data (i.e., usage display items, whether implicitly or explicitly specified). Data for other types of OTD fields are not subject to charset encoding, since these fields are capable of containing binary (non-character) data.

An ambiguity arises when an OTD field, corresponding to a usage display item, is also the object of redefinition(s) in the Copybook. Redefined items may have alternate, multiple storage types, and to deal with such an item, the OTD must decide which one of the multiple definition is in effect at the time of unmarshaling or marshaling, in relation to the available data. The current implementation of Cobol Copybook OTDs resolve this ambiguity by ignoring redefinitions. The decision whether or not to apply encoding to a field is based solely on the item's original storage specification in the Copybook.

DBCS Items

Cobol Copybook OTDs do not support any particular DBCS encoding. When inserted into DBCS nodes, it will not perform inspections of data to determine what specific DBCS encoding is used by character codes or byte sequences (e.g., discerning between a double-byte and a multi-byte encoding). As a consequence:

- DBCS items are represented in the OTD by Java byte array nodes, and their content will be treated as binary "blobs" with the following rules:
  - If content is set directly to a DBCS node, it is stored as-is.
  - If the content is retrieved directly from the DBCS node, the content that was originally set is also returned as-is.
  - If content is unmarshaled via the OTD root, the portion corresponding to the DBCS node is stored as-is. It should be noted however, that correctness of the

aggregate input is the responsibility of the root-level unmarshal call (e.g., do not use unmarshalFromString if the OTD contains DBCS items).

- If the OTD's content is marshaled, the portion corresponding to the DBCS node is yielded as-is, and is excluded from any character set transcoding that character data nodes of the OTD may be subjected to.

- Copybook OTDs will not auto-truncate DBCS data. Since the OTD cannot know the specific DBCS encoding of the data, it cannot correctly truncate it at the correct character boundaries. If the content which is set directly to a DBCS node exceeds the item's width, the OTD will raise an exception.

3.4.2. Root-level Methods

The following methods are the root-level methods provided:

- **"enableUnmarshalValidation(boolean enable)"**
- **"marshal()"**
- **"marshal(String charset)"**
- **"marshal(OtdOutputStream out)"**
- **"marshal(OtdOutputStream out, String charset)"**
- **"marshalToString()"**
- **"reset()"**
- **"resetHigh()"**
- **"resetLow()"**
- **"retrieveEncoding()"**
- **"unmarshal(byte[] in)"**
- **"unmarshal(OtdInputStream in)"**
- **"unmarshal(OtdInputStream in, String charset)"**
- **"unmarshal(byte[] in, String charset)"**
- **"unmarshalFromString(String in)"**
- **"useEncoding(String enc)"**

enableUnmarshalValidation(boolean enable)

Causes the OTD to validate data flow during an unmarshal call.

Syntax

void enableUnmarshalValidation(boolean enable)

Throws

none.

Examples

```
// enable validation during unmarshal
// call to unmarshal may raise an exception if content is not compatible
byte[] content = ...
OTD_1.enableUnmarshalValidation(true);
OTD_1.unmarshal(content);

// disable validation during unmarshal
// call to unmarshal will not raise data-related exceptions
// instead, data-related exceptions may/will occur when
// accessing specific nodes with invalid data.
byte[] content = ...
OTD_1.enableUnmarshalValidation(false);
OTD_1.unmarshal(content);
```

marshal()

Serializes the OTD's content as an array of bytes. The content is encoded with the OTD's current encoding, which is the encoding specified when data was last unmarshaled (see setEncoding() and unmarshal() for additional details). If no data was unmarshaled prior to a marshal call, then the OTD defaults to EBCDIC CP037 encoding. If the OTD content is incompatible with the current encoding (this can happen when data was unmarshaled with a different encoding that the current one), a com.stc.otd.runtime.MarshalException occurs.

Syntax

```
byte [] marshal()
```

Throws

MarshalException, IOException, UnsupportedEncodingException

Examples

```
// populate OTD and marshal entire content in EBCDIC
OTD_1.setField1(...
OTD_1.setField2(...
...
byte[] output = OTD_1.marshal();

// write ASCII data to OTD
// edit some fields
```

```
// marshal OTD data (still ASCII)
byte[] content = ...
OTD_1.unmarshal(content, "US-ASCII");
OTD_1.setField9(...
OTD_1.setField10(...
byte[] output = OTD_1.marshal();


// write ASCII data to OTD
// edit some fields
// marshal OTD data using different encoding (may fail depending on data)
byte[] content = ...
OTD_1.unmarshal(content, "US-ASCII");
OTD_1.setField9(...
OTD_1.setField10(...
OTD_1.useEncoding("CP277");
byte[] output = OTD_1.marshal();
```

marshal(String charset)

This method serializes the content of the OTD as an array of bytes. The content is encoded using the user-specified character set. The encoding specified in this call acts as a temporary override to the OTD's current encoding, but does not become the current encoding (see setEncoding and unmarshal documentation for information). If the OTD content is not compatible with the current encoding (this can happen if data was unmarshaled using an encoding different from the current one), com.stc.otd.MarshalException occurs. If the specified *charset* value does not name a supported character set, a java.io.UnsupportedEncodingException is generated.

Syntax

byte[] marshal(String charset)

Throws

MarshalException, IOException, UnsupportedEncodingException

Examples

```
byte[] content = cocoOtd.marshal("cp037"); // retrieve OTD content as EBCDIC data
byte[] content = cocoOtd.marshal("US-ASCII"); // retrieve OTD content as ASCII data
```

marshal(OtdOutputStream out)

This method serializes the content of the OTD and writes it to the supplied output stream object. The output is encoded using the same user-specified encoding used when the data was last unmarshaled (see setEncoding and unmarshal documentation

for additional details). If no data was unmarshaled prior to the call to marshal, then EBCDIC CP037 encoding is used. If the OTD content is not compatible with the current encoding (this can happen if the data was unmarshaled using an encoding different from the current one), com.stc.otd.MarshalException occurs. A java.io.IOException is generated if an output error occurs in attempting to write data to the stream object.

Syntax

void marshal(OtdOutputStream out)

Throws

MarshalException, IOException, UnsupportedEncodingException

marshal(OtdOutputStream out, String charset)

This method flows data out from the OTD to the supplied stream object, using the specified charset encoding. The given encoding acts as a temporary override to the OTD's current encoding, it does not become the current encoding (see setEncoding and unmarshal documentation for information).

If the specified charset is not compatible with the OTD content (this can happen when the data was unmarshaled to the OTD using a different encoding), com.stc.otd.runtime.MarshalException occurs. If the encoding is not supported or recognized, java.io.UnsupportedEncodingException is generated.

Syntax

void marshal(OtdOutputStream stream, String charset)

Throws

MarshalException, IOException, UnsupportedEncodingException

marshalToString()

This method serializes the content of the OTD to a String object. The String is created by decoding the byte data with the OTD's current encoding, which is the encoding specified when data was last unmarshaled (see setEncoding and unmarshal documentation for additional details). If no data was unmarshaled prior to a marshal call, then the OTD defaults to EBCDIC CP037 encoding. Only use this method with copybook OTDs built from copybooks comprised solely of usage display entries. Using this method on OTDs designed to hold binary data (e.g., packed decimal, internal decimal) may invalidate the data, because portions of the binary content may not have a suitable mapping to UTF-8. A java.io.UnsupportedEncodingException may occur if the current encoding (i.e., the encoding used by the last unmarshal call) is not capable of encoding the data. This is possible because certain charset encodings in Java are not two-way encodings (encodings that can decode or encode, but not both).

Syntax

String marshalToString()

Throws

MarshalException, IOException, UnsupportedEncodingException

reset()

Initializes the storage space of the OTD as follows:

- alphanumeric fields (PIC X) - blank spaces (EBCDIC value 0x40)
- numeric fields (PIC 9) - binary zero
- packed decimal fields - signed-trailing packed binary zero

Syntax

void reset()

Throws

none

resetHigh()

Initializes the entire storage space of the OTD to high bit values; each byte is initialized to 0xFF.

Syntax

void resetHigh()

Throws

none

resetLow()

Initializes the OTD storage space to low bit values; each byte is initialized to 0x0.

Syntax

void resetLow()

Throws

none

retrieveEncoding()

Returns the canonical name of the current OTD encoding. The default current OTD encoding is "CP037" until it is changed by a successful useEncoding call, or by a call to one of the encoding-specifiable unmarshal methods. The canonical name may differ from the one used previously to set the current encoding. See the Java 2 API documentation for java.nio.charset. Charset for more information.

Syntax

String retrieveEncoding()

Throws

none

unmarshal(byte[] in)

Deserializes the given input into an internal data tree. Data flowed to the OTD using this method must use EBCDIC CP037 encoding. This method sets the OTD's current encoding to EBCDIC CP037, which is used when data is subsequently marshaled without an overriding encoding; e.g., as allowed in a marshal(OtdOutputStream, String) call.

Syntax

void unmarshal(byte[] in)

Throws

UnmarshalException, IOException

unmarshal(OtdInputStream in)

This method populates the OTD using the supplied OtdInputStream object as the data source. The supplied object must be an opened stream with available data. A com.stc.otd.runtime.UnmarshalException is generated if the data obtained from the stream is incompatible with the OTD, and a java.io.IOException is generated if any other input error occurs in attempting to read data from the stream object. The stream object must flow data encoded in EBCDIC CP037. This method sets the OTD's current encoding to EBCDIC CP037, which is used when data is subsequently marshaled without overriding encoding; e.g., as allowed in a marshal (OtdOutputStream, String) call.

Syntax

void unmarshal(OtdInputStream in)

Throws

UnmarshalException, IOException

unmarshal(OtdInputStream in, String charset)

This method flows data to the OTD from the supplied Stream object. The stream must be open and have available data. The charset argument specifies the encoding of the stream data. The specified encoding becomes the current encoding of the OTD and is used when data is subsequently marshaled without overriding encoding; e.g., as allowed in a marshal(OtdOutputStream, String) call.

If the stream data is incompatible with the OTD, a com.stc.otd.runtime.UnmarshalException is generated. If the stream data cannot be read, a java.io.IOException is generated. If the charset value does not name a supported charset, or if it names a supported charset with one-way encoding (capable of decoding or encoding, but not both), a java.io.UnsupportedEncodingException is generated.

Syntax

void unmarshal(OtdInputStream in, String charset)

Throws

UnmarshalException, IOException, UnsupportedEncodingException

unmarshal(byte[] in, String charset)

This method populates the OTD using the data supplied in the byte array in. The charset argument specifies the encoding of the given data. The specified encoding becomes the current encoding of the OTD, and is used when data is subsequently marshaled without an overriding encoding; e.g., as allowed in a marshal (OtdOutputStream, String) call. If the specified charset value does not name a supported character set or names a supported charset with one-way encoding (one that can decode or encode, but not both), a java.io.UnsupportedEncodingException is generated.

Syntax

void unmarshal(byte[] in, String charset)

Throws

UnmarshalException, IOException, UnsupportedEncodingException

Examples

```
byte[] bytes = ...
cocoOtd.unmarshal(bytes, "cp037"); // Interpret bytes content as EBCDIC data
cocoOtd.unmarshal(bytes, "US-ASCII"); // Interpret bytes content as ASCII data
```

unmarshalFromString(String in)

This method populates the OTD using the specified String object as the input source. This method is useful only to unmarshal wholly character data to copybook OTDs comprised solely of character-data records (entries specified implicitly or explicitly as USAGE DISPLAY). The current OTD encoding (see setEncoding and unmarshal document for additional details) is used to encode the String's bytes.

Syntax

void unmarshalFromString(String in)

Throws

UnmarshalException, IOException

useEncoding(String enc)

Use this method to designate a particular encoding to be used as the OTD's current encoding. The current OTD encoding is used when the OTD is marshaled without an overriding encoding, which is permitted for the marshal (OtdOutputStream, String) method.

An OTD's current encoding is initially EBCDIC (CP037) when it is instantiated. There are two ways to change it:

1. Unmarshaling the data, whereby the data's stated encoding becomes the current encoding.
2. Using this method to specify it.

Changing the encoding thru the use of this method causes reset() to be subsequently (and automatically) called, causing the OTD's existing content to be erased. This behaviour exists to avoid situations where data, successfully unmarshaled with one charset, fails to marshal under a different charset, due to the absence of codepoint mappings between the two encodings. Use the marshal(String) method when data, which flowed in using a charset, must then be flowed out with a different charset.

If the specified encoding is the same as the current OTD encoding, the call returns without affecting the OTD's state (i.e., reset() is not called) and the data and current encoding will remain unchanged.

If the specified encoding is not supported, or is not a two-way encoding (one that can decode or encode, but not both), a java.io.UnsupportedEncodingException is thrown.

Syntax

void useEncoding(String enc)

Throws

UnsupportedEncodingException

3.4.3. Non-Root Methods

Every leaf node in a Cobol Copybook OTD represents an elementary item in the Copybook source. For every given leaf node, the OTD provides “getter” and “setter” methods of which the return type and input types depend on the data type and usage type specified in the copybook for the elementary item to which the node corresponds.

For a given non-repeating leaf node named Datum, the following method forms are provided, where *T* is determined from the follow table.

- *T* getDatum()
- void setDatum(*T*)

Usage Types	Display	COMP or COMP-4	COMP-1	COMP-2	COMP-3	COMP-5	INDEX
Data Types							
Alphabetic For example: PIC AAA	String						
Alphanumeric For example: PIC X9	String		String	String			

Usage Types	Display	COMP or COMP-4	COMP-1	COMP-2	COMP-3	COMP-5	INDEX
Data Types							
Alphanumeric edited For example: PIC XB9	String						
Numeric edited For example: PIC ZZZ99	String						
DBCS For example PIC GGBGG	byte[]						
External floating point For example: PIC +9V99E+99	BigDecimal						
Numeric integer (9 digits or less)	int	int			int		int
Numeric floating point (COMP-1 or COMP-2 items)	BigDecimal						
Numeric Integer (10 to 18 digits)	long	long			long	long	
Numeric integer (19 digits or more)	BigDecimal	Big Decimal			Big Decimal	Big Decimal	

For repeating leaf nodes, these two alternative methods are provided:

- *T* getDatum(int i)
- void setDatum(int *i*, *T*)

where *i* is expected to be a value from 0, representing the ordinal of the desired repetition instance, and where *T* is determined as previously described.

Chapter 4

Locating, Importing, and Using the Sample Projects

This chapter describes how to use the Cobol Copybook Converter to convert COBOL copybooks into OTDs. It also includes how to use the sample that comes with the Cobol Copybook Converter.

What's in This Chapter

4.1 About the Sample Projects

The Cobol Copybook Converter utility includes the following sample Projects that you can import. This enables you to see how ICAN Projects can work with Cobol copybooks.

- Cobol_BPEL_Sample for use with eInsight/eGate
- Cobol_Copybook_Sample for use with eGate

Each Project contains the following:

- Input data
- Connectivity Maps
- Collaborations
- Business Processes

Version Support

Consult the *Cobol_Converter.txt* file provided in the **CobolCopyBookDocs.sar** file for specific ESR requirements (if they exist) to import or run each of the sample projects.

4.1.1. Cobol_BPEL_Sample

The Cobol BPEL Sample is for use with eInsight. It provides an implementation of the Cobol Copybook Converter that uses the newly supported BPEL functionality. The unzipped Cobol_BPEL_Sample.zip project consists of the following files:

- *Cobol_BPEL_Sample.zip* - the project that needs to be imported to an eGate/eInsight installation.
- *qan3glr1.cobol* - the cobol copybook file used for the conversion to create the OTD.
- *inputcobolBPEL.txt* - the input file that the sample project requires when it is run.
- *CobolBPELoutput1.dat* - contains the expected output when the project is executed with the given input file.

4.1.2. Cobol_Copybook_Sample

The Cobol Copybook Sample is designed for use with eGate. It provides an implementation of the Cobol Copybook Converter that uses Java Collaborations to execute the desired business logic. The unzipped Cobol_Copybook_Sample.zip project consists of the following files:

- *EBCDICtoASCII_Sample.zip* - the project that needs to be imported to an eGate installation.
- *misco1a.cobol* - the cobol copybook file used for the conversion to create the OTD.
- *input.txt* - the input file that the sample project requires when it is run.
- *COBOLoutput1.dat* - contains the expected output when the project is executed with the given input file.

This sample Project converts EBCDIC input data to the format specified in the copybook. The input data is provided by a File eWay. This data is read into a Cobol Copybook OTD generated from the same copybook. The Collaboration shows the use of the Cobol Copybook OTD to retrieve the EBCDIC data as Java Strings for concatenation and forwards the output to an outbound File eWay. The resulting file output is the ASCII translation of the original input data.

4.2 Locating the Sample Projects

The sample Projects are included in the **CobolCopyBookDocs.sar**. You upload this file separately from the Cobol Copybook sar file during installation. For information, refer to **"Installing the Product Files" on page 11**.

Once you have uploaded the **CobolCopyBookDocs.sar** to the Repository and you have downloaded the sample Projects (**Cobol_Copybook_Sample.zip**) using the **DOCUMENTATION** tab in the Enterprise Manager, the sample resides in the folder specified during the download.

4.3 Importing the Sample Projects

This section describes the process required to import each of the sample projects into the Enterprise Designer.

To import the eGate sample

1. Unzip the **Cobol_Converter_Sample.zip** file to a temporary directory.

   For information about locating this file, refer to **Locating the Sample Projects** on page 27.
2. In the Project Explorer tab of the Enterprise Designer, right-click the Repository and click **Import Project**. The **Select File to Import** dialog box appears.
3. Browse to the temporary directory.
4. Double-click **EBCDICtoASCII_Sample.zip**. The **File Destination** dialog box appears.
5. Click **Import to a new Project**, enter the name of the Project, and click **OK**.
6. When the import has successfully completed, right-click the Repository and click **Refresh All from Repository**.

The Project is now imported. Before you deploy and run the Project, do the following:

- Configure the Cobol Copybook Converter for the correct input and output directories. Refer to the documentation for more information.
- Create an Environment and Deployment Profile, and run the Project. Refer to the *eGate Integrator User's Guide* for more information.

To import the eInsight sample

1. Unzip the **Cobol_Converter_Sample.zip** file to a temporary directory.

   For information about locating this file, refer to **Locating the Sample Projects** on page 27.
2. In the Project Explorer tab of the Enterprise Designer, right-click the Repository and click **Import Project**. The **Select File to Import** dialog box appears.
3. Browse to the temporary directory.
4. Double-click **Cobol_BPEL_Sample.zip**. The **File Destination** dialog box appears.
5. Click **Import to a new Project**, enter the name of the Project, and click **OK**.
6. When the import has successfully completed, right-click the Repository and click **Refresh All from Repository**.

The Project is now imported. Before you deploy and run the Project, do the following:

- Configure the Cobol Copybook Converter for the correct input and output directories. Refer to the relevant documentation for more information.
- Create an Environment and Deployment Profile, and run the Project. Refer to the *eGate Integrator User's Guide* for more information.

4.4 Running the Sample Projects

The sample Projects do not include the eGate Environments, Deployment Profiles, and the physical configuration needed for the files to deploy the Projects. The steps required to run the sample projects include:

1. Create an Environment Profile as described in the *eGate Integrator User's Guide.*
2. Create a Deployment Profile as described in the *eGate Integrator User's Guide.*
3. Run the Project as described in the *eGate Integrator User's Guide.*

4.5 Building Cobol Copybook Business Logic with eInsight

This section describes how to build the business logic with eInsight:

To see an example of Business Processes and Connectivity Maps, import the Cobol_BPEL_Sample sample Project as described in **Locating, Importing, and Using the Sample Projects** on page 26.

4.5.1. Adding a New Business Process

To add Business Processes

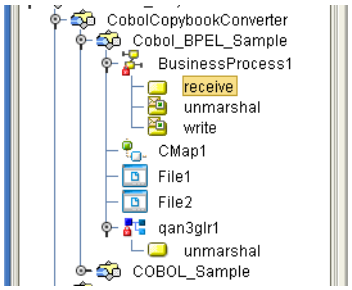
- In the **Project Explorer** tab of the Enterprise Designer, right-click the Project for which you intend to create a Business Process, click **New**, and then **Business Process**.

4.5.2. Building the Business Processes

To build Business Processes

1. In the **Project Explorer** tab of the Enterprise Designer, expand the OTD. This displays the OTD methods.

Figure 6 Cobol Copybook OTD Methods



2 Drag the *unmarshal* Cobol Copybook OTD method to the Business Process Designer canvas.

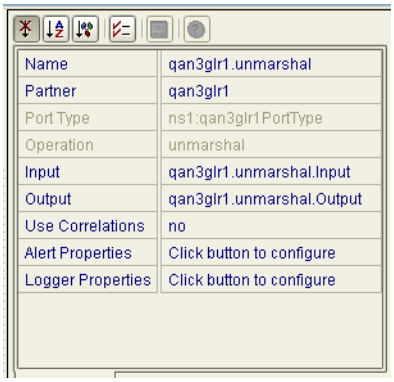
3 Expand the **SeeBeyond**, **eWays**, **File**, and **FileClient** folders in the **Project Explorer** tab.

4 Drag the *write* method to the Business Process Designer canvas.

5 Drag the *receive* method to the Business Process Designer canvas.

6 Click the *unmarshal* Business Activity and click **Show Properties**. The **Properties** dialog box appears.

Figure 7 Unmarshal Properties



7 Click the **Input** box and select **%OTDNAME%.unmarshal.input**.

8 Click the **Output** box and select **%OTDNAME%.unmarshal.output**.

9 Configure all other Activities by highlighting the Activity and clicking **Show Properties**. Refer to **Cobol Copybook OTD Methods** on page 16 for Business Operations syntax.

10 Link all components as described in *eInsight Business Process Manager User's Guide*.

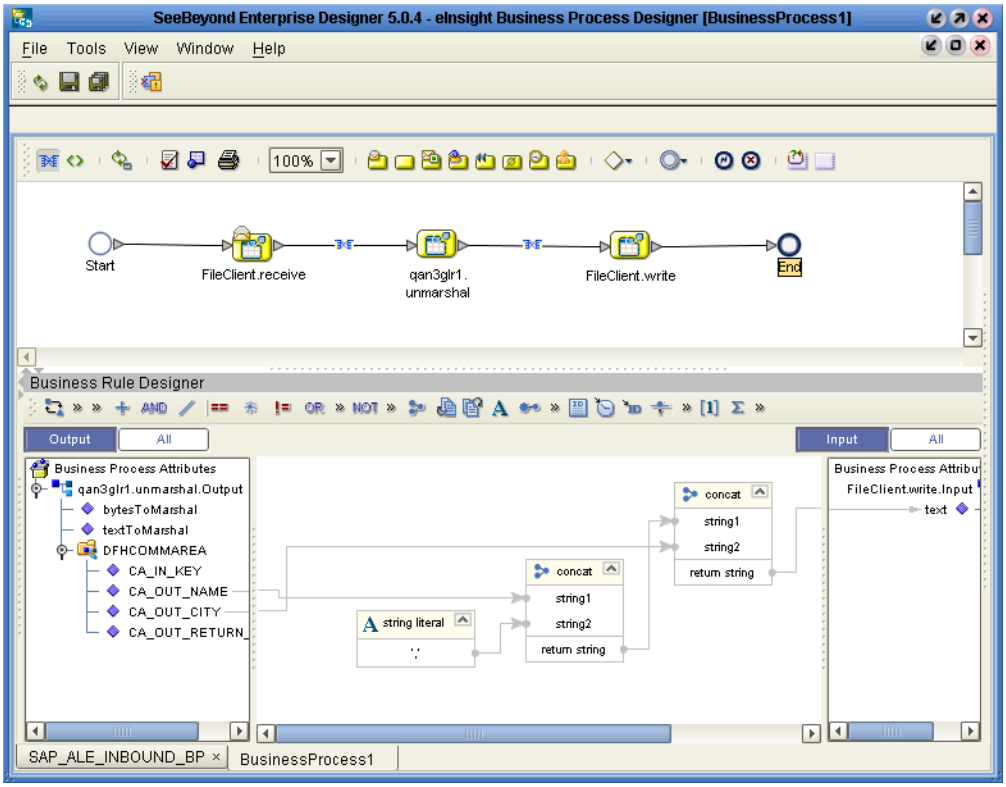
11 To create data mappings, right-click the link between the Activities and click **Add Business Rule**.

12 In the **Business Rule Editor** window, create the code and the data mappings. For details, refer to the *eInsight Business Process Manager User's Guide*.

Figure 8 shows an example of an Business Process including the data mapping in the **Business Rule Editor** window. To explore the business logic design for an actual

Project, import the Cobol_BPEL_Sample Project as described in **Importing the Sample Projects** on page 28.

Figure 8 Business Process and Data Mapping



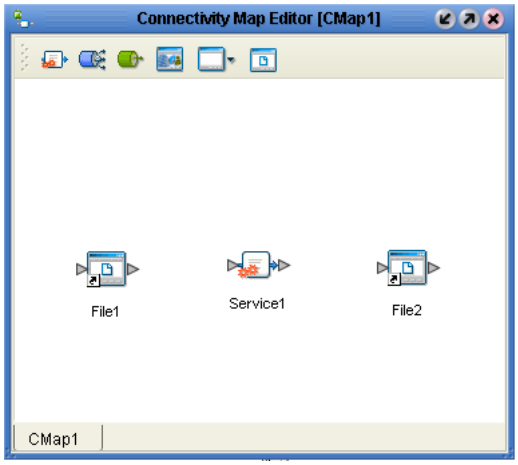
4.5.3. Creating the Connectivity Map

The procedure below describes how to create the Connectivity Map for the COBOL copybook conversion Project.

To create the Connectivity Map

1. In the Project Explorer tab of the Enterprise Designer, right-click the copybook conversion Project, click **New**, and click **Connectivity Map**. A blank Connectivity Map appears.
2. Click the **eWay** icon and click the eWay type.
3. Drag the **eWay** icon to the Connectivity Map to create the inbound eWay.
4. Drag the **Service** icon to the Connectivity Map.
5. Click the **eWay** icon and click the eWay type.
6. Drag the **eWay** icon to the Connectivity Map to create the outbound eWay. The Connectivity Map looks similar to the figure below.

Figure 9 COBOL Copybook Conversion Connectivity Map



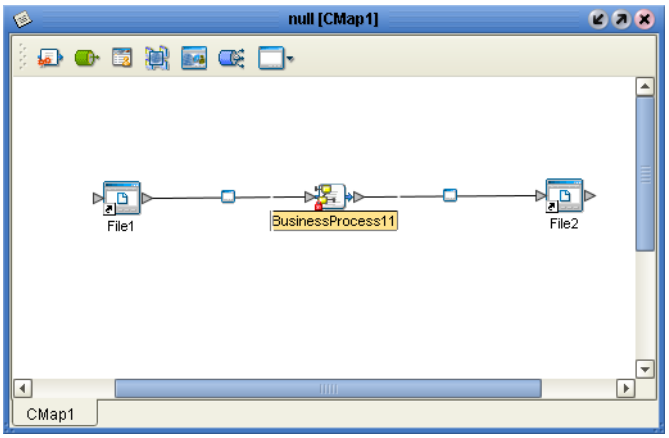
4.5.4. Binding the Business Process and eWays

Once you have created the Business Process and its business logic, you can bind the new business process to the Service, and then connect the Business Process to the eWays.

To bind the Business Process and eWays

1 From the Project Explorer of the Enterprise Designer, drag the newly-created Business Process to the Service in the Connectivity Map as shown below.

Figure 10 Binding the Business Process and Service

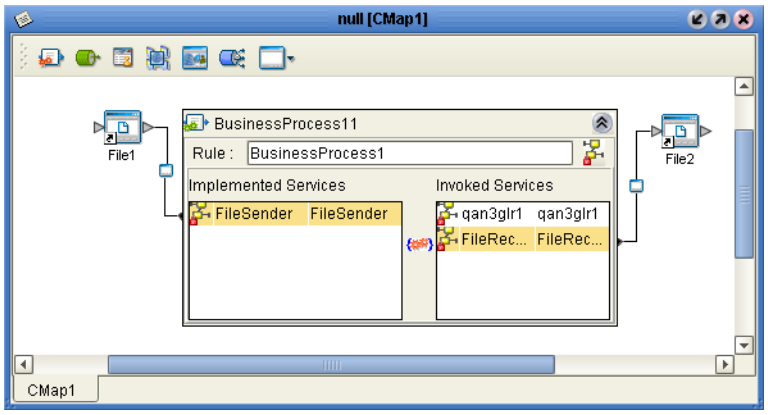


2 Double-click the **Service** icon. The **Service1** window appears.

3 Drag the input service to the inbound eWay. For example, for a File eWay, the input service is **FileSender**.

4 Drag the output service to the outbound eWay as shown below.

Figure 11 Connecting the Business Process to the eWays



5 Close the **BusinessProcess11** window and click **Save**.

Completing the Project

Once you have completed the Connectivity Map binding, you must do the following to finish the Project:

1 Configure the File eWays as described in the eWay documentation.

2 Create an Environment and Deployment Profile and run the Project as described in the *eGate Integrator User's Guide*.

4.6 Building Cobol Copybook Business Logic with eGate

This section describes how to use the Cobol Cobybook Converter to convert files using COBOL copybooks. As a quick start, the following list provides an overview of the steps taken:

1 Create an eGate Project (if necessary).

2 Create a Cobol Object Type Definition (OTD) that indicates the Collaboration will receive data, use the supplied COBOL copybook file to convert it, and forward the converted data to an output eWay - see **Creating a COBOL Copybook Project and OTD** on page 34.

3 Create a Connectivity Map with an inbound eWay, a Collaboration, and an outbound eWay - see **Creating the Connectivity Map** on page 34.

4 Creating the Collaboration Definition and its business logic - see **Creating the Collaboration Definition** on page 35 and **Building Collaboration Definitions** on page 37.

5 Bind the newly-created Cobol OTD to the Collaboration and connect the Collaboration to the eWays - see**Binding the Collaboration Definition and eWays** on page 42.

6 Create an eGate Environment - see **Running the Sample Projects** on page 29.

7 Create a Deployment Profile - see **Running the Sample Projects** on page 29.

8 Deploy and run the Project - see **Running the Sample Projects** on page 29.

To see an example of Cobol Copybook Converter Collaborations and Connectivity Maps, import the EBCDICtoASCII sample Project as described in **Locating, Importing, and Using the Sample Projects** on page 26.

4.6.1. Creating a COBOL Copybook Project and OTD

See **Creating Cobol Copybook OTDs** on page 12 for details about how to create a Cobol Copybook Converter OTD.

4.6.2. Creating the Connectivity Map

The procedure below describes how to create the Connectivity Map for the COBOL copybook conversion Project.

To create the Connectivity Map

1 In the Project Explorer tab of the Enterprise Designer, right-click the copybook conversion Project, click **New**, and click **Connectivity Map**. A blank Connectivity Map appears.

2 Click the **eWay** icon and click the eWay type.

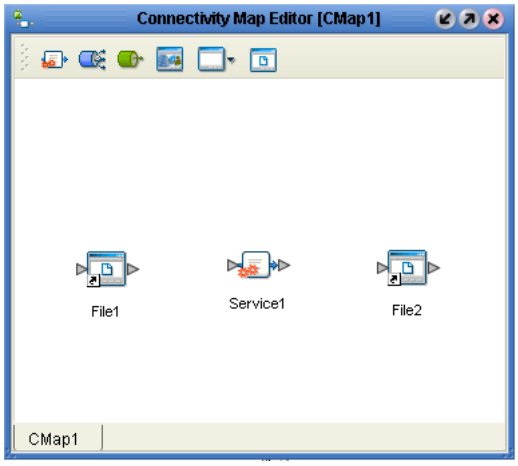
3 Drag the **eWay** icon to the Connectivity Map to create the inbound eWay.

4 Drag the **Service** icon to the Connectivity Map.

5 Click the **eWay** icon and click the eWay type.

6 Drag the **eWay** icon to the Connectivity Map to create the outbound eWay. The Connectivity Map looks similar to the figure below.

Figure 12 COBOL Copybook Conversion Connectivity Map

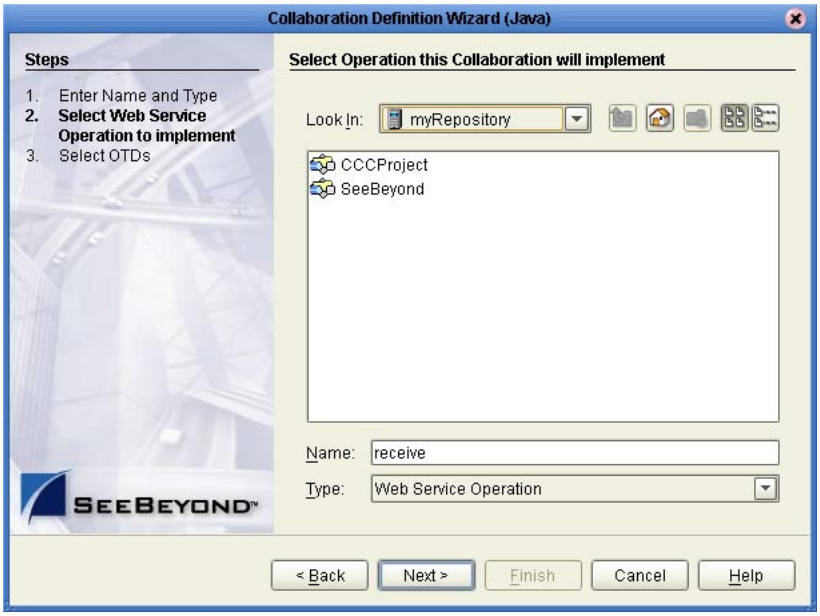


4.6.3. Creating the Collaboration Definition

To create the Collaboration Definition

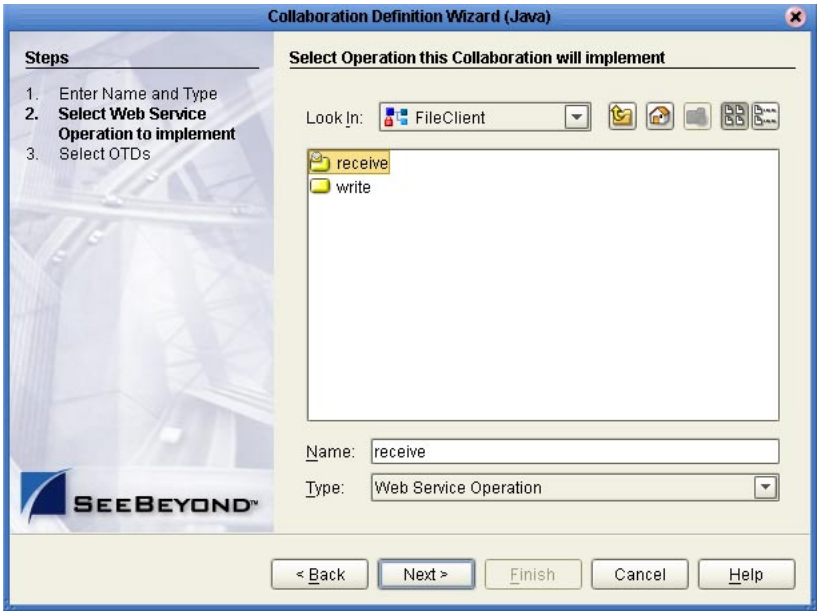
1. In the Project Explorer of the Enterprise Designer, right-click the COBOL copybook conversion Project, click **New** and click **Collaboration Definition (Java)**. The **Collaboration Definition** wizard appears.
2. In the **Collaboration Name** box, enter the name for the Collaboration and click **Next**. The **Select Operation** page appears as shown below.

Figure 13 Selecting Collaboration Operations



3. Double-click **SeeBeyond** and **eWays—**continue to double-click to select the inbound eWay and the (inbound) web service. For example, for the a File eWay, double-click **File**, **FileClient**, and click **receive** as shown below.

Figure 14 Selecting File Receive



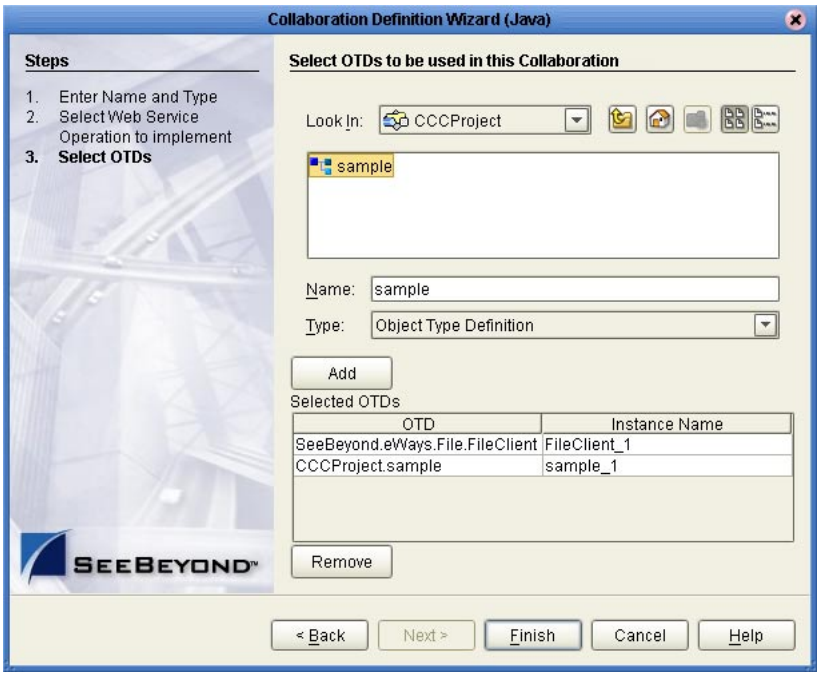
4 Click **Next**.

5 Double-click **SeeBeyond**, **eWays—**continue to double-click to select the outbound eWay and the (outbound) web service. For example, for the File eWay, double-click **File**, and then **FileClient**.

6 In the **Look In** box, browse to the Project with the copybook file to be used for this conversion.

7 Double-click the copybook file. This adds the copybook file as shown below.

Figure 15 Completed Collaboration Definition



8 Click **Finish**. The **Collaboration Editor** window appears.

You can now create the business logic for the Collaboration as described below.

4.6.4. Building Collaboration Definitions

Once you have created the Collaboration Definition as described in the section above, you can create the business logic for the Collaboration. The business logic for a copybook conversion consist of the following components:

1 **Unmarshaling the Input Formats** on page 37

2 **Specifying Destinations** on page 40

3 **Writing The Output to a File** on page 42

Unmarshaling the Input Formats

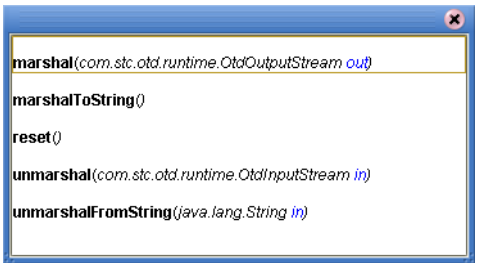
The first step in the business logic is to handle the data when it comes into the Project. The Cobol Copybook OTD can process text data, and as such, text data can easily be unmarshaled with the **unmarshalFromString method()**.

For other data, you must convert the array data into an array input stream, and then into an OTD input stream.

To unmarshal text input format

1 Right-click the copybook OTD and click **Select a method to call**. A list of methods appears.

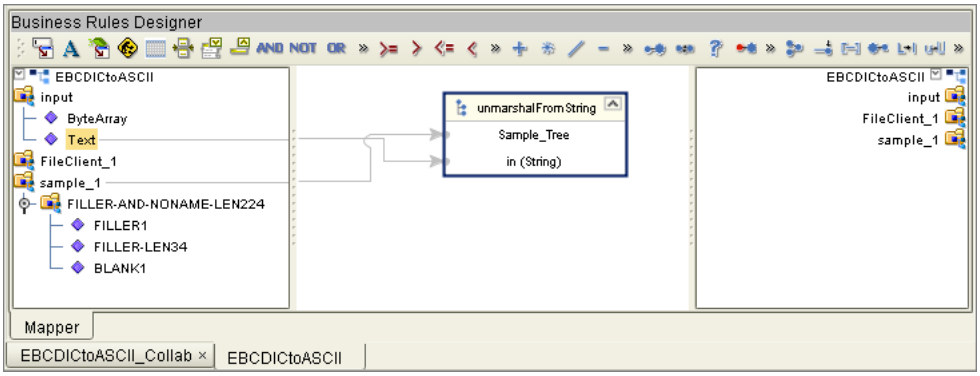
Figure 16 Cobol Copybook Converter Methods



2 Click **unmarshalFromString()**. The **unmarshalFromString** box appears.

3 Expand the input node and drag **Text** into in **(String)** as shown below.

Figure 17 Unmarshaling Text Input



To handle bytes input format

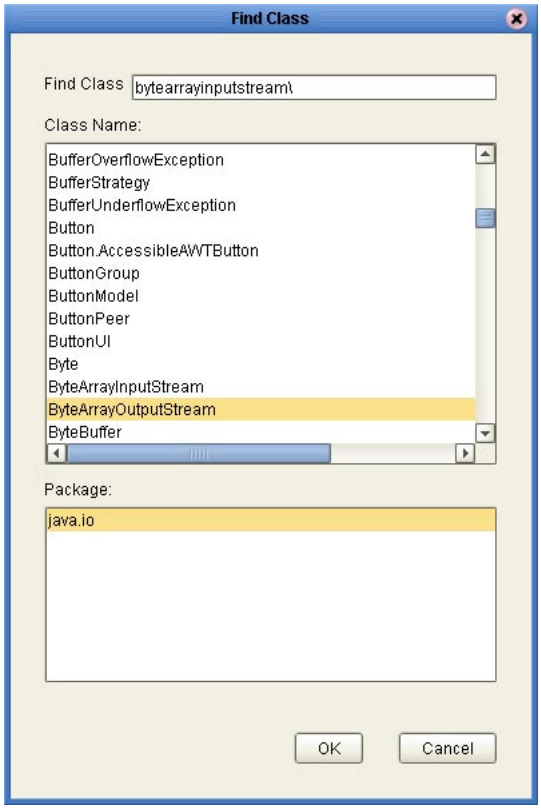
1 Click **Local Variable**. The **Create a variable** dialog box appears.

2 In the **Variable Name** box, enter the variable name.

3 Click **Class** and the ellipsis button. The **Find Class** dialog box appears.

4 In the **Find Class** box, type **bytearray** and press **ENTER**. The **Find Class** dialog box shows the package available for the ByteArrayInputStream as shown below.

Figure 18 Creating a ByteArrayInputStream Variable



5 Click **OK** twice.

6 Click **Local Variable** to create the second variable to convert the array input stream to the OTD input stream. The **Create a variable** dialog box appears.

7 In the **Variable Name** box, enter the name of the variable, for example, OTDstream.

8 In the **Class** box, type:

com.stc.otd.runtime.OtdInputStream

9 Click **OK**. This add the following business rule:

com.stc.otd.runtime.OtdInputStream.otdstream;

10 Click **Source code mode** and scroll to the business rule.

11 Delete the semi colon at the end of the line.

12 Add the following code:

= new com.stc.otd.runtime.provider.SimpleOtdInputStreamImpl(***firstvariable*** **);**

Where *firstvariable* is the variable created in step 6.

Figure 19 Added Variable Code

```
15    {
16        com.stc.otd.runtime.OtdInputStream otdStream = new com.stc.otd.runtime.provider.SimpleOtdInputStreamImpl( ByteStream );
17    }
```

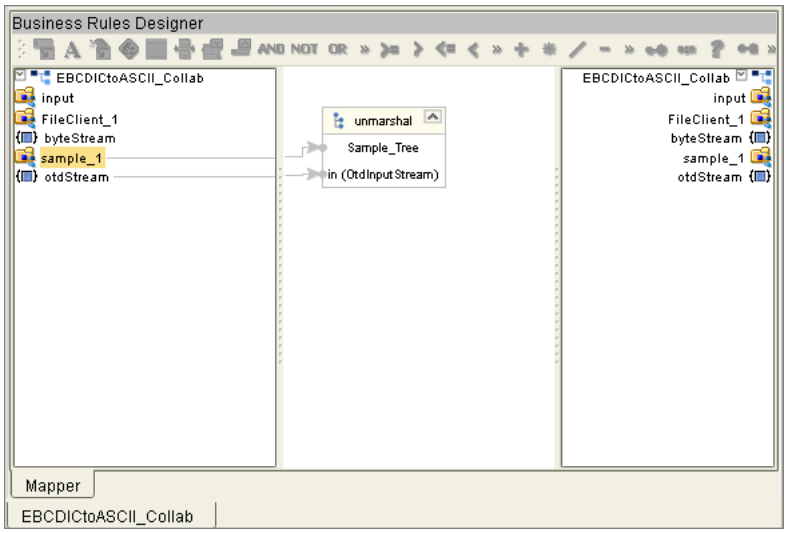
13 Click **Commit Changes**.

14 Right-click the copybook OTD (with the copybook filename) and click **Select a method to call**.

15 Click **unmarshal()**. This adds the **unmarshal** box.

16 Drag the *firstvariable* created in step 6 to **in (OtdInputStream)** as shown below.

Figure 20 Unmarshaling Non-String Data



Specifying Destinations

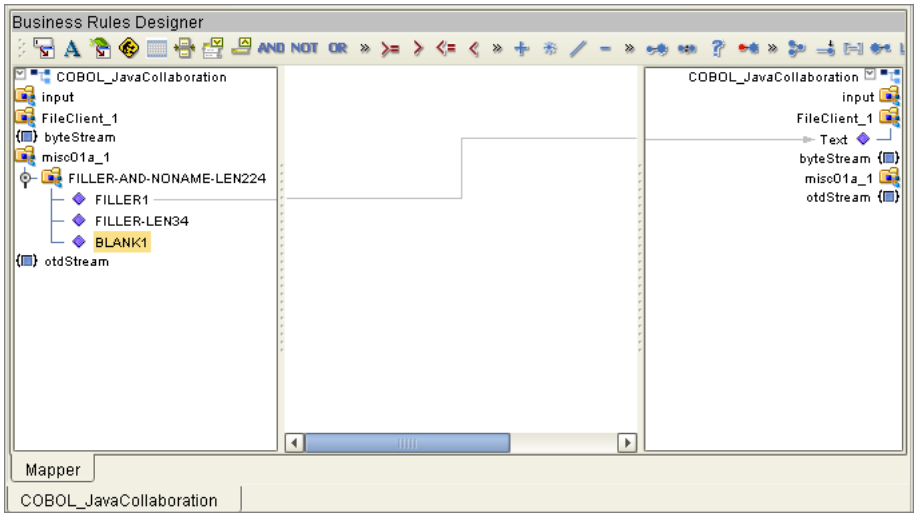
You can specify destinations by mapping specific input data to output data, or you can marshal the data to the destination.

To map input and output data

1 Expand the input OTD node.

2 Drag the input nodes to the output data type under the output service as shown below.

Figure 21 Mapping Input and Output Data

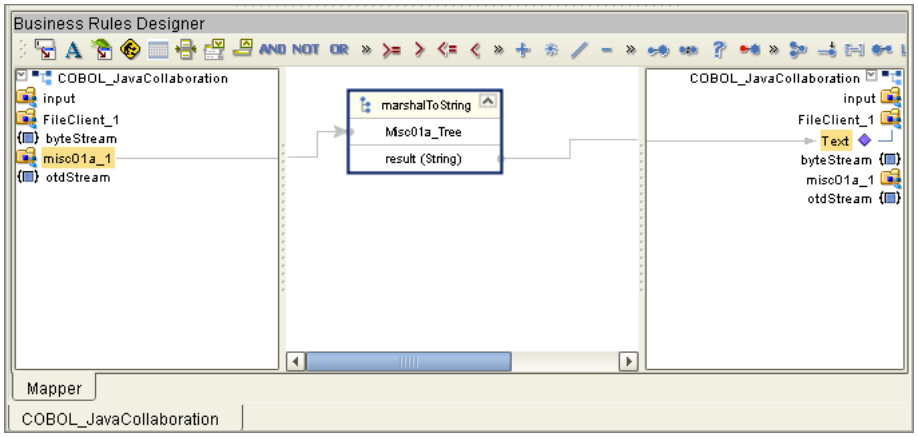


To marshal data as strings to an output destination

1 Right-click the copybook OTD, click **Select a method to call**, and click **marshalToString()**. The **marshalToString** box appears.

2 Drag **Result (String)** to the output OTD as shown below.

Figure 22 Marshaling Data as String to an Output Destination



To marshal data to an output destination

1 Right-click the copybook OTD, click **Select a method to call**, and click **marshal()**. The **marshal** box appears.

2 Drag **Out (OtdOutputStream)** to the appropriate payload node of the output OTD. Verify that the marshal method you selected has a result type compatible with the payload type.

Writing The Output to a File

If you are using a File eWay for the output of the copybook conversion Project, you can use the method below to write the output to a file.

To write the output to a file

1. Right-click **FileClient_1** in the input column, click **Select a method to call**, and click **write()**.
2. Click **Save**.

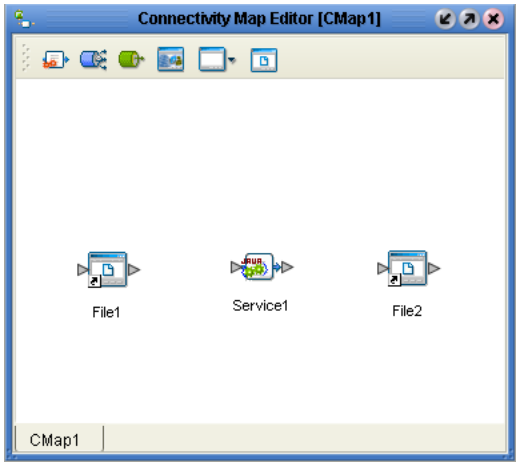
4.6.5. Binding the Collaboration Definition and eWays

Once you have created the Collaboration and its business logic as described in the section above, you can bind the new Collaboration Definition to the Service, and then connect the Collaboration to the eWays.

To bind the Collaboration Definition and eWays

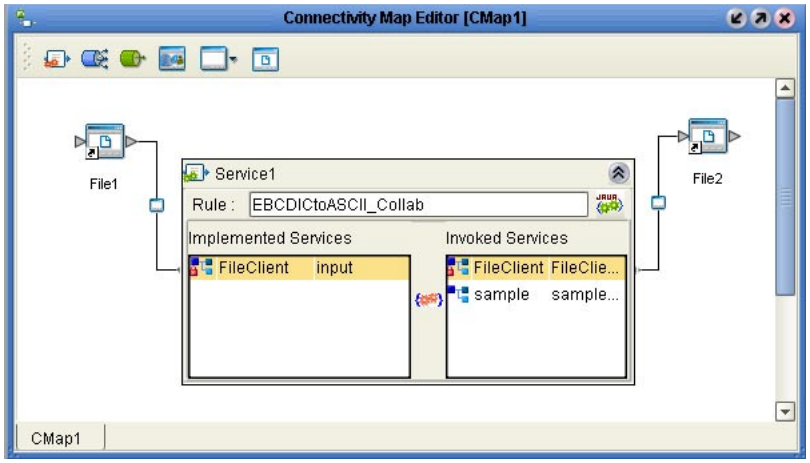
1. From the Project Explorer of the Enterprise Designer, drag the newly-created Collaboration Definition to the Service in the Connectivity Map as shown below.

Figure 23 Binding the Collaboration Definition and Service



2. Double-click the **Service** icon. The **Service1** window appears.
3. Drag the input service to the inbound eWay. For example, for a File eWay, the input service is **FileClient input**.
4. Drag the output service to the outbound eWay as shown below.

Figure 24 Connecting the Collaboration to the eWays



5 Close the **Service1** window and click **Save**.

Once you have completed the Connectivity Map binding, you must do the following to finish the Project:

1 Configure the File eWays as described in the eWay documentation.

2 Create an Environment and Deployment Profile and run the Project as described in the *eGate Integrator User's Guide*.

Index