Informatica® B2B Data Transformation
10.4.0

Libraries Guide

Informatica B2B Data Transformation Libraries Guide
10.4.0
April 2020

Publication Date: 2022-04-12

Table of Contents

Preface

See the *Data Transformation Libraries Guide* to learn how to use the library transformation components to transform data to or from industry-standard messaging formats and how to customize the library components for specialized needs, for example, to change the way that a message type is transformed. The libraries are described in the guide.

The guide assumes that you understand the standards of the libraries you use and that you know how to deploy and run Data Transformation services.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at http://velocity.informatica.com. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

CHAPTER 1

Libraries for Industry Standards

This chapter includes the following topics:

Purpose of Libraries

Data Transformation libraries contain predefined transformation components for use with the following industry messaging standards:

- ACORD AL3
- ACORD-MDM Mappers
- BAI
- Bloomberg Per Security, Back Office and Extended Back Office
- CREST
- DTCC-NSCC
- EDIFACT
- EDI-X12 and related EDI standards
- FIX
- FpML
- HIPAA
- HL7
- IATA PADIS
- IDC-IDSI
- MDM JavaBeans
- NACHA
- NCPDP
- SEPA
- SWIFT

- Telekurs VDF
- Thomson Reuters DataScope Select

Each library contains a large number of components, such as parsers, serializers, and XML schemas, designed for use with industry standard and specific application messages. You can use these components to transform documents from industry standards to XML and from XML to other formats.

Library components offer the following advantages:

- You can use predefined library components, rather than design and implement the components yourself.
- Predesigned components ensure standardized implementation and consistency among your applications.
- Robust construction supports all options and syntactic intricacies unique to each industry standard.

Library Structure

A Library is a set of transformations. All Libraries contain Parsers, Serializers, and XML schemas. Some Libraries contain additional components for message validation, acknowledgments, and diagnostic displays. A Library object is a set of components that convert a specific industry message type.

When you create a Data Processor transformation, you can include a Library object instead of creating your own Scripts to transform a standard industry message type. You can use a Library object without modification, or you can edit it based on your requirements.

Each Library object transforms a particular industry standard. For example, the HL7 Library contains components for each of the message types or data structures available in the HL7 messaging standard for medical information systems.

The Library contains specific types of message types. For example, the HL7 Library contains messages such as:

```
ADT_A01_Admit_a_Patient
ADT_A02_Transfer_a_Patient
```

Obtaining Libraries

The Data Transformation libraries are updated and released at frequent intervals. To get a library, contact Informatica. Specify the name of the library and your Data Transformation version. With certain limitations, a single library release can be used with multiple Data Transformation versions.

Library Documentation

In addition to reading this book, be sure to read the documentation supplied with each library, such as a ReadMe file, Release Notes, or other explanatory material.

Installing Libraries

Install the Libraries on a Microsoft Windows computer where you run Data Transformation on the Developer tool.

To install the DTCC-NSCC, EDIFACT, EDI-X12, HIPAA, HL7, or SWIFT libraries, use the installation procedure for Libraries that use the Library editor.

To install the ACORD, BAI, CREST, EDI-UCS & WINS, EDI-VICS, FIX, FpML, HIX, IATA, IDC, MDM Mappings, NACHA, NCPDP, SEPA, or Telekurs libraries, use the installation procedure for Libraries that do not use the Library editor.

Installing Libraries that Use the Library Editor

Before you can use a Data Transformation library, you must install it. Use the same steps to update a library.

Note: If you are running the previous version of HL7, you can update the library from the version 10.1.1 library .zip file.

1. Copy the library .zip file to your local file system and unzip the file.
2. From the unzipped folder, copy the folder `\<lib name>\Developer\DT` to the directory `<Install_Dir>\<version>\clients\`.

Installing Libraries that Do Not Use the Library Editor

Before you can use a Data Transformation library, you must install it. Use the same steps to update a library.

1. Copy the library .zip file to your local file system.
2. Unzip the file in the directory `<Install_Dir>\<version>\clients\DT\DxT_Libraries`.

   If the `DxT_Libraries` folder does not exist, create it and then copy the file.

Version Compatibility of Libraries

Use the libraries relevant to your current Data Transformation version. This ensures that you have the most recent version of each library component.

You can import projects from older library versions into the Developer tool. Import the projects as Data Transformation services. You can use the Scripts and edit them in the IntelliScript editor.

XML-Based Industry Standards

In addition to the libraries supporting specific industry messaging types, Data Transformation has been certified for use with the following XML-based standards. Certification means that Data Transformation was explicitly tested and correctly processes XML data conforming to the standard.

Note: Some of the table entries are XML versions of standards for which there are Data Transformation libraries. The library schemas are not necessarily the same as the schemas defined in the standards.

ACORD XML:

XML version of the ACORD standard for the insurance industry.

FIXML

XML version of the Financial Information eXchange standard.

FpML

Financial products Markup Language for complex financial products.

HL7 Version 3

XML version of the HL7 standard for the health industry.

IFX

Interactive Financial eXchange standard for sharing financial information.

MISMO

Mortgage Industry Standards Maintenance Organization for mortgage and investment products and services.

NIEM

National Information Exchange Model for exchange of information between US governmental agencies and private industries.

OAGi

Open Applications Group Inc. standard for business process interoperability.

RosettaNet

Standard for B2B commerce and collaborative processes.

SEPA

Single Euro Payments Area electronic payments standard.

SWIFT MX

XML version of the SWIFT standard for the financial industry.

TWIST

Transaction Workflow Innovation Standards Team for electronic commerce.

UNIFI (ISO 20022)

UNIversal Financial Industry messaging standard.

CHAPTER 2

Using Libraries

This chapter includes the following topics:

Libraries and the Data Processor Transformation

A Data Processor transformation uses Scripts and Library objects to transform industry message type inputs into other formats, and vice-versa.

A Library transformation contains a large number of objects and components, such as Parsers, Serializers, and XML schemas, that transform the industry standard input and specific application messages into XML output. A Library transformation might contain objects for message validation, acknowledgments, and diagnostic displays. A Library transformation uses objects to transform the messaging type from industry standard input to XML and from XML to the library standard format.

You can create Library objects for ACORD, BAI, CREST, DTCC-NSCC, EDIFACT, EDI-UCS & WINS, EDI-VICS, EDI-X12, FIX, FpML, HIPAA, HIX, HL7, IATA, IDS, MDM Mapping, NACHA, NCPDP, SEPA, SWIFT, and Telekurs libraries.

You can use a dedicated Library editor to edit the Library specifications for the DTCC-NSCC, EDIFACT, EDI-X12, HIPAA, HL7, and SWIFT libraries. A Library object contains a root element, container elements, and data elements. The types of container and data elements vary according to message type. You can add and delete elements and configure the properties of elements to change validation settings.

When you create transformations for the Bloomberg, COBOL, and Thomson Reuters libraries, they do not contain Library objects.

How to Create the Data Processor Transformation

To create a Library transformation, for most libraries you use the New Transformation wizard to create a Data Processor transformation and then configure it with a Library object.

For the ACORD, BAI, CREST, DTCC-NSCC, EDIFACT, EDI-UCS & WINS, EDI-VICS, EDI-X12, FIX, FpLM, HIPAA, HIX, HL7, IATA, IDC, IDS, MDM Mappings, NACHA, NCPDP, SEPA, SWIFT and Telekurs libraries, create a blank Data Processor transformation. After you create the transformation, you add a Library object.

Creating the Data Processor Transformation

Create a Data Processor transformation in the Developer tool. After that, you can create a Library object in the transformation. For the Bloomberg, COBOL, and Thomson Reuters libraries, you can use the New Transformation wizard to auto-generate a Data Processor transformation.

1. In the Developer tool, click **File** > **New** > **Transformation**.
2. Select the Data Processor transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model repository location to put the transformation.
4. Select to create a blank Data Processor transformation and click **Finish**.

   The Developer tool creates the empty transformation in the repository. The transformation appears in the **Overview** view.

Creating a Library Object

Create a Library object on the Data Processor transformation **Objects** view. Select the message type, component and name. Optionally, you can define a sample message type source file that you can use to test the Library object.

Before you create a Library object in the Data Processor transformation, install the library software package on your computer.

1. In the **Objects** view, click **New.**
2. Select Library and click **Next.**
3. Browse to select the message type.
4. Choose to create a Parser or Serializer.

   Create a Parser if the Library object input is a message type and the output is XML. Create a Serializer if the Library object input is XML and the output is a message type.
5. If the Library is the first component to process data in the Data Processor transformation, enable **Set as startup component**.

   Click **Next.**
6. If you have a sample message type source file that you can use to test the Library with, browse for and select the file from the file system.

   You can change the sample file.
7. Click **Finish**.

Library Object Development

A Data Processor transformation uses a Library object to transform a message type from industry standard input to XML, and vice-versa, and from XML to other formats. You can use the Library editor to customize the message type structures and elements for the DTCC-NSCC, EDIFACT, EDI-X12, HIPAA, HL7, and SWIFT libraries.

To change the structure of the message type, you can add, edit, and delete elements from the Library object in the Library editor. You can also test and run the Library object from the Library editor.

For the ACORD, BAI, Bloomberg, CREST, EDI-UCS & WINS, EDI-VICS, FIX, FpLM, HIX, IATA, IDC, NACHA, NCPDP, SEPA, Telekurs, and Thompson Reuters libraries, you do not use the Library editor to change the

structure of a message. You can use the IntelliScript editor to edit the transformation Parsers, Serializers, and Mappers.

Edit Libraries with the Library Editor

You can use a dedicated Library editor to edit the Library specifications for the EDI-X12, SWIFT, HL7, HIPAA, DTCC-NSCC, and EDIFACT libraries.

The Library editor lets you customize the message type structures and elements with an editor that is customized for each message type. You can add, edit and delete elements from the Library object with the Library Editor.

Message Structures and Properties

The message structures used in a particular library are defined in the corresponding industry standard. This section presents a brief description of the structures and the properties that you can edit in the Library Customization Tool. We have attempted to define the terms in everyday language, rather than the precise but highly technical language used in the standards.

For more information about the precise definition of each term, see the appropriate industry standard. For example, for detailed information about HIPAA message structures and properties, see the HIPAA documentation.

Hierarchical Structure

The Library editor displays the hierarchical structure of a message. It labels each structural element with an icon. The following table describes the icons the editor uses to display HIPAA messages:

Icon	Structural Element
	Transaction set. Description: A message.
	Loop Description: A group of segments, possibly repeated.
	Segment Description: A sequence of data elements or composites.
	Composite Description: A structure that contains data elements.
	Data element Description: A simple data item.

The HIPAA standard defines the meaning of these elements and how they are nested. For example, a segment can contain composites and data elements. A composite contains data elements.

The icons and structure elements differ for other industry standards. In some standards, the elements have names such as message, record, and field instead of transaction, segment, and data element. Some standards define additional logical constructs such as options or alternatives.

Global and Positional Settings

The Library editor can display two types of settings for each element:

- Global settings. Properties that apply to all instances of an element. If the same element is defined in multiple contexts of the message, editing the global settings affects all the instances.
- Positional settings. Properties that apply to a single definition of the element in the message. Editing the positional settings affects only the message context that you have selected in the Library editor.

In the HIPAA library, for example, a global setting might permit an element to contain any of 10 enumerated values. A positional setting might limit the same element to a subset of two values.

Adding an Element with the Library Editor

A Library message contains a root element, container elements, and data elements. Use the Library editor to add an element to a message.

1. In the **Objects** view, select the Library editor object and click **Open**.

   The Library editor appears.
2. Click the **Add element** icon and choose where you want to add the element.
   - **New**. Append the element to the end of the list of elements.
   - **Insert Above**. Insert the element above the element selected in the Library editor.
   - **Insert Below**. Insert the element below the element selected in the Library editor.
   - **Insert Within**. Insert the element within the container element selected in the Library editor.
3. Select whether to copy an existing element or create a new element and click **OK**.

Editing the Element Properties with the Library Editor

Use the Library editor to edit a container element or data element in a Library. When you change the properties of an element, you change the specifications for the message type.

1. In the **Message Definition** window of the Library editor, select the element to edit.

   The **Properties** window displays the element properties.
2. In the **Properties** window, select a property and enter the new property value.

   If you enter an incorrect value type or an out-of-range value, you will be prompted to correct the value.

Testing a Library

Test the Library object in the **Data Viewer** view.

Before you test the Library, select a sample input file to test.

1. To select a sample input file, in the Library editor **General** panel, near the **Sample input** field, browse to select a sample input file.
2. Open the **Data Viewer** view.
3. To select the Library message type that you edited as the component to run, click **Synchronize with Editor**.
4. Click **Run**.

   The Developer tool runs the Library object Parser. The output results appear in the **Output** window.
5. If there are validation errors, the **Output Errors** panel in the **Output** window lists the errors and their location. To find the source of the error, double-click the error.

6. To view the validation error files, click the file names in the **Additional Outputs** panel in the **Output** window.

   When you click the `ErrorsFound.txt` or `Errors.xml` file name, the file opens in an external browser.

Create a Library Transformation with the Wizard

For the Bloomberg, COBOL, MDM JavaBeans, and Thomson Reuters libraries, you use the New Transformation wizard to auto-generate a Data Processor transformation.

Creating the Data Processor Transformation with the Wizard

Use the New Transformation wizard to auto-generate a Data Processor transformation.

1. In the Developer tool, click **File** > **New** > **Transformation**.
2. Select the Data Processor transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model repository location to put the transformation.
4. Select to create a Data Processor transformation with a wizard, and then click **Next**.
5. Select an input format. Browse to select a schema, copybook, example file, or specification file if required for certain input formats.
6. Select an output format. Browse to select a schema, copybook, example file, or specification file if required for the output format.
7. Click **Finish**. The wizard creates the transformation in the repository.

   The transformation does not contain a Library object, but might contain a Parser, Serializer, Mapper, or an object with common components. If you selected a schema, example file, or specification file, the wizard also creates a schema in the repository that is equivalent to the hierarchy in the file. You can edit the Script components with the IntelliScript editor.

Identifying the Startup Component for a Library Object

Library objects typically have a complex structure, with multiple components defined at the top level and nested levels of the IntelliScript script. The structure is designed to support all the options defined in the industry standard.

Each Library object has a startup component that the Data Processor transformation activates when you run the Library object.

1. In the **Overview** view, check the component listed in the **Service startup component** field.
2. To change the startup component, click **Browse** and select a Script component.

Further Changes to Library Transformations

Advanced users might want to change the way that a message type is transformed. A Library transformation contains many components, such as Scripts with Parsers, Serializers, and XML schemas, that define the transformation.

To access and edit the Scripts, XMaps and schemas that you create with the Library editor, you must generate the Library objects. For more information, see “ Generate Library Objects Overview” on page 91.

CHAPTER 3

Descriptions of the Libraries

This chapter includes the following topics:

ACORD AL3 Library

The ACORD AL3 library implements the AL3 messaging standard used in the insurance industry. AL3 is maintained by the Association for Cooperative Operations Research and Development (ACORD), a nonprofit insurance association. AL3 messages handle the transmission of insurance information such as policies and claims. For more information, see http://www.acord.org/standards/al3.aspx.

The ACORD AL3 library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

ACORD AL3 Message Structure

ACORD has a positional format. It uses question marks as placeholders for missing data.

The following sample is an excerpt of an ACORD message that deals with watercraft insurance:

```
1MHG176   IBM716ESTEDL      ??????????????????231225               AFW
50270000010406111100051           70
2004061111000512TRG212   70         3P PBOAT FMG
0001040611        040511 PCH
20040611200405112TCG135
2TAG212   00003278                     P06260822
WTRP  PRODUCER
2ACI200   Stephen F. Newman, Inc.
1600 Delaware Avenue, Suite 999 Seneca NY1499916837168563387
5BIS172   B10001              P        Ester W.         Powell
??????????????????????????????????00003278
??????????????????????
9BIS168   B10001              3672 Almond drive
Hopatka NY14888    7166090166
5ISI183   B200015BISB10001          ??   ?????????? ???????????????
5SNG117   B200015BISB10001    01DECJane W. Fieldstone
??1??????????????????
5BPI285   F100015BISB10001    P06260822               ??????????
FISH 040511050511
```

XML of an ACORD AL3 Message

The ACORD AL3 library processes XML structures such as the following example:

```
<AL3_301_WaterCraft>
  <Message_Header_Group>
    <Header_00>1MHG176</Header_00>
    <Message_Address_Origination_01>IBM716ESTEDL</Message_Address_Origination_01>
    <Message_Address_Destination_02>????????????????</Message_Address_Destination_02>
    <Contract_Number_03>231225</Contract_Number_03>
    <Password_User_04 />
    <System_Type_Code_05>AFW</System_Type_Code_05>
    <Interface_Software_Revision_Level_06>5027</Interface_Software_Revision_Level_06>
    <Message_Sequence_Number_07>000001</Message_Sequence_Number_07>
    <Message_Transmission_Date_Time_08>0406111100051</Message_Transmission_Date_Time_08>
    <Count_Unit_Code_09 />
    <Special_Handling_10 />
    <Message_Standard_Revision_Level_11>70</Message_Standard_Revision_Level_11>
    <Transmission_Status_Flag_12 />
    <Message_Response_Code_13 />
    <Error_Code_14 />
    <Network_Reference_Code_15 />
    <Network_Reserved_For_Future_Use_16 />
    <Message_Transmission_Date_Time_17>200406111100051</
Message_Transmission_Date_Time_17>
  </Message_Header_Group>
```

ACORD-MDM Mappers Library

The ACORD library implements the LA messaging standard used in the insurance industry. The MDM JavaBeans library creates components that transform flat XML representations of JavaBeans components to and from hierarchical XML files. The ACORD-MDM Mappers library contains Mappers that translate MDM JavaBeans XML files to ACORD LNA XML files and then back to output MDM JavaBeans XML files.

Before you run the Mappers, use the Developer tool to transform the MDM JavaBeans XML file to an output XML file. For more information about the MDM JavaBeans library, see "MDM JavaBeans Library" on page 74.

You can use the Developer tool to import and run the mappers as XMap objects.

The following table describes the files in the XMap directory of the ACORD-MDM Mappers library:

File Name	Description
ACORD_LNA_to_MDM_<version>.xml	Mapper that translates from an ACORD LNA file to an output MDM JavaBeans XML file with the Developer tool.
MDM_to_ACORD_LNA_<version>.xml	Mapper that translates from an output MDM JavaBeans XML file to an ACORD LNA file with the Developer tool.

The following table describes the folders in the ACORD-MDM Mappers directory of the ACORD-MDM Mappers library:

File Name	Description
ACORD_LNA_to_MDM	Folder with mapper files that translate from an ACORD LNA file to an output MDM JavaBeans XML file with the Developer tool. You cannot modify the mapper scripts after you import the files.
MDM_to_ACORD_LNA	Folder with mapper that translate from an output MDM JavaBeans XML file to an ACORD LNA file with the Developer tool. You cannot modify the mapper scripts after you import the files.

Output MDM JavaBeans File XML Structure

The ACORD-MDM Mappers library translates to and from output MDM JavaBeans XML files. The MDM JavaBeans library generates the output XML file with the following name: `input.xml`

The following example shows the XML structure of an output MDM JavaBeans XML file:

```
<agreemts>
        <MDM:com.siperian.inservice.bo.Agreemt>
            <agrmntTp>Policy</agrmntTp>
            <identifier>Z9876543210</identifier>
            <productSpecification>
                <MDM:com.siperian.inservice.bo.ProductSpecification>
                    <shortname>123456</shortname>
                </MDM:com.siperian.inservice.bo.ProductSpecification>
            </productSpecification>
        </MDM:com.siperian.inservice.bo.Agreemt>
    </agreemts>
    <controlData>
        <MDM:com.infa.b2b.ControlData>
            <transRefGUID>01231212121234567890</transRefGUID>
            <transType>Values Inquiry</transType>
        </MDM:com.infa.b2b.ControlData>
    </controlData>
```

ACORD LNA file XML Structure

The ACORD-MDM Mappers library translates to and from ACORD LNA files.

The following example shows the XML structure of an ACORD LNA file:

```
<ACORD:TXLifeRequest PrimaryObjectID="Holding_1">
        <ACORD:TransRefGUID>01231212121234567890</ACORD:TransRefGUID>
        <ACORD:TransType tc="212">Values Inquiry</ACORD:TransType>
        <ACORD:TransSubType tc="21207"/>
        <ACORD:BusinessService>
            <ACORD:ServiceName>http://www.acord.org/schema/ws-porttypes/PnP/1/
FundArrangementValuesInquiry</ACORD:ServiceName>
            <ACORD:ServiceVersion>
                <ACORD:TransVersion>1.0</ACORD:TransVersion>
            </ACORD:ServiceVersion>
        </ACORD:BusinessService>
        <ACORD:TransExeDate>2007-03-29</ACORD:TransExeDate>
        <ACORD:TransExeTime>10:49:01</ACORD:TransExeTime>
        <ACORD:OLifE>
            <ACORD:Holding id="Holding_1">
                <ACORD:HoldingTypeCode tc="2">Policy</ACORD:HoldingTypeCode>
                <ACORD:Policy>
                    <ACORD:PolNumber>Z9876543210</ACORD:PolNumber>
                    <ACORD:ProductCode>123456</ACORD:ProductCode>
                    <ACORD:CarrierCode>12345</ACORD:CarrierCode>
                </ACORD:Policy>
            </ACORD:Holding>
        </ACORD:OLifE>
    </ACORD:TXLifeRequest>
```

Creating an ACORD-MDM Mapper Transformation in the Developer Tool

Import the Mappers to the Developer tool to create a Data Processor transformation with XMap objects.

Before you translate from an MDM JavaBeans XML file to an ACORD LNA file, use the MDM JavaBeans library to generate an XML file from an MDM JavaBeans file.

1. In the Developer tool, click **File** > **Import**.

   The **Import** wizard opens to the **Select** screen.

2. Select **Informatica** > **Import Object Metadata File (Advanced)** and click **Next**.

   The **Import File** screen appears.

3. Click **Browse** and select the mapper XML file that you want to import.

   The **Select Objects to Import** screen appears.

4. In the **Source** pane, select the XMap and the schemas for the mapper and click **Add to Target**.

5. Click **Next**.

   The **Summary** screen appears.

6. Click **Finish**.

   The Developer tool imports the mapper and creates a Data Processor transformation with XMap objects.

For more information about XMap and the Developer tool, see the *Data Transformation User Guide*.

ASN.1 Library

The ASN.1 library processes data that complies with the Abstract Syntax Notation One (ASN.1) standard. The ISO and ITU-T organizations maintains the ASN.1 standard, a standard that the telecommunication industry uses. The standard uses a protocol specification syntax that represents data in compact structures, in binary or text formats.

Use the wizard to create a Data Processor transformation that parses ASN.1 protocol specifications. The Developer tool creates the following objects:

- An XML schema used to represent the ASN.1 data.
- A Parser, Serializer, and Mapper that contain an `AsnToXml` document processor. The processor input data conforms to the protocol specification. The output represents the same data in XML.

The transformation contains objects that you configure to process ASN.1 source data. The current ASN.1 library release processes input in the ASN.1 BER encoding.

The library includes a copy of the OSS Nokalva ASN-1Step utility. Use the utility to edit ASN.1 files. You can access the ASN.1 library services through the C/C++ API or the .NET API.

Note: The ASN.1 library transformations do not validate their input. If the input is not valid, the transformations might fail or might generate unexpected output.

ASN.1 Protocol Specification

When you create a Data Processor transformation for the ASN.1 library, you select an ASN.1 protocol specification for the transformation. The specification defines the expected input structure.

An ASN.1 protocol specification is a schema for an ASN.1 data structure. The specification is a text file with an `*.asn` extension.

The protocol specification defines the expected structure for input data. The following lines from a protocol specification define a nested data structure:

```
MobileTerminatedCall ::= [APPLICATION 10] SEQUENCE
{
    basicCallInformation        MtBasicCallInformation    OPTIONAL,
    locationInformation         LocationInformation       OPTIONAL,
    equipmentInformation        EquipmentInformation      OPTIONAL,
    basicServiceUsedList        BasicServiceUsedList      OPTIONAL,
    supplServiceUsedList        SupplServiceUsedList      OPTIONAL,
    camelServiceUsed            CamelServiceUsed          OPTIONAL,
    valueAddedServiceUsedList   ValueAddedServiceUsedList OPTIONAL,
    dualServiceRequested        DualServiceCode           OPTIONAL,
    operatorSpecInformation     OperatorSpecInfoList      OPTIONAL
}

...


ChargeableSubscriber ::= CHOICE
{
    simChargeableSubscriber SimChargeableSubscriber,
    minChargeableSubscriber MinChargeableSubscriber
}

...

SimChargeableSubscriber ::= [APPLICATION 199] SEQUENCE
{
    imsi     Imsi   OPTIONAL,
    msisdn   Msisdn OPTIONAL
}
```

The entire structure is a protocol data unit (PDU). A single protocol specification can define multiple PDUs. A protocol specification defines multiple types of data structures with different top-level elements.

ASN.1 Input Message Structure

The ASN.1 input is in binary format. The wizard creates a transformation for ASN.1 that includes the `AsnToXml` document processor. The document processor shows the input in an XML representation.

An ASN.1 source usually contains many messages. Each message is a data structure that conforms to the protocol specification. The source can contain separator strings between the messages, and there can be a header string before the first message.

The data is in a binary format. The following example contains data in the BER encoding:

```
_nSOHTe>□SSUB□XSYN_TSOHSOH_YSTX13_WBEL1000000_BELETXUSD□PDC1□jSO_i
SOHSOH_USSOHETX_hSTXENQÅ_tSOHENQfu□jDC2□iSO_hSOHSOH_gENQ-0400□<<□7
SUB_8SOHNUL_:SOHSOH_ETBFF595991799503□7SUB_8SOHSOH_:SOHSOH_ETBFF5
95993269503_GSSOHSOH□ySYN□vDC2□NACK_METX595□ACK_~ETX002gM□n!_mSOHSOH
_pSTXVD_oDC2Voice Mail  Retrie□n$_mSOHSTX_pSTXVR_oNAKVoice Mail
Retrievalcþjû□EM>□GSYN_SOHBSE@BNULX…E__CANACK…)EOTidUS□)EOT_!SOHSI□
,NAKPSO20040130160701_hSOHSOH__SOH[□
DC3□FSSI_8SOHNUL_BSSOHe_;STX+ETX□fENQ_DC4SOHSTX□&–□''□$US_ZSTX11_3
SOHNUL_4SOHSOH_JSOHSOH_KSOHSOH_dSOHSOH□Fm□Ej_BSOHD_iSOHSOH□,STXSI_
,ETXSOHSOH_□SOHNUL_,NULSOHSOH□@:□?SUB_GSTX00_>SOHNUL_ASOH[_DSOHx_U
SOHN_\SOHI□?SUB_GSTX01_>SOHNUL_ASOH[_DSOHx_USOHN_\SOHI□VSO□U
_TSOHSOH_f
```

The `AsnToXml` document processor converts the data to the ASN.1 XER encoding, which is an XML representation. When the Data Processor transformation wizard creates a transformation for ASN.1, it includes the `AsnToXml` document processor in the transformation.

The following example contains the XML that the `AsnToXml` document processor generates:

```
<mobileTerminatedCall>
  <basicCallInformation>
    <chargeableSubscriber>
      <simChargeableSubscriber>
        <imsi>45404200</imsi>
        <msisdn>85290469</msisdn>
      </simChargeableSubscriber>
    </chargeableSubscriber>
```

The XML hierarchy corresponds to the input message structure.

Creating an ASN.1 Library Transformation

To create a Data Processor transformation for ASN.1 input, use the Data Processor transformation wizard. The wizard creates the transformation in the Model repository with a Parser, Serializer, Mapper, and an object with common components. Use the IntelliScript editor to edit these components.

1. Create the transformation with the Data Processor transformation wizard. Add an ASN.1 specification that defines the expect input structure.
2. Use the IntelliScript editor to edit and customize the Parser, Serializer, and Mapper that the wizard created.

Step 1. Create an ASN.1 Library Transformation with the Wizard

Create a Data Processor transformation with ASN.1 input. The Data Processor transformation wizard creates the transformation with components that transform ASN.1 input into XML.

1. In the Developer tool, click **File** > **New** > **Transformation**.
2. Select the Data Processor transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model repository location to put the transformation.
4. Select **Create a data processor using a wizard** and click **Next**.
5. Select ASN.1 as the input format and click **Next**.
6. Browse to select an ASN.1 definition file. Click **Next**.

   Typically, the file has an `*.asn` extension.
7. Select an output format and click **Next**.
8. Click **Finish**.

   The wizard creates the transformation in the Model repository with a Parser, Serializer, Mapper, and an object with common components. The Parser, Serializer, and Mapper are configured with an AsnToXml document processor that converts the input to XML. The wizard also creates an XML schema in the repository that is equivalent to the protocol specification.

Step 2. Edit the ASN.1 Library Transformation

The wizard creates an ASN.1 Library transformation in the Model repository with a Parser, Serializer, Mapper, and an object with common components. You must edit the transformation components to add an example source, anchors, and actions.

The Parser, Serializer, and Mapper are configured with an AsnToXml document processor that converts the input to XML. The AsnToXml document processor converts the input data to the ASN.1 XER encoding, which is an XML representation.

1. In the **Objects** view, double-click the Parser, Serializer, or Mapper to open it in the IntelliScript editor.

   The transformation contains only the AsnToXml document processor. It does not contain an example source, anchors, or actions.
2. Configure the properties of the AsnToXml processor as required.

   The properties are described in the following section.
3. Add an example source to the Parser, Serializer, or Mapper.

   The example source must contain one or more messages that conform to the protocol specification. In the current release, the example must be in the BER encoding (a `*.ber` file).
4. Display the example source.

   It appears as an XML representation of the input data.
5. Under the `contains` line, add components such as anchors and actions that process the XML and generate output.

Installing the ASN-1Step Utility

Use the ASN-1Step development environment to edit and validate ASN.1 files. You can install ASN-1Step on a Windows computer. You can use ASN-1Step to edit ASN.1 protocol specifications and source data.

1. Run the ASN.1Step setup file.

   The setup file has a name such as `asnstep_winVVV.exe`, where `VVV` is a version number.
2. At the prompt, select the option to **Specify Location of ossinfo Manually**.
3. Browse to the Data Transformation installation directory where you stored the `ossinfo` license file.
4. Follow the on-screen instructions to complete the installation.

   For more information about using ASN-1Step, see the online help included in the utility.

ASN.1 Streamer

An `AsnStreamer` component converts and splits a large multi-message ASN.1 document into ASN.1 messages. The `AsnStreamer` can pass each message to an appropriate transformation.

ASN.1 Streamer Properties

The **AsnStreamer** component . splits and converts a large multi-message ASN.1 document into ASN.1 messages.

Each ASN.1 message is converted into XML.

The **AsnStreamer** must be defined at the global level of the Script and it must be the startup component of the transformation.

The following table describes the properties of the **AsnStreamer** component:

Property	Description
asn_file	Defines an ASN.1 specification file.
count	Defines the maximum number of converted ASN.1 messages to pass to the transformation.
encoding	Defines the ASN.1 encoding.
header	Defines a header to exclude from the ASN.1 file. The **header** property has the following options: - NewlineSearch. The header is a newline. - OffsetSearch. Define the header according to the number of characters from the beginning of the file. - PatternSearch. Define the header with a regular expression. - TextSearch. Define the header by an explicit string or a string that you retrieve dynamically from the source document.
max_lookup_size	Defines the maximum quantity of new data, in kilobytes, that the **AsnStreamer** searches for each new segment. For optimal performance, set this property to twice the maximum possible segment size. When an application activates a deployed **AsnStreamer** service through an API, it must set the chunk size parameter to a value that is smaller than the **max_lookup_size**. Default is 10000.

Property	Description
no_constraints	Determines whether to process the ASN file with constraints. The **no_constraints** property has the following options: - true. Process the ASN file without constraints. - false. Process the ASN file with constraints. Default is false.
run_component	Defines a transformation that processes the segment. You can choose one of the following options: - The name of a Parser, Serializer, or Mapper that is configured at the global level of the Script. - A **Mapper** or **Serializer** component. Configure the mapper or serializer within the segment. - A **WriteSegment** component that copies the segment to the output.
pdu_type	Defines the PDU type. Use this property to clarify an ambiguity.
separator	Defines text to ignore between records. The **separator** property has the following options: - NewlineSearch. The separator is a newline. - OffsetSearch. Define the separator by the number of characters from the end of the previous record. - PatternSearch. Define the separator by a regular expression. - TextSearch. Define the separator by an explicit string or a string that you retrieve dynamically from the source document.
xml_root	Define an XML root element to contain the converted ASN.1 messages.

AsnToXml Document Processor

The `AsnToXml` document processor converts ASN.1 data to XML.

The processor input format must be the BER encoding format that the ASN.1 standard defines. The output XML is identical to the ASN.1 XER encoding.

Note: This processor is available when you install the Data Transformation ASN.1 library.

The following table describes the properties of the **AsnToXml** document processor:

Property	Description
asn_file	Defines an ASN.1 specification file.
header	Defines a header to exclude from the ASN.1 file. The **header** property has the following options: - NewlineSearch. The header is a newline. - OffsetSearch. Define the header according to the number of characters from the beginning of the file. - PatternSearch. Define the header with a regular expression. - TextSearch. Define the header by an explicit string or a string that you retrieve dynamically from the source document.
no_constraints	Determines whether to process the ASN file with constraints. The **no_constraints** property has the following options: - true. Process the ASN file without constraints. - false. Process the ASN file with constraints. Default is false.
pdu_type	Defines the PDU type. Use this property to clarify an ambiguity.

Property	Description
process_first_message	Determines whether to process the entire CDR file. The **process_first_message** property has the following options: - true. Only process the first record. - false. Process the entire CDR file. Default is false.
separator	Defines text to ignore between records. The **separator** property has the following options: - NewlineSearch. The separator is a newline. - OffsetSearch. Define the separator by the number of characters from the end of the previous record. - PatternSearch. Define the separator by a regular expression. - TextSearch. Define the separator by an explicit string or a string that you retrieve dynamically from the source document.

Example

In an ASN.1 BER file, there is a header before the first message and separators between the subsequent messages. The header and the separators are random sequences of hexadecimal 00 and FF bytes, of any length. For example, a header or separator might have the following form:

```
00 00 FF 00 FF FF FF 00 FF 00
```

The following configuration enables the `AsnToXml` processor to find the header and separators. The `header` and `separator` properties use regular expressions to define the pattern.

```
pre_processor = AsnToXml
  asn_file = SGSN-CDR-def-v2009A.asn
  header = PatternSearch
    pattern = "[\x00|\xFF]+"
  separator = PatternSearch
    pattern = "[\x00|\xFF]+"
```

The processor ignores the headers and separators, and it processes the messages between them.

ASN.1 Transformers

Transformers modify the output of other components. You can use transformers in Scripts within components such as anchors, serialization anchors, and actions. You can use transformers as document processors.

The following table describes the general properties of the ASN.1 transformer:

Property	Description
disabled	Determines whether the transformation ignores the component and all its child components. Choose one of the following options: - Selected. The transformation ignores the component. - Cleared. The transformation applies the component. Use this property for testing, debugging, and making minor modifications in a project without deleting the existing component. Default is cleared.
name	A descriptive label for the component. This label appears in the `events.cme` file and the **Events** view. Use the **name** property to identify which component caused the event.

Property	Description
optional	Determines whether a failure in the component causes the parent component to fail. Choose one of the following options: - Selected. The parent component is unaffected when the component fails. - Cleared. The parent component fails when the component fails. Default is cleared.
remark	A user-defined comment that describes the purpose or action of the component.

Viewing Advanced Properties

By default, advanced properties are hidden in the script panel of the IntelliScript editor.

- To view advanced properties for a specific component, in the IntelliScript editor press the Right Arrow key or click the double right arrow to the right of the component.

BCDSwapToString

The **BCDSwapToString** transformer transforms packed binary code decimal input into a string. The transformer swaps the nibbles for each byte, truncates the input to the first byte containing FF, and converts the result into a decimal string.

Example

You provide binary input that is equivalent to the following input to a **BCDSwapToString** transformer:

```
35 54 FF 54 54
```

The transformer swaps nibbles in each byte. When the transformer reaches byte FF, the transformer stops processing input and the output is truncated.

The transformer provides the following output:

```
5345
```

BCDToString

The **BCDToString** transformer converts packed binary code decimal input into a string. A binary code decimal number is a number coded so each nibble represents a decimal digit. A nibble is a half byte. The output is in little endian format, where the order of bytes is reversed so that the last byte in the input is the first byte in the output.

Example

You provide binary input that is equivalent to the following input to a **BCDToString** transformer:

```
52 54 98
```

The transformer takes the input and reverses the order of the bytes in the output.

The transformer provides the following string output:

```
985452
```

HexSequenceToDecimal

The **HexSequenceToDecimal** transformer converts an even number of characters representing a series of hexadecimal numbers to a decimal number.

Example

You provide the following hexadecimal input to a **HexSequenceToDecimal** transformer:

```
10170A
```

The transformer translates each input byte from hexadecimal to decimal and provides the following output:

```
162310
```

HexToDecimal

The **HexToDecimal** transformer converts an even number of characters representing a series of hexadecimal numbers to a series of decimal numbers.

The following table describes an additional property of the **HexSequenceToDecimal** transformer:

Property	Description
endian	Determines how the transformation stores data in memory. The **endian** property has the following options: - big_endian. The transformation stores the most significant byte in the smallest address. The bytes are in the same order as the input. - little_endian. The transformation stores the least significant byte in the smallest address. The bytes are in the reverse order of the input. This is the default setting.

Example

You provide the following hexadecimal input to a **HexToDecimal** transformer:

```
1017
```

The transformer translates the full input number from hexadecimal to decimal. By default, the transformer produces a little-endian transformation. Little-endian transformations reverse the order of the bytes.

The transformer provides the following output:

```
5904
```

Swap

The **Swap** transformer takes hexadecimal input in bytes and swaps odd and even nibbles in each byte. A nibble is a half byte.

Example

You provide the following hexadecimal input to a **Swap** transformer:

```
000102
```

The transformer swaps the nibbles in each byte, then provides the following hexadecimal output:

```
001020
```

BAI Library

The BAI library implements the following specifications developed by the BAI financial service industry organization:

- Cash Management Balance Reporting Specifications Version 2 (BAI2)
- Lockbox Communications Standards for Banks (BAI Lockbox)

The BAI library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

For more information about the specifications, see http://www.bai.org/operations/reportingcodes.asp.

BAI2 Message Structure

BAI2 messages have a record format. Each record ends with a line break and contains comma-delimited fields.

The first two digits of each record indicate the record type. In the following excerpt, the `01` record is a file header, containing information such as the sender and receiver IDs. The `02` records label account groups. The other records contain the data for specific bank transactions. The `88` records are continuation lines.

```
01,122099999,123456789,040621,0200,1,65,,2/
02,031001234,122099999,1,040620,2359,,2/
03,0123456789,,010,+4350000,,,040,2830000,,/
88,072,1020000,,,074,500000,,/
16,115,450000,S,100000,200000,150000,,,/
16,115,10000000,S,5000000,4000000,1000000/
88,AX13612,B096132,AMALGAMATED CORP. LOCKBOX
88,DEPOSIT-MISC. RECEIVABLES
49,9150000,4/
03,9876543210,,010,-500000,,,100,1000000,,,400,2000000,,,190/
88,500000,,,110,1000000,,,072,500000,,,074,500000,,,040/
88,-1500000,,/
16,115,500000,S,,200000,300000,,,LOCK BOX NO.68751
49,4000000,5/
98,13150000,2,11/
02,053003456,122099999,1,040620,2359,,2/
03,4589761203,,010,10000000,,,040,5000000,,,074,4000000,,/
...
```

XML Version of a BAI2 Message

The BAI library processes XML representations of BAI2 messages such as the following example:

```
<BAI>
  <Delimiters>
    <RecordDelimiter>/</RecordDelimiter>
    <FieldDelimiter>,</FieldDelimiter>
  </Delimiters>
  <FileHeader_01>
    <SenderIdentification>072000096</SenderIdentification>
    <ReceiverIdentification>072000096</ReceiverIdentification>
    <FileCreationDate>060322</FileCreationDate>
    <FileCreationTime>0433</FileCreationTime>
    <FileIdentificationNumber>1</FileIdentificationNumber>
    <PhysicalRecordLength>80</PhysicalRecordLength>
    <VersionNumber>2</VersionNumber>
  </FileHeader_01>
  <LoopGroups>
    <GroupHeader_02>
      <UltimateReceiverIdentification>000000000</UltimateReceiverIdentification>
      <OriginatorIdentification>072000096</OriginatorIdentification>
      <GroupStatus>1</GroupStatus>
```

```
      <AsOfDate>060321</AsOfDate>
      <AsOfTime>0433</AsOfTime>
      <CurrencyCode />
      <AsOfDateModifier>2</AsOfDateModifier>
    </GroupHeader_02>
    <LoopAccounts>
      <AccountIdentifier_03>
        <CustomerAccountNumber>1000000001</CustomerAccountNumber>
        <CurrencyCode />
        <TypeCode>015</TypeCode>
        <Amount>+0000016668216</Amount>
        <FundsType />
        <TypeCode>040</TypeCode>
        <Amount>+0000016668216</Amount>
```

BAI Lockbox Message Structure

BAI Lockbox messages have a positional structure. The following excerpt is an example:

```
1004121092548012100024806112105O5
241210925480121000248000000000040008000801
5001001020174506112141210925480121000248
6001001000037796911100061463456697O 071324    MAYDAY
4001001601034145204       0000016750
4001001602034145409       0000009500
```

XML of a BAI Lockbox Message

The BAI library processes XML representations of BAI Lockbox messages such as the following example:

```
<BAI_Lockbox>
  <ImmediateAddressHeader>
    <PriorityCode>00</PriorityCode>
    <ImmediateDestination>4121092548</ImmediateDestination>
    <OriginCode>0121000248</OriginCode>
    <DepositDate>061121</DepositDate>
    <TransmissionTime>0505</TransmissionTime>
  </ImmediateAddressHeader>
  <ServiceRecord>
    <UltimateDestination>4121092548</UltimateDestination>
    <OriginCode>0121000248</OriginCode>
    <ReferenceCode>0000000000</ReferenceCode>
    <ServiceType>400</ServiceType>
    <RecordSize>80</RecordSize>
    <BlockingFactor>0080</BlockingFactor>
    <FormatCode>1</FormatCode>
  </ServiceRecord>
  <Lockbox>
    <DetailHeader>
      <BatchNumber>001</BatchNumber>
      <ItemNumber>001</ItemNumber>
      <LockboxCode>0201745</LockboxCode>
      <DepositDate>061121</DepositDate>
      <UltimateDestination>4121092548</UltimateDestination>
      <OriginCode>0121000248</OriginCode>
    </DetailHeader>
    <Batch>
      <DetailRecord>
        <BatchNumber>1</BatchNumber>
        <ItemNumber>1</ItemNumber>
        <RemittanceAmount>377969</RemittanceAmount>
        <RemitterIdentification>11100061</RemitterIdentification>
```

Bloomberg Library

The Bloomberg library validates Bloomberg response messages and converts them to XML.

The Bloomberg library performs the following types of validation:

- Field datatype and width
- Business rules defined by Informatica

The Bloomberg field type definitions change on a daily basis. The Data Transformation Bloomberg library can create a service from a sample request or response message. The service contains a parser for small input files and a streamer for large input files.

The following table describes the files that the service produces:

File	Description
output.xml	Contains all of the data in the original Bloomberg request or response.
Errors.xml	Contains a detailed description of errors in the input.
ErrorsFound.txt	Boolean flag that indicates whether the service detected errors in the input.

Note: When you use a streamer to process large files, some comments do not appear in the output.

Bloomberg Message Structure

A Bloomberg message is made up of a header section, a field-definition section, and a data section. The header section and the field-definition section are optional.

The following example of a Bloomberg message shows a header section, a field-definition section, and a data section:

```
START-OF-FILE
RUNDATE=20110911
PROGRAMNAME=getdata
REPLYFILENAME=DLMRT_comma_bulklist.out
FIRMNAME=dl927
PROGRAMFLAG=oneshot
COLUMNHEADER=yes
SECMASTER=yes
CREDITRISK=yes
DELIMITER=,
OUTPUTFORMAT=bulklist

START-OF-FIELDS
CALL_SCHEDULE
END-OF-FIELDS

TIMESTARTED=Sun Sep 11 10:44:19 EDT 2011
START-OF-DATA
US4581401001 US|10|1| |
US4581401001 US Equity|0|1| |
431022KW1 Equity|10|1| |
431022KW1 Corp                                    |10/02/2009|100.0000
431022KW1 Govt                                    |10/02/2009|100.0000
US694032BD48 Muni                                 |11/01/2007|103.203500
END-OF-DATA
TIMEFINISHED=Sun Sep 11 10:44:22 EDT 2011
END-OF-FILE
```

XML Version of a Bloomberg Message

The following example shows the XML version of a Bloomberg message:

```
<?xml version="1.0" encoding="windows-1252" ?>
<BBG:BloombergDataLicense xmlns:BBG="http://www.informatica.com/B2B/Bloomberg"
product="PerSecurity"
    current_date="20110911" time_started="Sun Sep 11 10:44:19 EDT 2011"
time_finished="Sun Sep 11 10:44:22 EDT 2011">
  <Header>
    <Flag name="RUNDATE">
      <Value>20110911</Value>
    </Flag>
    <Flag name="PROGRAMNAME">
      <Value>getdata</Value>
    </Flag>
    <Flag name="REPLYFILENAME">
      <Value>DLMRT_comma_bulklist.out</Value>
    </Flag>
    <Flag name="FIRMNAME">
      <Value>dl927</Value>
    </Flag>
    <Flag name="PROGRAMFLAG">
      <Value>oneshot</Value>
    </Flag>
    <Flag name="COLUMNHEADER">
      <Value>yes</Value>
    </Flag>
    <Flag name="SECMASTER">
      <Value>yes</Value>
    </Flag>
    <Flag name="CREDITRISK">
      <Value>yes</Value>
    </Flag>
    <Flag name="DELIMITER">
      <Value>,</Value>
    </Flag>
    <Flag name="OUTPUTFORMAT">
      <Value>bulklist</Value>
    </Flag>
  </Header>
  <Fields>
    <Field name="CALL_SCHEDULE" />
  </Fields>
  <Record object_id="US4581401001 US">
    <CALL_SCHEDULE />
  </Record>
  <Record object_id="US4581401001 US Equity">
    <CALL_SCHEDULE />
  </Record>
  <Record object_id="431022KW1 Equity">
    <CALL_SCHEDULE />
  </Record>
  <Record object_id="431022KW1 Corp">
    <CALL_SCHEDULE>
      <Row>
        <BC_DT>10/02/2009</BC_DT>
        <BC_CALL_SCHEDULE_PX>100.0000</BC_CALL_SCHEDULE_PX>
      </Row>
    </CALL_SCHEDULE>
  </Record>
  <Record object_id="431022KW1 Govt">
    <CALL_SCHEDULE>
      <Row>
        <BC_DT>10/02/2009</BC_DT>
        <BC_CALL_SCHEDULE_PX>100.0000</BC_CALL_SCHEDULE_PX>
      </Row>
    </CALL_SCHEDULE>
  </Record>
  <Record object_id="US694032BD48 Muni">
    <CALL_SCHEDULE>
```

```
      <Row>
        <BC_DT>11/01/2007</BC_DT>
        <BC_CALL_SCHEDULE_PX>103.203500</BC_CALL_SCHEDULE_PX>
      </Row>
    </CALL_SCHEDULE>
  </Record>
</BBG:BloombergDataLicense>
```

Field Definition Files

The fields in Bloomberg responses are defined in files. You can get the most up-to-date files from the Bloomberg FTP site.

The following table describes the field definition files:

File	Description
lookup.out	Defines valid values for fields and the structure of bulk format fields.
fields.csv	Defines fields for Per Security, Back Office, and Extended Back Office.
crisk_fields.csv	Defines fields for the Credit Risk Module.
backoffice_fields.xls	Defines fields for Back Office.
backoffice_extended_fields.xls	Defines fields for Extended Back Office.

Installing the Bloomberg Library

Before you create a Bloomberg service, you must install the Bloomberg library, and then restart the Developer tool.

Note: If you are running a previous version, you can update the library from this release 10 .ZIP file, following the Installing Libraries steps in the *Data Transformation Libraries Guide*.

1. To install the library, copy the Bloomberg library .ZIP file to your local file system and unzip the file.
2. From the unzipped folder, copy the file `Bloomberg_PS_Import.tgp` to the directory `<Install_Dir>\<version>\clients\DT\autoInclude\system`.
3. Copy the folder `CmInternal_Bloomberg` to the directory `<Install_Dir>\<version>\clients\DT\bin\internalServices`.

Creating a Bloomberg Transformation

Create a Bloomberg transformation when you install the Bloomberg library, or when the Bloomberg field definition files change.

1. In the Developer tool, click **File** > **New** > **Transformation**.
2. Select the Data Processor transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model Repository location to put the transformation.
4. Select **Create a data processor using a wizard** and click **Next**.

5. Select the Bloomberg input format. Select one of the following types:
   - Bloomberg Back Office file
   - Bloomberg Extended Back Office file
   - Bloomberg Per Security file
6. Click **Next**.
7. Browse to select a Bloomberg request file or response file. Click **Next**.
8. Select an output format and click **Next**.
9. Click **Finish**.

   The Developer tool creates the transformation in the repository for the Bloomberg message type with a Parser and Streamer. The transformation appears in the **Overview** view.
10. In the **Settings** view **Output** panel, select **Disable automatic output**.
11. To run the Streamer, in the **Overview** view, select the Streamer as the startup component.

COBOL Processing Library

The COBOL library transforms COBOL data to and from XML. When you use the wizard to create a transformation with COBOL input or output, you select a COBOL copybook to define the expected structure of the input or output data.

When you create a Data Processor transformation with COBOL input or output with the wizard, Developer adds the following objects to the transformation:

- A schema object that defines an XML representation of the COBOL data structure.
- For COBOL input, Developer adds a Parser that transforms input data from the COBOL data definition to XML.
- For COBOL output, Developer adds a Serializer that transforms XML to COBOL.

Note: You can create a Data Processor transformation that uses COBOL input or output, but not both. To process EBCDIC encoded COBOL, ensure that you change the encoding settings for the Data Processor transformation to EBCDIC.

Creating a Transformation for COBOL

Use the Data Processor transformation wizard to create a Data Processor transformation with COBOL input or output.

1. In the Developer tool, click **File** > **New** > **Transformation**.
2. Select the Data Processor transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model Repository location to put the transformation.
4. Select **Create a data processor using a wizard** and click **Next**.
5. Select an input format and click **Next**.
6. If you select COBOL as an input format, browse to select a COBOL copybook. Click **Next**.

   The copybook specification file generally has an `*.txt` extension. Developer adds an XSD schema file representing the copybook to the Model repository.

7. Select an output format and click **Next**.
8. If you select COBOL as an output format, browse to select an COBOL copybook. Click **Next**.

   If you selected COBOL as the input format, you do not have the option to select COBOL as the output format.
9. Click **Finish**.

   The Developer tool creates the transformation in the repository. The **Overview** view appears in the Developer tool.
10. In the **Objects** view, double-click the Parser to open it in the IntelliScript editor.
11. If the COBOL data is encoded in EBCDIC, in the **Settings** view change the input or output encoding to the relevant EBCDIC codepage.

COBOL Data Definitions

The COBOL copybook that you use to create a Data Processor transformation can contain data definitions of any complexity. The COBOL copybook and input must comply with data definition rules described in this section.

Supported Data Definitions

The COBOL import supports data definitions of any complexity. For example, the data definitions can use the packed decimal (`COMP-3`), binary (`COMP-1`, `COMP-2`, or `COMP-4`), and logical decimal point (`99v99`) data types. They can contain features such as `REDEFINES`, `OCCURS`, and `OCCURS DEPENDING ON` clauses.

Data Definition Rules

A COBOL data definition must comply with the following rules:

- No more than 72 characters for each line, and no text beyond column 72
- The first line must be a remark, with a `*` in column 7, or it must start with a level number
- The first level number must be in column 1 or 8.

Unsupported Data Definitions

The Data Processor transformation does not support the following COBOL data definitions:

- The special level numbers 66, 77, and 88
- `USAGE` clauses at a group level
- `INDEXED BY` clauses
- `POINTER` and `PROCEDURE-POINTER`

Test Procedures

When you test the COBOL Parser you transform sample COBOL data to XML and verify the output. After you test the Parser, you can run the COBOL serializer on the output of the Parser.

Testing a COBOL Parser

To test the COBOL Parser you need an input file that contains sample COBOL data. The data structure must conform to the data definition that you imported. The Parser transforms sample COBOL data to XML and you verify the output.

1. In the **Object** view, double-click the COBOL Parser.

   The Parser appears in the script panel of the IntelliScript editor.
2. Right-click the Parser name, and then click **Set as Startup Component**.
3. Expand the IntelliScript tree, and then edit the `example_source` property of the Parser. Change its value from `Text` to `LocalFile`.

   The wizard configures the COBOL Parser in a way that does not require an example source document. When you finish testing, you can remove the example source. The example source has no effect on the transformation at run-time.
4. To assign an example file, expand the `LocalFile` component by clicking the double right arrows **>>**. Double-click the `file_name` property and browse to the input file that contains the sample COBOL data.
5. In the **Data Viewer** view **Input** panel, you can examine the example file. If the document does not display, right-click the Parser name in the IntelliScript, and click **Open Example Source**.
6. Click **Run** > **Run Data Viewer** to test the Parser.
7. In the **Data Viewer** view **Output** panel, examine the Parser output.
8. To confirm that the Parser ran without error, in the **Data Viewer** view **Output** panel, click the **Show Events** button. Examine the execution log in the **Data Processor Events** view.

Testing a COBOL Serializer

After you test a COBOL Parser, you can run the COBOL Serializer on the output of the Parser.

1. In the Data Transformation Explorer, double-click the TGP script file of the Serializer.

   The Serializer appears in the script panel of the IntelliScript editor.
2. Right-click the Serializer name, and then click **Set as Startup Component**.
3. Click **Run** > **Run** to activate the Serializer. At the prompt, browse to the `Results\output.xml` file, which you generated when you ran the Parser.
4. Examine the execution log in the **Events** view. Confirm that the Serializer ran without error.
5. To view the Serializer output, double-click the `Results\output.xml` file in the Data Transformation Explorer.

   The display should be the same as the original input on which you ran the Parser.

Editing a Transformation for COBOL

You can edit a transformation for COBOL that you generate with the Data Processor transformation wizard.

If you do this, document your editing. The documentation might be essential if you later revise the COBOL data definition, re-import it to a new transformation, and need to reproduce your editing.

CREST Library

The CREST library processes the messages produced by an electronic settlement system used for stock and bond transactions in the United Kingdom. The CREST standard is maintained by Euroclear.

The CREST library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

For more information about the standard, see https://www.euroclear.com/site/public/EUI.

CREST Message Structure

A CREST message is a file containing multiple messages delimited by headers and trailers. The fields contain a colon-delimited tag followed by data.

```
:79:/HSMT/ZHDR
:79:/HVRN/001
:79:/HMDE/L
:79:/HSEQ/00000001
:79:/UFTQ/0003457
:79:/IUSE/UKUSER01
:79:/UOPR/OP001
:79:/HSMT/ADVN
:79:/HVRN/003
:79:/HMDE/L
:79:/HSEQ/00000002
:79:/HFPI/PRT01
:79:/HDPI/PRT01
:20:TRANSREF-01
:31P:19980123
```

XML of a CREST Message

The CREST library processes XML structures such as the following example:

```
<CREST>
  <FileTransferHeader>
    <SubMessageType>ZHDR</SubMessageType>
    <VersionNumber>001</VersionNumber>
    <Mode>L</Mode>
    <FileTransferItemNumber>00000001</FileTransferItemNumber>
    <FileTransfer>0003457</FileTransfer>
    <UserId>UKUSER01</UserId>
    <OperatorReference>OP001</OperatorReference>
  </FileTransferHeader>
  <SubMessage>
    <ADVN_3>
      <SubMessageHeader>
        <SubMessageType>ADVN</SubMessageType>
        <VersionNumber>003</VersionNumber>
        <Mode>L</Mode>
        <FileTransferItemResponseNumber>00000002</FileTransferItemResponseNumber>
        <FunctionParticipantID>PRT01</FunctionParticipantID>
        <DataParticipantID>PRT01</DataParticipantID>
      </SubMessageHeader>
      <TransactionDetails>
        <TRANSACTION_REFERENCE>TRANSREF-01</TRANSACTION_REFERENCE>
        <TRADE_DATE>19980123</TRADE_DATE>
```

DTCC-NSCC Library

The Depository Trust and Clearing Corporation (DTCC) provides broker-to-broker securities transaction processing services, such as clearance, settlement, and information services for stocks, bonds, and mutual finds.

The National Securities Clearing Corporation (NSCC), a DTCC subsidiary, provides clearing, settlement, and risk management services for US broker-to-broker trades involving equities, corporate and municipal debt, and unit investment trusts. The NSCC messaging protocol is the proprietary system by which DTCC customers exchange financial information and transactions. For more information about DTCC and NSCC, see http://www.dtcc.com.

The Data Transformation DTCC-NSCC library provides components that transform NSCC messages.

You can use the Library editor to edit the DTCC-NSCC Library object.

DTCC-NSCC Message Structure

DTCC-NSCC messages have a positional layout that includes both fixed and variable-length fields. The following example is a small excerpt of an APP - Initial Application and Premium message for purchase of an annuity:

```
1TRANS20070122  792ANNUITIES APP/SUB/CX0001  10230079204       0000000093DTCCCDT
20070122   14:00
RDT 20070122   14:00
```

XML of a DTCC-NSCC Message

The DTCC-NSCC library processes XML structures such as the following example:

```
<APP xmlns="http://www.informatica.com/B2B/DTCC/NSCC">
  <AutorouteHeader>
    <ApplicationDate>20070122</ApplicationDate>
    <SenderIdentifier />
    <OlderReportNumber>792</OlderReportNumber>
    <ProductDescription>ANNUITIES APP/SUB/CX</ProductDescription>
    <OldBranchNumber>000</OldBranchNumber>
    <MulticycleTransmissionCounter />
    <RetransmissionIndicator>1</RetransmissionIndicator>
    <ProductIdentifier>02300792</ProductIdentifier>
    <ApplicationsMulticycleCounter>04</ApplicationsMulticycleCounter>
    <RecipientIdentirfier />
    <RecordCount>0000093</RecordCount>
    <Filler1 />
    <DateCreated>20070122</DateCreated>
    <Filler1/>
    <TimeCreated>14:00</TimeCreated>
```

DTCC-NSCC Message Properties

For DTCC-NSCC messages, the Library editor displays Transaction, Loop, Record, Field, Redefine, and Option properties.

The following table describes the Transaction Set properties:

Property	Description
Library type	DTCC-NSCC. Global setting.
ID	Message ID. Global setting.
Name	Message name. Global setting.

The following table describes the Loop properties:

Property	Description
ID	Loop identifier. Global setting.
Repetitions	Number of times that the loop is repeated, for example, 1 or 2. For an unlimited number, enter >1. Positional setting.

The following table describes the Record properties:

Property	Description
ID	Record identifier. Global setting.
Record type	DTCC-NSCC record type. Global setting.
Sequence	Sequence number of the record. Global setting.
Usage	Required, optional, conditional, or not used. Positional setting.
Repetitions	Number of times that the segment is repeated, for example, 1 or 2. For an unlimited number, enter >1. Positional setting.

The following table describes the Redefine properties:

Property	Description
ID	Redefine identifier. Global setting.
Condition element	The name of an element that is used to evaluate conditions defined in the nested options. Global setting.

The following table describes the Option properties:

Property	Description
ID	Option identifier. Global setting.
Condition	A string value. If the string has the same value as the condition element of the redefine, the option is included in the redefined structure. Global setting.

The following table describes the Field properties:

Property	Description
ID	Field identifier. Global setting.
Number	Field number. Global setting.
Enumerations	A list of permitted data values. Global setting.
Usage	Required, optional, conditional, or not used. Positional setting.
Length	Length of the data. Positional setting.
Type	Data type of the field, for example, AN. Positional setting.
Reject code	A field identifier for display in error messages if the field contains invalid data. Positional setting.
Description	Data description. Positional setting.

Configure the Streamer to Parse Large Inputs

If memory is insufficient, the library Parsers cannot process very large inputs in a single block. Instead, you can use a Streamer to break the input into segments and parse them. The DTCC-NSCC library provides a set of sample Streamers.

The input that you pass to the Streamer must terminate with the following string, signaling the end of the data:

```
#INFA_EOF#
```

In Streamer mode, the DTCC-NSCC library Parsers do not validate the uniqueness of contract IDs.

To use the sample Streamers, import the sample Script with the Streamers into the Developer tool, then copy the Scripts to a Data Processor transformation with a DTCC-NTCC Library object.

Validation

The DTCC-NSCC library Parsers validate the input messages. The Parsers implement DTCC validation types such as mandatory field checks, field-level validation, order of messages, and rule validation. For more information about these validation types, see the DTCC documentation.

By default, the library enables the validation. In cases where the input has already been validated, you can disable the validation. To do this, pass a service parameter called `strictMode` to the Parser, and set its value to `false`. Alternatively, edit the IntelliScript and initialize the `strictMode` variable to a value of `false`.

To selectively disable certain types of validations, you can edit the message schemas in the Library editor. For example, you can change a mandatory field to optional.

The library contains Serializers in two versions. Serializers with names ending in `_Serializer_Restricted` perform strict validation according to the DTCC standard. Serializers with names ending in `_Serializer` perform a lesser degree of validation.

Except in extreme cases, validation errors do not cause the library Parsers or Serializers to fail.

Validation Reports

In addition to the standard parser or serializer output, the DTCC-NSCC transformations write validation messages output ports. The following table describes the output ports:

Output port name	Output
errors	An XML listing of any validation errors that the transformation found.
errorsFound	A boolean value: `true` if the transformation found validation errors, or `false` if it found no errors.

An application can pass the port locations to the transformation service. For more information about output ports, see the *Data Transformation User Guide*.

Handling DTCC REJECT Messages

To parse REJ messages, set the parameter **strictMode** to **false**, and pass the service parameter to the parser. Alternatively, edit the parser with the IntelliScript editor and initialize the **strictMode** variable to a value of **false**.

EDIFACT Library

The EDIFACT library implements the EDIFACT international electronic data interchange standard maintained by the United Nations.

You can use the Library editor to edit the EDIFACT Library object.

EDIFACT Message Structure

An EDIFACT message contains segments and fields, separated by a hierarchy of delimiters characters as in the following example:

```
UNA:+,? '
UNB+UNOA:2+BANESTO+TELEFONICA DE ESPAÑA S.A.+050321:1204+0030200503211'
UNH+20050321120412+BANSTA:098:96A:UN'
BGM+XZ8+20050321120412+9'
DTM+137:20050321:102'
RFF+ACW:20050321113331'
DTM+171:20050321:102'
LIN+1'
SEQ+YF2+1'
GIS+1'
DTM+140:20050321:102'
MOA+9:0,00'
CNT+2:1'
CNT+X27:1'
AUT+INTERV49         107835'
UNT+14+20050321120412'
UNZ+1+0030200503211'
```

XML of an EDIFACT Message

The EDIFACT library processes XML structures such as the following example:

```
<interchange>
  <delimiters>
    <field_separator>+</field_separator>
    <segment_separator>'</segment_separator>
    <composite_separator>:</composite_separator>
    <escape_character>?</escape_character>
  </Delimiters>
  <UNA>
    <R01>:</R01>
    <R02>+</R02>
    <R03>,</R03>
    <R04>?</R04>
    <R05 />
    <R06>'</R06>
  </UNA>
  <UNB>
    <R01>
      <R01>UNOA</R01>
      <R02>2</R02>
    </R01>
    <R02>
      <R01>BANESTO</R01>
    </R02>
```

EDIFACT Message Properties

For EDIFACT messages, the Library editor displays the Transaction Set, Loop, Segment, Composite, and Data Element properties.

The following table describes the Transaction Set properties:

Property	Description
Library type	EDIFACT. Global setting.
Library version	Library version number. Global setting.
ID	Message ID. Global setting.
Name	Message name. Global setting.

The following table describes the Loop properties:

Property	Description
ID	Loop identifier. Global setting.
Name	Loop name. Global setting.
Optional	True or false. Positional setting.
Repetitions	Number of times that the loop is repeated, for example, 1 or 2. For an unlimited number, enter 9999. Positional setting.

The following table describes the Segment properties:

Property	Description
ID	Segment identifier. Global setting.
Name	Segment name. Global setting.
Optional	True or false. Positional setting.
Repetitions	Number of times that the segment is repeated, for example, 1 or 2. For an unlimited number, enter 9999. Positional setting.

The following table describes the Composite properties:

Property	Description
ID	Composite identifier. Global setting.
Name	Composite name. Global setting.
Optional	True or false. Positional setting.

The following table describes the Data Element properties:

Property	Description
ID	Data element identifier. Global setting.
Name	Data element name. Global setting.
Type	Data element type. Global setting.
Min length	Minimum length of the data. Global setting.
Max length	Maximum length of the data. Global setting.
Enumerations	A list of permitted data values. Global setting.
Optional	True or false. Positional setting.

Validation and Acknowledgments

The EDIFACT library contains two versions of the library directories, providing different levels of input validation. The version providing stricter validation also provides a transaction acknowledgment mechanism.

Library Directories with Validations

An Edifact library provides schemas and Parsers that validates input against the EDIFACT standard.

A validating Parser does not fail if it encounters non-conforming input. Instead, the Parser labels its XML output with an error-code attribute that has a value specified in the EDIFACT standard. For example, if a mandatory field is missing, both the field and the segment containing it are labeled with error codes. The error output has the following appearance:

```
<BGM segErrorCd="18">
  <R01 compErrorCd="18">
```

```
    <R01 errorCd="12">352</R01>
  </R01>
  ...
</BGM>
```

The Parsers validate the following features of the input:

- Input type
- Enumerations
- Mandatory fields
- Minimum and maximum length

The following table describes the EDIFACT error codes:

Code	Description
12	Invalid value
13	Missing
18	Unspecified error
35	Too many data element or segment repetitions
36	Too many segment group repetitions
37	Invalid type of character(s)
39	Data element too long
40	Data element too short

EDIFACT Acknowledgments

If you use the `with_validations` components of the EDIFACT library, you can configure a transformation to generate EDIFACT CONTRL acknowledgments automatically. To do this, the transformation can run a predefined Parser on the XML output of the library component. The output of this Parser is the acknowledgment, including a summary of the validation errors.

To generate EDIFACT CONTRL acknowledgments:

1. Run the transformation defined in the `EDIFACT_XML_to_CONTRL` service of the EDIFACT library.
2. As the input of the transformation, pass the XML output of a library Parser.
3. The following table describes the service parameters to pass to the transformation:

Parameter	Required/ Optional	Instructions
inInterchangeReferenceNumber	Required	Outgoing interchange reference number.
inExpectedInterchangeReferenceNumber	Required	Expected incoming interchange reference number for the partner.

Parameter	Required/ Optional	Instructions
inCorrectRecipientIdentification	Optional	UNB-03 identification of the incoming interchange recipient.
inSupportedSyntaxIdentifiers	Optional	A comma-delimited list of supported syntax identifiers, for example: `UNOA,UNOB,UNOC`
inSupportedSyntaxVersions	Optional	A comma-delimited list of supported syntax versions, for example: `1,2,3`
inTestIndicator	Optional	Set the value to `1` to indicate an outgoing interchange test message. The default is empty, meaning a production message.

The transformation generates the CONTRL acknowledgment, including a summary of the validation errors.

Library Directories without Validations

The EDIFACT library services in the `without_validations` directory provide a lesser degree of validation than those in the `with_validations` directory.

The schemas of the `without_validations` projects define all fields as optional. Nevertheless, the parsers check for mandatory data.

The parsers use the `optional` property to implement the validation. If a field is mandatory according to the EDIFACT standard, the `optional` property of the corresponding anchor is false. If a mandatory field is missing, this causes the parser to fail. In that case, the parser does not generate XML output. The event log reports the failure.

EDI-X12 Library

The Data Transformation EDI-X12 library contains components that convert messages between X12 and XML, and generate validation error reports and acknowledgements.

X12 is a standard for electronic data interchange (EDI) between trading partners over data networks. X12 is developed and maintained by the Accredited Standards Committee (ASC). For more information, see http://www.x12.org.

You can use the Library editor to edit the EDI-X12 Library object.

X12 Message Structure

The X12 standard defines the structure of X12 messages.

A valid X12 message is made up of the following layers:

- Interchange. The outer layer that wraps the entire X12 message.

- Functional group. The middle layer that wraps one or more transaction sets.
- Transaction set. The inner layer that contains the data.

Each transaction set contains a header, one or more data segments, and a trailer. A transaction set might contain data such as a purchase order, an invoice, or a statement of account.

The following example illustrates an X12 Vessel Content Details transaction set. In this example, segments are delimited by tildes (~) and data elements are delimited by asterisks (*).

```
ST*109*0001~B4*15*0*2*19960515*1424*Statu*Equi*Equipment*D*XXXX*
Location Identifier*A*3~N9*01*Reference Identification*
Free-form Description*19960515*1424*01*01^Reference Identification^01^
Reference Identification^01^Reference Identification~Q2*X*XX*19960515*
19960515*19960515*10045*21392*A*Flight/Voy*01*Reference Identification*
B*Vessel Name*10691*B*E~V9*AAD*Event*19960515*1424*City Name*XX*XX*001*
XXXXXX*25946*Tr*Free-Form Message*01*25878*XXXXXX*717*437*272*2457*
12935~R4*1*A*Location Identifier*Port Name*XX*Terminal Name*Pier*XX~DTM*
001*19960515*1424*01*CC*Date Time Period~V9*AAD*Event*19960515*1424*
City Name*XX*XX*001*XXXXXX*4685*Tr*Free-Form Message*01*13647*XXXXXX*813*
605*52*20035*12104~N9*01*Reference Identification*Free-form Description*
19960515*1424*01*01^Reference Identification^01^Reference Identification^
01^Reference Identification~SG*2*001*XX*19960515*1424*01~SE*11*0001~
```

XML of an X12 Message

The following example illustrates part of the XML output of an EDI-X12 service:

```
<interchange>
  <delimiters>
    <field_separator>*</field_separator>
    <segment_separator>~</segment_separator>
    <composite_separator>^</composite_separator>
    <escape_character>?</escape_character>
  </Delimiters>
  <ISA>
    <R01>00</R01>
    <R02 />
    <R03>00</R03>
    <R04 />
    <R05>01</R05>
    <R06>003897733</R06>
    <R07>ZZ</R07>
    <R08>SLRGATEWAY</R08>
    <R09>050202</R09>
    <R10>1338</R10>
    <R11>U</R11>
    <R12>00401</R12>
    <R13>000000708</R13>
```

EDI-X12 Message Properties

For EDI-X12 messages, the Library editor displays the Transaction Set, Loop, Segment, Data Element, and Composite properties.

The following table describes the Transaction Set properties:

Property	Description
Library type	EDI-X12. Global settings.
Library version	The library version number. Global settings.
ID	Message identifier. Global settings.
Name	Message name. Global settings.

The following table describes the Loop properties:

Property	Description
ID	Loop identifier. Global settings.
Name	Loop name. Global settings.
Optional	True or false. Global settings.
Repetitions	Number of times that the loop is repeated, for example, 1 or 2. For an unlimited number, enter 9999. Positional settings.

The following table describes the Segment properties:

Property	Description
ID	Segment identifier. Global settings.
Name	Segment name. Global settings.
Optional	True or false. Positional settings.
Repetitions	Number of times that the segment is repeated, for example, 1 or 2. For an unlimited number, enter 9999. Positional settings.

The following table describes the Composite properties:

Property	Description
ID	Composite identifier. Global settings.
Name	Composite name. Global settings.
Optional	True or false. Positional settings.

The following table describes the Data Element properties:

Description
Data element identifier. Global settings.
Data element name. Global settings.
Data element type. Global settings.
Minimum length of the data. Global settings.
Maximum length of the data. Global settings.
A list of permitted data values. Global settings.
True or false. Positional settings.

Validation of X12 Messages

When a Data Processor transformation converts an X12 message to XML, it validates the message for conformity with the X12 standard and generates an error report.

Each Data Transformation X12 service provides a parser and two serializers. The parser performs strict validation. The following table describes the serializers by validation type:

Serializer Validation Type	Description
Loose validation	Service name ends with `_Serializer`. If the service finds validation errors, it generates XML output and an error report.
Strict validation	Service name ends with `_Serializer_Restricted`. If the service finds validation errors, it generates only an error report.

Use the Library editor to disable or modify certain types of validations.

Both serializer types generate SE, GE, and IEA trailer segments. You can omit the SE, GE, and IEA data from the XML input of the serializers.

X12 Validation Error Reports

When an EDI-X12 service finds validation errors in the input message, it generates an error report.

When you run an EDI-X12 service, you specify the **AdditionalOutputPort** for each of the error outputs. The following table describes the error outputs:

Name	Description
Errors	An XML listing of validation errors that the service found, including the standard X12 error codes.
ErrorsFound	Indicates whether the service found validation errors. The **errorsFound** output port can have the following values: - true. The X12 message contained validation errors. - false. The X12 message did not contain validation errors.
ValidationReport	An HTML representation of the `Errors.xml` file.

X12 Message Acknowledgments

X12 message acknowledgements indicate whether a message was received and whether it is valid.

The EDI-X12 library contains services that generate the following types of X12 acknowledgments:

- 997 acknowledgement. Confirms the receipt and validity of a transaction.
- 999 acknowledgement. Confirms the receipt and validity of a transaction.
- TA1 acknowledgement. Confirms the receipt and validity of the interchange envelope, such as the sender and receiver data.

Generating 997 or 999 Acknowledgments

You can generate 997 or 999 acknowledgements from the error report output of an EDI-X12 parser service.

Note: 997 or 999 acknowledgements are relevant only to parser services.

▶ Run one of the following services from the library:

- ACK_997_Generator
- ACK_999_Generator

The following table describes the values to pass to the service:

Parameter	Value
Input document	`Errors` output.
inOutgoingInterchangeControlNumber	Identifies the message. Default is 1.

Generating TA1 Acknowledgments

You can generate a TA1 acknowledgement from an input message. You can also validate the sender ID and qualifier against a list of known senders.

1. To enable validation of sender ID and qualifier, edit the lookup file.

   You can find the `inInterchangeIdsAndQualifiersForSenders.xml` lookup file in the project directory. Enter each sender in the following format:

   ```
   <Entry key="sender_id" value="sender_qualifier"/>
   ```

2. Run the TA1_Generator service from the EDI-X12 library.

   The following table describes the values to pass to the service:

Parameter	Description
Input document	The X12 message. Required.
inOutgoingInterchangeControlNumber	Outgoing interchange control number. Default is 1. Required.
inInterchangeReceiverId	Receiver data. Optional.
inInterchangeReceiverIdQualifier	Receiver data. Optional.
inExpectedInterchangeControlNumbers	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional. For example: `;1;2;3;`
inSupportedFunctionalGroups	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inSupportedFunctionalGroupVersions	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.

Parameter	Description
inValidInterchangeVersionIDs	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inSupportedInterchangeVersionIDs	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inValidInterchangeControlStandardsIdentifiers	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inSupportedInterchangeControlStandardsIdentifiers	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inValidSegmentTerminators	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inValidDataElementSeparators	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inValidComponentElementSeparators	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inValidResponsibleAgencyCodes	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inValidAuthorizationInformationQualifiers	Enter the values in a list that begins with a semicolon and has a semicolon after each value. Optional.
inAddNewlinesToOutgoingMessage	By default, the TA1 message has the same linebreak format as the input messsages. To override the defaults, specify one of the following values: - NONE - CR - LF - CRLF Optional.
luVersionReleaseIndustryToImplementation	By default the TA1 message generates the 999 acknowledgment. To generate the 997 acknowledgment, specify 997. Optional.

EDI Libraries Based on X12

In addition to the EDI-X12 library, Data Transformation provides several libraries implementing X12-based standards used in specific industries. The following list describes the libraries:

EDI-UCS_and_WINS library

Library implementing the Uniform Communication Standard (UCS) and Warehouse Information Network Standard (WINS), used in the grocery and warehouse industries, respectively.

EDI-VICS library

Library implementing the Voluntary Interindustry Commerce Solutions standard, used in the retail industry.

These library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

FIX Library

The FIX library supports the Financial Information eXchange protocol for real-time electronic exchange of securities transactions. The protocol is maintained by FIX Protocol, Ltd.

The FIX library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

For more information about the protocol, see http://www.fixprotocol.org.

FIX Message Structure

A FIX message is a string containing fields delimited by special characters. Each field has a numerical identifier followed by data.

In the following excerpt, the delimiter is ANSI character 01, which is an invisible character. For clarity, we have depicted the delimiter by using the caret symbol (^).

```
8=FIX.
4.1^9=154^35=6^49=BRKR^56=INVMGR^34=239^52=19980604-08:00:36^23=115687^28=N^55=PIRI.MI^
54=1^27=300000^44=5950.001^25=H^10=168
```

XML of a FIX Message

The FIX library processes XML structures such as the following example:

```
<IndicationOfInterest>
  <Header>
    <BeginString>FIX.4.1</BeginString>
    <BodyLength>154</BodyLength>
    <MsgType>6</MsgType>
    <SenderCompID>BRKR</SenderCompID>
    <TargetCompID>INVMGR</TargetCompID>
    <MsgSeqNum>239</MsgSeqNum>
    <SendingTime>19980604-8:00:36</SendingTime>
  </Header>
  <Body>
    <IOIid>115687</IOIid>
    <IOITransType>n</IOITransType>
    <Symbol>PIRI.MI</Symbol>
    <Side>1</Side>
    <IOIShares>300000</IOIShares>
    <Price>5950.001</Price>
    <IOIQltyInd>H</IOIQltyInd>
  </Body>
  <Trailer>
    <CheckSum>168</CheckSum>
  </Trailer>
</IndicationOfInterest>
```

FpML Library

The FpML library implements the industry-standard FpML protocol for complex financial products.

The FpML library implements the Financial Products Markup Language standard for electronic deals and processes regarding over the counter (OTC) derivatives. The FpML library establishes the industry protocol for sharing information about financial derivatives and structured products and dealing in these derivatives and products.

The FpML library supports the following views:

- Confirmation
- Reporting
- Recordkeeping
- Transparency

The library validates messages according to the FpML rules in the specifications.

FpML Validation Error Reports

When an FpML service finds validation errors in the input message, it generates an error report.

When you run an FpML service, you specify the **AdditionalOutputPort** for each of the error outputs. The following table describes the error outputs:

Name	Description
Errors	An XML listing of validation errors that the service found, including the standard FpML error codes.
ErrorsFound	Indicates whether the service found validation errors. The **errorsFound** output port can have the following values: - true. The FpML message contained validation errors. - false. The FpML message did not contain validation errors.

HIPAA Library

HIPAA is an industry standard for administrative and financial health-care transactions.

HIPAA is based on the X12 standard. For more information about X12, see "EDI-X12 Library" on page 49.

In addition to the regular parser, serializer, and schema components provided in every Data Transformation library, there is a special HIPAA library component that validates input messages and generates acknowledgments. The special component implements the complete set of validation and acknowledgment types defined in the HIPAA standard up through version 5010A1.

You can use the Library editor to edit the HIPAA Library object.

HIPAA Message Structure

The following excerpt illustrates a HIPAA message. The example is a Benefit Enrollment and Maintenance transaction, used by an insurance sponsor to provide benefit enrollment information to payers.

```
ISA*00**00**ZZ*EDI0021*ZZ*54828*032403*1253*U*00405*000000905*1*T*:
GS*BE*EDI0021*54828*20030324*1253*5*X*004050X125~
ST*834*26608~
BGN*00*26608*20021226*1200*ES*2~
N1*P5*UES CORP*FI*111222333~
N1*IN*UNITED HEALTHCARE*FI*777888999~
```

```
INS*Y*18*021*28*A***FT~
REF*0F*851027461~
REF*1L*199301~
DTP*356*D8*20021219~
NM1*IL*1*CANOE*NANCY*A*MRS.*I*99*99999999~
PER*IP**HP*4104038790*WP*4104035500*EX*5684~
N3*9999 ROSE VILLAGE*FIRST FLOOR~
N4*FLOWERS*ND*999995*CY*WHEELING~
DMG*D8*19740801*F*M~
HD*021**PPO*PLAN COVERAGE DESCRIPTION*ESP~
DTP*348*D8*20021219~
SE*16*26608~
ST*834*26608~
BGN*00*26608*20021226*1200*ES*2~
N1*P5*UES CORP*FI*111222333~
N1*IN*UNITED HEALTHCARE*FI*777888999~
INS*N*01*021*28*A***FT~
REF*0F*948047150~
REF*1L*199301~
```

HIPAA messages consist of multiple segments. Each segment begins with an identifying code. The following table describes a few examples of the identifying codes contained in the sample message:

Identifying Code	Segment Description
ISA	Interchange Control header
GS	Functional Group Header segment
ST	Header segment
BGN	Beginning segment
N1	Name
SE	Trailer segment

XML Version of a HIPAA Message

The HIPAA library produces XML structures such as the following example:

```
<interchange xmlns="http://www.informatica.com/B2B/HIPAA/4010" xmlns:H834="http://
www.informatica.com/B2B/HIPAA/4010/834" >
  <delimiters>
    <field_separator>*</field_separator>
    <composite_separator>:</composite_separator>
    <segment_separator>!</segment_separator>
  </Delimiters>
  <ISA>
    <R01>00</R01>
    <R02 />
    <R03>00</R03>
    <R04 />
    <R05>ZZ</R05>
    <R06>101018</R06>
    <R07>ZZ</R07>
    <R08>107039</R08>
    <R09>04101</R09>
    <R10>1334</R10>
```

HIPAA Message Properties

For HIPAA messages, the Library editor displays the Transaction Set, Loop, Segment, Data Element, and Composite properties.

The following table describes the Transaction Set properties:

Property	Description
Library type	HIPAA. Global settings.
Library version	Library version number. Global settings.
ID	Message identifier. Global settings.
Standard name	Message name. Global settings.
Internal ID	An internal message identifier. Global settings.
Functional group ID	The group to which the message belongs. Global settings.

The following table describes the Loop properties:

Property	Description
ID	Loop identifier. Global settings.
Repetitions	Number of times that the loop is repeated, for example, 1 or 2. For an unlimited number, enter >1. Positional settings.

The following table describes the Segment properties:

Property	Description
ID	Segment identifier. Global settings.
Standard name	Segment name. Global settings.
Child dependencies	Defines relational conditions between data elements within the segment. Global settings.
Segment qualifier	Index of the data element that qualifies the segment. Global settings.

Property	Description
Usage	Required, optional, conditional, or not used. Positional settings.
Repetitions	Number of times that the segment is repeated, for example, 1 or 2. For an unlimited number, enter >1. Positional settings.
Industry name	Industry-standard descriptive name of the segment. Positional settings.

The following table describes the Composite properties:

Property	Description
ID	Composite identifier.
Standard name	Composite name.
Usage	Required, optional, conditional, or not used. Positional settings.

The following table describes the Data Element properties:

Property	Description
ID	Data element identifier. Global settings.
Standard name	Data element name. Global settings.
Type	Data element type.
Min length	Minimum length of the data. Global settings.
Max length	Maximum length of the data. Global settings.
Enumerations	A list of permitted data values. Global settings.
Usage	Required, optional, conditional, or not used. Positional settings.
Enum subset	Subset of the enumerated values that is valid in this instance of the data element. Positional settings.

Validation and Acknowledgment Behavior

The HIPAA library is supplied in two versions, called HIPAA and HIPAA Validation:

- The HIPAA version contains parsers and other components that do not validate incoming messages and that do not generate acknowledgments.
- The HIPAA Validation version contains the same parsers and other components as the HIPAA version. In addition, the HIPAA Validation version contains a special project that implements the full set of validation and acknowledgment types defined in the HIPAA standard.

Behavior without Validation

The HIPAA parsers are designed to be as forgiving as possible. If the input HIPAA message is invalid, the parsers do not fail, but the XML output might contain missing or incorrect data.

Behavior with Validation

By using the HIPAA Validation version, you can pass incoming messages to a validation and acknowledgment project before you pass them to the regular processors. In this way, you can incorporate logic ensuring that the input data conforms fully to the HIPAA specifications.

Configuring the Edifecs Engine Version 8.x

To use the validation and acknowledgment features, you must first setup and run the Edifecs XEngine.

Configuring the Edifecs Server Version 8.x

Configure the Edifecs version 8.x server to work with Data Transformation.

1. Open the Edifecs Application Manager.
2. To open the Repository, in the **System** view expand the **Configuration Manager** node and double-click the **Repository**.

   The **Repository** view appears.
3. In the **Repository** view, click **Upload** and browse to the following route file:
   `XEServer(8.3.0)_webservice_profile.zip`
4. In the **Repository** view, select the `.zip` file and click **Deploy**.

   The **Installation Wizard** dialog box appears.
5. Select **New Profile** and name the profile as **Informatica_webservice**. Click **Next** twice.
6. Click **Deploy**.

   The **Informatica_webservice** profile appears in the **System** view.
7. To open the Admin Console, in the **System** view expand the **XEServer** node and double-click the Admin Console.

   Check the HTTP port for the **Informatica_webservice** profile.
8. Select the **Informatica_webservice** profile and click **Start**.
9. Create an **Edifecs_Results** folder in the server machine. Ensure that the folder is accessible from the machine running Data Transformation.
   - On a Windows machine, create a folder at the following path: `C:\public\Edifecs_Results`
   - On a Windows or Linux machine, create a folder at the following path: `/Edifecs_Results`

Configuring Data Transformation for Validation with Edifecs Engine Version 8.x

To configure Data Transformation to use HIPAA Validation:

1. In the Data Transformation installation directory, edit the file `CMConfig.xml`.
2. Find the `CMLibraries` element and edit it as in the following example:

```
<CMLibraries>
  <HipaaValidationWS>
    <RemoteWS>
      <LocalPath>\\HIPAAValidationServer\HIPAAValidation\_JMS_Results</LocalPath>
      <RemotePath>/opt/Informatica/HIPAAValidation/_JMS_Results</RemotePath>
    </RemoteWS>
    <Url>http://10.40.40.54:8077</Url>
    <Version>8</Version>
  </HipaaValidationWS>
</CMLibraries>
```

The following table describes the parameters:

Parameter	Description
LocalPath	The path to the `Edifecs_Results` file that Data Transformation uses to access the directory. The path is relative to the Data Transformation machine. The `Edifecs_Results` file is a temporary directory that both Data Transformation and the Edifecs Engine use to transfer data.
RemotePath	The path to the `Edifecs_Results` file that the Edifecs Engine uses to access the directory. The path is relative to the Edifecs Engine machine.
Url	The IP address or host name of the Edifecs server machine, and the HTTP port number that you configured in `$XESRoot/webservice/bin/apache-tomcat-5.5.20/conf/server.xml`.
Version	Specify the major Edifecs version you have installed, such as version 8.

Using Validation and Acknowledgments

The HIPAA Validation version of the HIPAA library contains a special component called `HIPAA_Validation`. This component implements the validation and acknowledgment mechanisms.

The project can process all types of HIPAA messages. It can perform all the validation types defined in the HIPAA standard, for example, checking for mandatory fields and balancing. The project can generate TA1, 997, 999, 824, and 277 acknowledgments. For more information about the validation and acknowledgment types, see the HIPAA documentation.

You can use the `HIPAA_Validation` component in the following way:

1. In the Developer tool, user Parsers from the HIPAA Validation library to create one or more HIPAA Library objects.
2. Create and deploy an additional Library object, based on the `HIPAA_Validation` component of the HIPAA Validation library.
3. When an application receives a HIPAA message, it can pass it to the `HIPAA_Validation` component for validation and generation of acknowledgments.
4. As required, the application passes the acknowledgments back to the sender.
5. If the input passed the validation tests, that application passes the input to the regular parser service, generating an XML representation of the HIPAA message.

Optionally, you can configure transformations that combine these steps. For example, a single transformation might pass the same input to `HIPAA_Validation` and to a regular Parser, in sequence.

Creating a Validation Component

You can create a `HIPAA_Validation` component as part of a Library object.

To create a HIPAA_Validation component, create a Library object and select a HIPAA Validation message type.

1. In the Data Processor transformation **Objects** view, click **New.**
2. Select **Library** and click **Next.**
3. Browse to select the message type `HIPAA_Validation`.
4. Click **Finish**.

Selecting Validation and Acknowledgment Options

1. In the Developer tool, in a Data Processor transformation, create a `HIPAA_Validation` Library object. To create the Library object, perform the following steps:
   a. In the **Objects** view for the Data Processor transformation, click **New.**
   b. Select **Library** and click **Next.**
   c. Select the **HIPAA** > **Validation** message type.
   d. Choose to create a Parser or Serializer.
   e. If you have a sample message type source file that you can use to test the Library with, browse for and select the file from the file system.
   f. Click **Finish**.
2. To select validation and acknowledgment options, open the `HIPAA_Validation_CMLIBHIPAA.tgp` element. The following table describes the validation and acknowledgment options:

Option	Explanation
Separate valid transactions from invalid transactions	If selected, the service copies valid and invalid input messages to different output ports. If not selected, the services copies all messages to the valid port.
Validate transactions	Select this option to enable the validation. If not selected, the service can still generate acknowledgments.
Generate acknowledgments	Select the acknowledgment types to generate. You can select types TA1, 997, 999, 824, and 277.
Validation error reporting format	Select XML output, HTML output, or both. You can use XML for data processing applications such as further processing in Data Transformation. You can use HTML for display.
Validation types	You can select HIPAA validation types 1 to 7.

By default, all the options are selected. Select or clear the relevant options.

Validation Output

The `HIPAA_Validation` service writes validation messages to the following output ports:

Output port name	Output
Acknowledgments	The generated acknowledgments.
ErrorsCount	The number of errors in the input message.
HtmlReport	An HTML validation report.
XmlReport	An XML validation report.
InvalidTransactions	A copy of all invalid transactions in the input message.
ValidTransactions	A copy of all valid transactions in the input message.

An application can pass the port locations to the `HIPAA_Validation` service. For more information about output ports, see the *Data Transformation User Guide*.

HIX Library

The Data Transformation HIX Library addresses the ASCX12 healthcare industry standard. The HIX library uses the Health Insurance Exchange protocol to exchange information for government-regulated and standardized health care plans in the United States.

The HIX protocol is based on the HIPAA standard. In addition, the HIX Library contains HIPAA messages that are relevant to the ASCX12 standard. The messages have Validation Level 1 and 2 only.
Note: Do not use HIPAA or EDI-X12 validation components with the HIX Library. The HIX Library contains separate validation components. These components process supported ASCX12 messages.

The HIX Library message components contain all the relevant variables for the ASCX12 standard. Individual states might require that only a subset of these variables be used. You can configure message variables and rules to address specific state requirements.

The HIX Library is optimized for transaction-based messages and achieves high performance with relatively short messages. If you use the library to run large batches of messages, the performance of the library is not optimal.

You can improve the performance of projects that process large batches of messages by disabling validations for those projects. For example, processing a short message of approximately 1 MB in size with no validations takes a few seconds, while processing the same message with all the available validations might take up to a few minutes.

Size and processing time are not directly proportional. With large messages, the ratio between time to process without validations and time to process with validations might be larger, and the processing of large batches of messages is not optimal even when all the validations are disabled.

HIX Message Structure

A HIX message is a string that contains fields delimited by special characters. Each field has an identifier followed by data.

The following excerpt is an example of the Related Payments message type (820).

```
ISA*00*..........*01*SECRET....*ZZ*SUBMITTERS.ID..*ZZ*RECEIVERS.ID...*030101*1253*^*00501
*000000905*1*T*:~
GS*RA*SENDER CODE*RECEIVERCODE*19991231*0802*1*X*005010X306~
ST*820*0001*005010X306~
BPR*C*310*C*ACH*CCP*******01*199999999*DA*98765*20121103~
TRN*3*78905~
N1*PE*BATA INSURANCE CO.*FI*012222222~
ENT*1~
NM1*IL*1*DOE*JOHN****C1*777222~
REF*IG*555666~
RMR*ZZ*1**30~
ENT*2~
NM1*IL*1*FIRSTONE*EMILY****C1*777333~
REF*IG*555777~
RMR*ZZ*1**45~
ENT*3~
NM1*EY*1*MIDDLEONE*JULIE****C1*777444~
REF*IG*544477~
RMR*ZZ*1**50~
DTM*582****RD8*20120901-20121031~
ENT*4~
NM1*EY*1*NEWONE*KELLY****EI*777111~
RMR*ZZ*2**450~
SE*27*0001~
GE*1*1~
IEA*1*000000905~
```

XML of a HIX Message

The HIX library processes XML structures such as the following example:

```
  <?xml version="1.0" encoding="windows-1252" ?>
- <X5010:interchange xmlns:X5010="http://www.informatica.com/B2B/HIX/5010"
xmlns:X5010_820="http://www.informatica.com/B2B/HIX/5010/820">
    - <delimiters>
      <field_separator>*</field_separator>
      <segment_separator segment_separator_suffix="">~</segment_separator>
      <composite_separator>:</composite_separator>
      <repeat_separator />
      </delimiters>
- <X5010:ISA>
   <R01>00</R01>
   <R02>..........</R02>
   <R03>01</R03>
   <R04>SECRET....</R04>
   <R05>ZZ</R05>
   <R06>SUBMITTERS.ID..</R06>
   <R07>ZZ</R07>
   <R08>RECEIVERS.ID...</R08>
   <R09>030101</R09>
   <R10>1253</R10>
   <R11>^</R11>
   <R12>00501</R12>
   <R13>000000905</R13>
   <R14>1</R14>
   <R15>T</R15>
   <R16>:</R16>
  </X5010:ISA>
- <X5010:GS>
   <R01>RA</R01>
   <R02>SENDER CODE</R02>
   <R03>RECEIVERCODE</R03>
   <R04>19991231</R04>
   <R05>0802</R05>
   <R06>1</R06>
   <R07>X</R07>
   <R08>005010X306</R08>
  </X5010:GS>
```

```
- <X5010:GE>
   <R01>1</R01>
   <R02>1</R02>
   </X5010:GE>
- <X5010:IEA>
   <R01>1</R01>
   <R02>000000905</R02>
   </X5010:IEA>
  </X5010:interchange>
```

Message Acknowledgements

Message acknowledgements indicate whether a message has been received and whether it is valid.

The HIX library contains services that generate the following types of EDI acknowledgments:

997 acknowledgements

Confirms the receipt and validity of a transaction.

999 acknowledgements

Confirms the receipt and validity of a transaction.

TA1 acknowledgement

Confirms the receipt and validity of the interchange envelope, such as the sender data and receiver data.

Generating 997 or 999 Acknowledgments

You can generate 997 or 999 acknowledgements from the error report output of an HIX parser service. To generate an acknowledgement, run one of the following services from the library:

- ACK_997_Generator
- ACK_999_Generator

Note: The 997 or 999 acknowledgements are relevant only to parser services.

Generating TA1 Acknowledgments

Use the TA1_Generator to generate acknowledgements. The input is the message text file, as with a parser.

Disabling Validations

Disable validations for a project in the IntelliScrip editor. You can disable a single validation type, multiple validation types, or disable all level 2 validations. The project doesn't run validations that you disable.

1. Open the main script that runs the project in the IntelliScript editor.

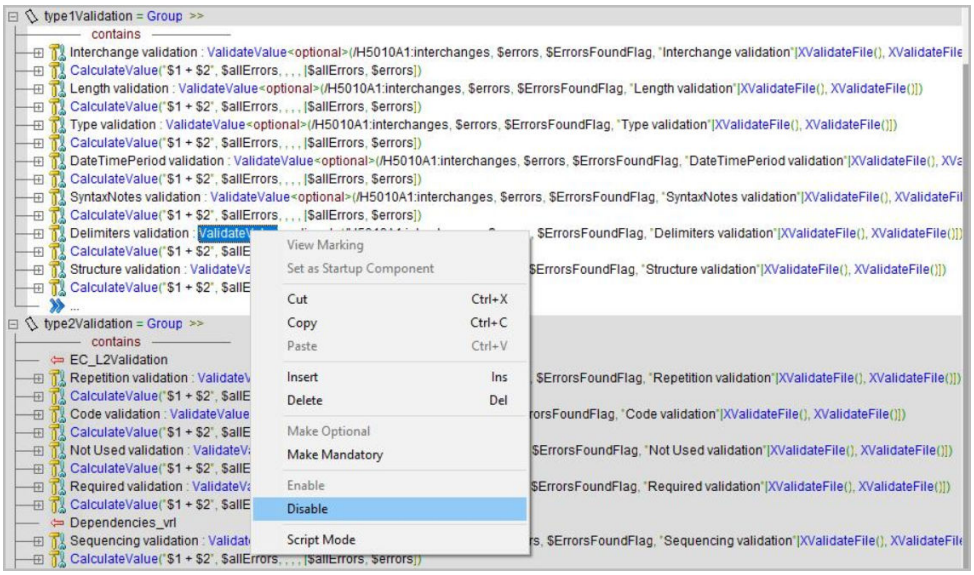
   For example, to disable validations for the HIPPA version 5010A1 parser, open the project `HIPPA_5010A1_Main_Parser_CMLIBHIX.tgp`.

2. To disable specific validation types, perform the following tasks:

   a. Based on the type of validations to disable, expand the **type1Validation** group, the **type2Validation** group, or both.

b. On the line of the validation type, right-click **ValidateValue** and select **Disable**. Disable as many validations as required.

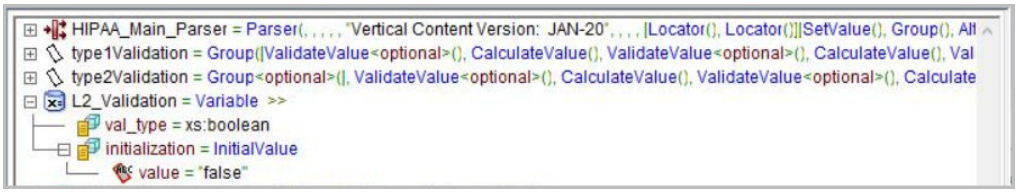
The following image shows how you disable the validation of delimiters in the **type1Validation** group:



The project doesn't run the validation types you disabled.

3. To disable all level 2 validations, expand **L2_Validation**, right-click **value**, and enter `false`.

The following images shows a project where level 2 validations are disabled:



The project doesn't run level 2 validations.

Serializer and Restricted Serializer

Trailers

If the `input.xml` file contains trailers, the Serializer will serialize the data as contained in the input. If one of the trailers is missing, such as the SE, GE, or IEA trailer, the Serializer determines which data is missing and serializes the data to the `output.txt` file.

HL Segment

The Serializer calculates the following data:

- HL01 Hierarchical ID Number, the sequencing number
- HL02 Hierarchical Parent ID Number
- HL03 Hierarchical Level Code
- HL04 Hierarchical Child Code

If the `input.xml` file contains valid data, the transformation does not generate an error. If the HL segment in the `input.xml` file contains data that is not valid, the transformation logs an error to the `Errors.xml` file and

the Serializer fixes the data. After the Serializer fixes the values, the HL segment in the `output.txt` file is valid.

HIX Validation Error Reports

When a HIX service finds validation errors in the input message, it generates an error report.

When you run a HIX service, you specify the **AdditionalOutputPort** for each of the error outputs. The following table describes the error outputs:

Name	Description
Errors	An XML listing of validation errors that the service found, including the standard X12 error codes.
ErrorsFound	Indicates whether the service found validation errors. The **errorsFound** output port can have the following values: - true. The HIX message contained validation errors. - false. The HIX message did not contain validation errors.
ValidationReport	An HTML representation of the `Errors.xml` file.

HL7 Library

The HL7 library implements the Health Level Seven version 2.x messaging standard used in the health services industry. The HL7 standard is used world-wide in hospital and medical information systems. For more information about the standard, see http://www.hl7.org.

The HL7 library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

You can use the Library editor to edit the HL7 Library object.

HL7 Message Structure

An HL7 message is composed of segments that are separated by line breaks. Each segment has a three-character label, such as `MSH` (message header) or `PID` (patient identification). Each segment contains a predefined hierarchy of fields and sub-fields. The fields are delimited by the characters immediately following the `MSH` designator, typically `|^~\&`. The message type is specified by a field in the `MSH` segment.

The following example is a message of type `ADT`, subtype `A01`, which is an `Admit a Patient` message.

In the example, the patient's name, `Smith^William^A`, follows the `PID` label by five `|` delimiters. The last and first names, `Smith` and `William`, are separated by a `^` delimiter.

For readability, we have inserted extra line breaks, in addition to the line breaks that were originally in the message.

```
MSH|^~\&|ADT1|MCM|FINGER|MCM|198808181126|SECURITY|ADT^A01|MSG00001|P|2.3.1
EVN|A01|198808181123
PID|1||PATID1234^5^M11^ADT1^MR^MCM~123456789^^^USSSA^SS||
SMITH^WILLIAM^A^III||19610615|M||C|1200 N ELM STREET^^JERUSALEM^TN^99999?
1020|GL|(999)999?1212|(999)999?3333||S||PATID12345001^2^M10^ADT1^AN^A|
123456789|987654^NC
NK1|1|SMITH^OREGANO^K|WI^WIFE||||NK^NEXT OF KIN
PV1|1|I|2000^2012^01||||004777^CASTRO^FRANK^J.|||SUR||||ADM|A0
```

XML Version of an HL7 Message

The library implements XML structures conforming to the HL7 version 2.x XML schemas. The following excerpt is an example:

```
<HL7:ADT_A01 xmlns:HL7="http://www.informatica.com/B2B/HL7">
  <HL7:MSH>
    <MSH.1>|</MSH.1>
    <MSH.2>^~\&</MSH.2>
    <MSH.3>
      <HD.1>ADT1</HD.1>
    </MSH.3>
    <MSH.4>
      <HD.1>MCM</HD.1>
    </MSH.4>
    <MSH.5>
      <HD.1>LABADT</HD.1>
    </MSH.5>
    <MSH.6>
      <HD.1>MCM</HD.1>
    </MSH.6>
    <MSH.7>
      <TS.1>198808181126</TS.1>
    </MSH.7>
    <MSH.8>SECURITY</MSH.8>
    <MSH.9>
      <MSG.1>ADT</MSG.1>
      <MSG.2>A01</MSG.2>
    </MSH.9>
    <MSH.10>MSG00001</MSH.10>
```

HL7 Message Properties

For HL7 messages, the Library editor displays the Message, Segment Group, Segment, Field, Type, and Component properties.

The following table describes the Message properties:

Property	Description
ID	Message identifier. Global settings.
Library type	HL7. Global settings.
Library version	Library version number. Global settings.
Description	Description. Global settings.
Category	Category to which the message belongs. Global settings.

The following table describes the Segment Group properties:

Property	Description
ID	Segment group identifier. Global settings.
Requirement	Mandatory or optional. Positional settings.
Repetitions	Number of times that the segment group is repeated, for example, 1 or 2. For an unlimited number, enter >1. Positional settings.

The following table describes the Segment properties:

Property	Description
ID	Segment identifier. Global settings.
Description	Description. Global settings.
Requirement	Mandatory or optional. Positional settings.
Repetitions	Number of times that the segment is repeated, for example, 1 or 2. For an unlimited number, enter >1. Positional settings.

The following table describes the Field properties:

Property	Description
ID	Field identifier, usually the ID of the parent segment followed by an index number. Global settings.
Type	Field type. Global settings.
Description	Description. Global settings.
Requirement	Mandatory or optional. Positional settings.
Repetitions	Number of times that the subsegment is repeated, for example, 1 or 2. For an unlimited number, enter >1. Positional settings.

The following table describes the Type properties:

Property	Description
ID	The type of the parent field or component. Global settings.
Description	Description. Global settings.

The following table describes the Component properties:

Property	Description
ID	Component identifier, usually the ID of the parent type followed by an index number. Global settings.
Type	Component type. Global settings.
Description	Description. Global settings.
Requirement	Mandatory or optional. Global settings.

IATA PADIS Library

The IATA PADIS library implements the Passenger and Airport Data Interchange Standards maintained by the International Air Transport Association.

The IATA PADIS library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

IATA PADIS Message Structure

An IATA PADIS message contains segments and fields separated by characters such as `'` and `+`:

```
UNH+1+APSRES:94:2:IA+X000000'
MAP+2+DL+:ITAREQ:2'
UNT+3+1'
```

XML of an IATA PADIS Message

The IATA PADIS library processes XML structures such as the following example:

```
<Interchange>
  <delimiters>
    <field_separator>+</field_separator>
    <segment_separator>'</segment_separator>
    <composite_separator>:</composite_separator>
  </Delimiters>
```

```
<Transactions>
  <ASPRES>
    <UNH>
      <R01>1</R01>
      <R02>
        <R01>ASPRES</R01>
        <R02>94</R02>
        <R03>2</R03>
        <R04>IA</R04>
      </R02>
      <R03>X000000</R03>
    </UNH>
    <LOOP_GR1>
      <MAP>
        <R01>
          <R01>2</R01>
        </R01>
        <R02>DL</R02>
        <R03>
          <R02>ITAREQ</R02>
          <R03>2</R03>
        </R03>
      </MAP>
    </LOOP_GR1>
    <UNT>
      <R01>3</R01>
      <R02>1</R02>
    </UNT>
  </ASPRES>
 </Transactions>
</Interchange>
```

IDC-IDSI Library

Interactive Data Corporation (IDC) and its subsidiary, Interactive Data Services, Inc. (IDSI), provide reference data for evaluating securities.

The Data Transformation IDC-IDSI library performs the following actions:

- Parses input data in IDSI 320-column format.
- Validates the data for the following characteristics:
  - Code
  - Format
  - Some business logic
- Converts the data to XML.

The parsers have the following output ports:

- Default output. An XML file containing the parsed input.
- Errors. An XML validation-error report.
- ErrorsFound. If the transformation encounters a validation error, returns true. If the transformation does not find an error, returns false.

IDC-IDSI Message Structure

IDC-IDSI messages have a 320-column, positional layout that includes both fixed and variable-length fields. The following example is a small excerpt of an IDC-IDSI message:

```
A0699G10 %AUKU F9999AUSTRIAMIKROSYSTEMS AG, UNTESHS                  011800003200041250004105000410500041050090 9999
A35855AB NSAPP321731SAPPI PAPIER HLDG AG        DEBT  7.500% 6/15/320119000000000785000078500007850000079000680K0615
B3344DAA NDELH409999DELHAIZE GROUP              DEBT  5.700%10/01/400119000000000977130097713009771300989068OK1001
C06840AU NBANK146081BANK NOVA SCOTIA            DEBT  1.450% 7/26/140119000000001009570100957010095701009786800V0726
C0780KAA NBARR396726BARRICK AUSTRALIA FIN PTY LTDEBT  5.950%10/15/390119000000001145710114571011457101160066801K1015
C0780KAB NBARR206726BARRICK AUSTRALIA FIN PTY LTDEBT  4.950% 1/15/200119000000001105850110585011058501105850111141680K0115
C1473XRZ NCAIS136019CAISSE CENTRALE DESJARDINS  DEBT  1.700% 9/16/130119000000001001970100197010019701002106800K0916
C14802AA NCCSI159999CCS INCORPORATED            DEBT 11.000%11/15/150119000000000990000099000009900000990006801K1115
C21847AB NCATA162621CATALYST PAPER CORP         DEBT 11.000%12/15/160119000000000645000064500006450000650006801K1215
00079FBH MSHH 146726ABN AMRO BK N V LONDON BRH  DEBT         1/16/140119000000000000000102900010290000000006A090116
00081TAB LABD 153579ACCO BRANDS CORP            NOTE  7.625% 8/15/150119000000000000000010250001025000000006A030815
001041AX NAFBT126029AFB&T ATHENS GA             CD    2.100% 9/11/120119000000001010470101047010104701010526800C0911
001041AY NAFBT126029AFB&T ATHENS GA             CD    1.400% 1/27/120119000000001000080100008010000801000116800C0127
001055AF LAFL 406321AFLAC INC                   NOTE  6.450% 8/15/400119000000000000000106608010660800000006A030815
001084AK LZZZZ143523AGCO CORP                   NOTE  6.875% 4/15/140502000000000000000000000000000000000069530502
001192AC LATG 134924AGL CAP CORP                NOTE  4.450% 4/15/130119000000000000000010324601032460000006A030415
001192AD LATG 344924AGL CAP CORP                NOTE  6.000%10/01/340119000000000000000011624001162400000006A031001
001192AE LATG 154924AGL CAP CORP                NOTE  4.950% 1/15/150119000000000000000010769401076940000006A030115
001192AF LATG 164924AGL CAP CORP                NOTE  6.375% 7/15/160119000000000000000011630401163040000006A030715
001192AG LAGL 194924AGL CAP CORP                NOTE  5.250% 8/15/190119000000000000000011286301128630000006A030815
001192AH LAGL 414924AGL CAP CORP                NOTE  5.875% 3/15/410119000000000000000011952701195270000006A030315
001192AJ LAGL 214924AGL CAP CORP                NOTE  3.500% 9/15/210119000000000000000009981900998190000006A030915
00176LEBULAMR 124512AMR CORP DEL MEDTERM NTS BE MTNF  9.140% 2/21/120119000000000000000022125002212500000006A7M0221
00508XAD LATU 173545ACTUANT CORP                NOTE  6.875% 6/15/170119000000000000000010300001030000000006A030615
00758220 WADTGNX6726ADVISORS DISCIPLINED TR     GNMA ADVINCM 2      011900000000007178000717800071780007196E80 9999
00758221 WAZVQ216726ADVISORS DISCIPLINED TR     UT GNMAINC2RR       011900000000007178000717800071780007196E80 9999
00758235 WADTNAX6726ADVISORS DISCIPLINED TR     UT TXEMT BDIC       011900000000010133001013300101330010130010129E80 9999
01556537 IASCZ X6722ALGER FDS                   SMLCAP GRWTH Z      011900000000007550000755000075500007480780 9999
01556538 IASMZ X6722ALGER FDS                   SMID CAP GRW Z      011900000000016110001611000161100015940780 9999
01556539 IALCZ X6722ALGER FDS                   LRGCAP GRWTH Z      011900000000012440001244000124400012350780 9999
01556541 IACAZ X6722ALGER FDS                   CAPTL APP FD Z      011900000000015190001519000151900015120780 9999
01556543 IAOFI X6722ALGER FDS                   GRWTH OPPTYS I      011900000000010470001047000104700010370780 9999
```

XML of an IDC-IDSI Message

The IDC-IDSI Library produces XML structures like the following example:

```
<?xml version="1.0" encoding="windows-1252"?>
<IDC:DataFeed xmlns:IDC="http://www.informatica.com/B2B/IDC_IDSI" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
  <Header>
  </Header>
  <Body>
    <Row xsi:type="IDC:IDSI320service_HeaderRecordFormatType">
      <Serviceformat>320</Serviceformat>
      <pricesevaluations>P</pricesevaluations>
      <bulletins> </bulletins>
      <dividends> </dividends>
      <PricingEvaluationdate>
        <raw>20120119</raw>
        <data>20120119</data>
      </PricingEvaluationdate>
      <Productiondate>
        <raw>20120120</raw>
        <data>20120120</data>
      </Productiondate>
      <Statementssuppliedbythecustomer>                              </
Statementssuppliedbythecustomer>
      <FileIndicators>
        <Listedbondspriceevaluationhierarchy>3</Listedbondspriceevaluationhierarchy>
        <Replacepreviousclosewithopenprice>0</Replacepreviousclosewithopenprice>
        <Replaceaveragewithbidorzero>0</Replaceaveragewithbidorzero>
        <Replaceclosingpricewithoptionsmarkingprice>0</
Replaceclosingpricewithoptionsmarkingprice>
        <DualEquityFlag>1</DualEquityFlag>
        <DualADRFlag>2</DualADRFlag>
        <PinkOTCMarketsQuoteFlag>1</PinkOTCMarketsQuoteFlag>
```

```
            <Identifyask_onlyquotes>0</Identifyask_onlyquotes>
            <FactoradjustedCMOs>0</FactoradjustedCMOs>
            <OptionsNBBOpricingcoverageflag>0</OptionsNBBOpricingcoverageflag>
            <Priceevaluationcoverageflag>1</Priceevaluationcoverageflag>
          </FileIndicators>
        </Row>
      </Body>
    </IDC:DataFeed>
```

MDM JavaBeans Library

The MDM JavaBeans library creates components that transform flat XML representations of JavaBeans components to and from hierarchical XML files.

Before you build the MDM JavaBeans library, you export the interconnected JavaBeans components from Master Data Management to a flat XML file with the XMLEncoder function in Java 6 or later. Make sure to include all of the elements that you want to transform with the library.

You use the Developer tool to create the project that contains the transformed files, Mappers, and schemas.

After you build the library, you can use the transformed files in an external tool or import them back to Master Data Management as model schemas. You can also transform the hierarchical XML files back to flat JavaBeans files with the XMLDecoder function in Java 6 or later.

The following table describes the files that are generated after the library processes the files:

File Name	DeScription
MDMJavaBeans.xml	Source XML file from Master Data Management.
<project_name>_MDMJavaBeans.xsd	Source XSD schema from Master Data Management.
input.xml	Output XML file to use as the input XML file for mappings.
<project_name>_<schema_root_name>.xsd	Output XSD schema.
<project_name>_From_MDMJavaBeans_To_XML.tgp	Script that transforms the source MDM JavaBeans file to the output XML file.
<project_name>_To_MDMJavaBeans.tgp	Script that transforms the output XML file to an MDM JavaBeans file.
<project_name>_From_MDMJavaBeans_UserDefined_Mapper.tgp	Script template for mapping the input.xml file to other message types.
<project_name>_To_MDMJavaBeans_UserDefined_Mapper.tgp	Script template for mapping other message type to the input.xml file.

After you transform the MDM JavaBeans file, you can use the MDM Mappers library to translate between the output XML file and an ACORD LNA file. You use the Developer tool to import and run the MDM Mappers. For more information about the MDM Mappers library, see "ACORD-MDM Mappers Library" on page 22.

Source JavaBeans XML File Structure

A JavaBeans XML file contains representations of the JavaBeans components that export from the interconnected JavaBeans model.

The following example shows the XML structure of an MDM JavaBeans file:

```
<object class="com.infa.b2b.INFA_ACORD_MDM">
  <void property="agreemt">
   <array class="com.siperian.inservice.bo.Agreemt" length="2">
    <void index="0">
     <object class="com.siperian.inservice.bo.Agreemt">
      <void property="agreemtype">
       <string>testvalue</string>
      </void>
      <void property="amount">
       <string>testvalue</string>
      </void>
      <void property="approvaldate">
       <object class="java.util.Date">
        <long>1330027204389</long>
       </object>
      </void>
      <void property="createDate">
       <object class="java.util.Date">
        <long>1330027204386</long>
       </object>
      </void>
      <void property="creator">
       <string>testvalue</string>
      </void>
      <void property="currencycode">
       <string>testvalue</string>
      </void>
      <void property="effectiveperiod">
       <string>testvalue</string>
      </void>
      <void property="identifier">
       <string>testvalue</string>
      </void>
      <void property="issuedate">
       <object class="java.util.Date">
        <long>1330027204389</long>
       </object>
      </void>
      <void property="lastUpdateDate">
       <object class="java.util.Date">
        <long>1330027204386</long>
       </object>
      </void>
      <void property="lineofbusinesscode">
       <string>testvalue</string>
      </void>
     </object>
    </void>
   </array>
  </void>
 </object>
</java>
```

Output XML File Structure

The MDM JavaBeans library transforms the source MDM JavaBeans XML file to an output XML file with the name `input.xml`. You use the output file as the input for transformations and mappings. For example, you

can use the ACORD-MDM Mappers library to translate the XML file to an ACORD LNA file with the Developer tool.

The following example shows the XML structure of an `input.xml` file:

```
<agreemt>
    <MDM:com.siperian.inservice.bo.Agreemt>
        <agreemtype>testvalue</agreemtype>
        <amount>testvalue</amount>
        <approvaldate>
            <MDM:java.util.Date>
                <long>1330027204389</long>
            </MDM:java.util.Date>
        </approvaldate>
        <createDate>
            <MDM:java.util.Date>
                <long>1330027204386</long>
            </MDM:java.util.Date>
        </createDate>
        <creator>testvalue</creator>
        <currencycode>testvalue</currencycode>
        <effectiveperiod>testvalue</effectiveperiod>
        <identifier>testvalue</identifier>
        <issuedate>
            <MDM:java.util.Date>
                <long>1330027204389</long>
            </MDM:java.util.Date>
        </issuedate>
        <lastUpdateDate>
            <MDM:java.util.Date>
                <long>1330027204386</long>
            </MDM:java.util.Date>
        </lastUpdateDate>
        <lineofbusinesscode>testvalue</lineofbusinesscode>
    </MDM:com.siperian.inservice.bo.Agreemt>
</agreemt>
```

Installing the MDM Library

Before you create an MDM Java Beans service, you must install the library, and then restart the Developer tool.

1. To install the library, copy the `MDM_JavaBeans.zip` file to your local file system and unzip the file.
2. From the unzipped folder, copy the file `MDM_import.tgp` to the directory `<Install_Dir>\<version>\clients\DT\autoInclude\system`.
3. Copy the folder `CmInternal_MDM_JavaBeans` to the directory `<Install_Dir>\<version>\clients\DT\bin\internalServices`.

Creating an MDM JavaBeans Transformation

Create the MDM JavaBeans transformation in the Developer tool after you install the MDM JavaBeans library.

1. In the Developer tool, click **File** > **New** > **Transformation**.
2. Select the Data Processor transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model Repository location to put the transformation.
4. Select **Create a data processor using a wizard** and click **Next**.
5. Select the MDM JavaBeans input format and click **Next**.
6. Browse to select an MDM JavaBeans sample file and click **Next**.

7. Select an output format and click **Next**.
8. Click **Finish**.

   The Developer tool creates the transformation in the repository. The transformation contains the source files, the output files and the mapper scripts. The transformation appears in the **Overview** view.
9. In the **Settings** view **Output** panel, select **Disable automatic output**.
10. To run the Streamer, in the **Overview** view, select the Streamer as the startup component.

After you create the transformation, you can use the MDM Mappers library to translate the output XML file to and from an ACORD LNA file. Use the Developer tool to import the MDM mappers.

NACHA Library

The NACHA library implements the Automated Clearing House standard maintained by the NACHA Electronic Payments Association and used by financial institutions.

The NACHA library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

For more information, see http://www.nacha.org.

NACHA Message Structure

A NACHA message contains positional records. The following excerpt illustrates a NACHA message:

```
101 091000022 1130006090208240516A094101US BANK CORP PAY SYS   CHASE OF
TX
5200BEST WIDGETS USA                    1234567890CTXPAYMENT
020828240111300060000066
602152545851FDHHDFHDFHDFHFHFH1254565236              1254ADRFVFRTG5467HYJ
1125456523658745
602152545851FDHHDFHDFHDFHFHFH1254565236              1254ADRFVFRTG5467HYJ
1125456523658745
```

XML of a NACHA Message

The NACHA library processes XML structures such as the following example:

```
<File>
  <FileHeaderRecord>
    <PriorityCode>1</PriorityCode>
    <ImmediateDestination>91000022</ImmediateDestination>
    <ImmediateOrigin>113000609</ImmediateOrigin>
    <FileCreationDate>20824</FileCreationDate>
    <FileCreationTime>516</FileCreationTime>
    <FieldIdModifier>A</FieldIdModifier>
    <RecordSize>094</RecordSize>
    <BlockingFactor>10</BlockingFactor>
    <FormatCode>1</FormatCode>
    <ImmediateDestinationName>US BANK CORP PAY SYS</ImmediateDestinationName>
    <ImmediateOriginName>CHASE OF TX</ImmediateOriginName>
    <ReferenceCode />
  </FileHeaderRecord>
  <Batch>
    <BatchHeaderRecord>
      <ServiceClassCode>200</ServiceClassCode>
      <Name>BEST WIDGETS USA</Name>
      <DiscretionaryData />
      <Identification>1234567890</Identification>
```

```
  <StandardEntryClassCode>CTX</StandardEntryClassCode>
  <EntryDescription>PAYMENT</EntryDescription>
  <DescriptiveDate />
  <EffectiveEntryDate>20828</EffectiveEntryDate>
  <SettlementDateJulian>240</SettlementDateJulian>
  <OriginatorStatusCode>1</OriginatorStatusCode>
  <OriginatingDfiIdentification>11300060</OriginatingDfiIdentification>
  <BatchNumber>666</BatchNumber>
</BatchHeaderRecord>
<Record>
  <CTX>
    <CTXEntryDetailRecord>
      <TransactionCode>2</TransactionCode>
      <ReceivingDfiIdentification>15254585</ReceivingDfiIdentification>
      <CheckDigit>1</CheckDigit>
      <DfiAccountNumber>FDEHHFDHDFHDFHFHFH</DfiAccountNumber>
```

NCPDP Library

The National Council for Prescription Drug Programs (NCPDP) develops standards for information processing in the pharmacy services sector of the health care industry. Insurance companies use NCPDP transactions to process pharmacy drug claims.

The Data Transformation NCPDP library implements two NCPDP standards for communicating pharmacy claims:

- NCPDP Batch Transaction Standard versions 1.1 and 1.2. Used to transmit multiple claims in batch format.
- NCPDP Telecommunication Standard Format versions 5.1 and D.0. Used to transmit individual claims from the point-of-sale to the insurance carrier.

The NCPDP library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

For more information about NCPDP, see http://www.ncpdp.org.

NCPDP Message Structure

NCPDP messages have a positional layout containing both fixed and variable-length fields. The segments and fields are delimited by control characters such as STX (ASCII 0x02), ETX (0x03), FS (0x1C), GS (0x1D), and RS (0x1E).

The following illustration is a fragment of an NCPDP billing message.

```
STX00TCOBA                    3300714200711262141P51LOUIPROD                ETX
STXG10000000019018003518135392101040537670280    200710171111111111
RSFSAM01FSCX99FSCY434562240MFSC419410205FSC51FSCAOFSCBJACKSONFSCM323 PENNSYLVAN
RSFSAM04FSC2434562240MFSCCOFSCDJACKSONFSC61
GSRSFSAM07FSEM1FSD26666032FSE103FSD749502068531FSSE1FSERKOFSE7360000FSD33FSD53
RSFSAM03FSEZ06FSDBF47947FSDRSTELL
RSFSAM05FS4C1FS5C01FS6C99FS7C18003FSE820071108FSHB4FSHC08FSDV108CFSHC07FSDV135
RSFSAM11FSDC00{FSDX27DFSDU136H
RSFSAM12FSPA1FSPB20071017FSPC20071017FSPDPRFSPY0FSPPSAC
```

XML of an NCPDP Message

The NCPDP library processes XML structures such as the following example:

```
<File>
  <Batch>
    <BatchHeaderRecord>
      <TransmissionType>Transaction</TransmissionType>
      <SenderID>COBA</SenderID>
      <BatchNumber>3300714</BatchNumber>
      <CreationDate>20071126</CreationDate>
      <CreationTime>2141</CreationTime>
      <FileType>Production</FileType>
      <VersionReleaseNumber>51</VersionReleaseNumber>
      <ReceiverID>LOUIPROD</ReceiverID>
    </BatchHeaderRecord>
    <DetailDataRecord>
      <TransactionReferenceNumber>0000000019</TransactionReferenceNumber>
      <TransmissionTransactionType="Billing">
        <Header>
          <BinNumber>18003</BinNumber>
          <VersionReleaesNumber>51</VersionReleaesNumber>
```

SEPA Library

The SEPA library implements the Single Euro Payments Area messaging standard used in the banking industry. The library validates messages according to the formatting rules of the European Payments Council and the rules defined in ISO 20022. The library does not support SEPA usage rules.

The parsers have the following output ports:

- Default output. An unaltered copy of the input SEPA message.
- Errors. An XML validation-error report.
- ErrorsFound. If the transformation encounters a validation error, returns true. If the transformation does not find an error, returns false.

The SEPA transformation projects are grouped in the following categories:

- SCT. SEPA Credit Transfer.
- SDD_CORE. SEPA Direct Debit Core.
- SDD_B2B. SEPA Direct Debit Business-to-Business.

The SEPA project names have the following suffixes, which correspond to the SEPA implementation guides:

- IB. Interbank.
- C2C. Customer-to-customer.
- B2C. Bank-to-customer.
- EM. e-Mandate Service.

For more information about SEPA, see:

- http://www.iso20022.org/
- http://www.europeanpaymentscouncil.eu/

SEPA Validation Error Reports

When an SEPA service finds validation errors in the input message, it generates an error report.

When you run an SEPA service, you specify the **AdditionalOutputPort** for each of the error outputs. The following table describes the error outputs:

Name	Description
Errors	An XML listing of validation errors that the service found, including the standard SEPA error codes.
ErrorsFound	Indicates whether the service found validation errors. The **errorsFound** output port can have the following values: - true. The SEPA message contained validation errors. - false. The SEPA message did not contain validation errors.

SWIFT Library

SWIFT is a messaging standard for the financial industry, maintained by the Society for Worldwide Interbank Financial Telecommunication. The standard defines messages for purposes such as electronic funds transfer and check processing. For more information, see http://www.swift.com/. There are two versions of the standard:

- SWIFT MT, for text-based messages
- SWIFT MX, for XML-based messages

The Data Transformation SWIFT library implements transformations that process MT and MX messages. The SWIFT organization has certified the Data Transformation implementation for use with SWIFT versions 2008, 2009, 2010, and 2011. SWIFT has authorized Data Transformation to display the SWIFT-Ready Application logo.

For MT messages, the library provides parsers, serializers, schemas, and validation components. For MX messages, the library provides validation components.

You can use the Library editor to edit the SWIFT Library object.

SWIFT MT Message Structure

The following excerpt illustrates a SWIFT MT transaction. The example is a Request for Transfer message:

```
{1:F01DEUTDEFFEDIX0317000005}{2:I101ABCDEFGHIJKAU1020}{4:
:20:Sender Ref Dom01
:21R:Customer Ref
:28D:1/1
:50L:Instructing Party
:50H:/00190020974010046074
DB SPAIN
:30:030120
:21:REFF21-1
:32B:EUR1,00
:57D:/ES999999999
Madrid
:59:/64930056505
DB Spain
Barcelona
:70:First payment11
:71A:OUR
```

```
:21:REFF21-2
:32B:EUR2,00
:57D:/E999999999
Madrid
:59:/64930056506
DB Spain
Barcelona
:70:Second payment2
:71A:OUR
```

XML Version of a SWIFT MT Message

The SWIFT library parsers transform SWIFT MT messages to XML structures such as the following example. The serializers transform the XML representation to the native MT representation.

```
<MT101>
  <BasicHeaderBlock>
    <ApplicationIdentifier>F</ApplicationIdentifier>
    <ServiceIdentifier>01</ServiceIdentifier>
    <LTAddress>
    </LTAddress>
    <SessionNumber></SessionNumber>
    <SequenceNumber></SequenceNumber>
  </BasicHeaderBlock>
  <ApplicationHeaderInputBlock>
    <Indicator>I</Indicator>
    <MessageType>101</MessageType>
    <DestinationAddress>
      <BankCode>ABCD</BankCode>
      <CountryCode>EF</CountryCode>
      <LocationCode>GH</LocationCode>
      <LogicalTerminalCode>1</LogicalTerminalCode>
      <BranchCode>JKA</BranchCode>
    </DestinationAddress>
    <MessagePriority>U</MessagePriority>
    <DeliveryMonitoring>1</DeliveryMonitoring>
    <ObsolescencePeriod>020</ObsolescencePeriod>
  </ApplicationHeaderInputBlock>
```

SWIFT Message Properties

For SWIFT messages, the Library editor displays the Message, Sequence, Field, Option, Line, Alternative Layout, Code Group, Code, and Fixed Component properties.

The following table describes the Message properties:

Property	Description
ID	Message identifier. Global settings.
Name	Message name. Global settings.

The following table describes the Sequence properties:

Property	Description
Name	Sequence name. Global settings.
Repetitions	R if repetitions are allowed, or S if only single occurrences are allowed. Positional settings.
Requirement	Mandatory or optional. Positional settings.

The following table describes the Field properties:

Property	Description
Code	Identifier for the field. Global settings.
Name	Field name. Global settings.
Letter options	Letters such as A, B identifying the valid Option elements nested within the field. Global settings.
Repetitions	R if repetitions are allowed, or S if only single occurrences are allowed. Positional settings.
Requirement	Mandatory or optional. Positional settings.

The following table describes the Option properties:

Property	Description
Letter option	A letter such as A or B identifying the option. Global settings.

The following table describes the Line properties:

Property	Description
Repetitions	The number of repetitions allowed for the line. Global settings.

The following table describes the Alternative Layout properties:

Property	Description
(None)	An alternative layout element has no editable properties.

The following table describes the Component properties:

Property	Description
Name	Component name. Global settings.
Prefix	A prefix such as / preceding the data. Global settings.
Length	Length of the data. Global settings.
Fixed	If true, the component has a fixed length defined by the Length property. If false, the Component has a variable length. The Length property defines the maximum length. Global settings.
Char set	Character set used to encode the data. Global settings.
Requirement	Mandatory or optional. Global settings.

The following table describes the Code Group properties:

Property	Description
Description	A comment describing the valid values of a component. Global settings.
Error code	The error code to report in the event of a validation error. Global settings.
Qualifier dependency	A value of a qualifier component. The code group is valid only if the qualifier has the specified value. Global settings.

The following table describes the Code properties:

Property	Description
Value	A valid value for a component. Global settings.
Name	The name of the value. Global settings.
Description	A comment describing the value. Global settings.
Order	Integer identifier of a mutually exclusive set of values. Two codes having the same Order cannot appear together. Global settings.
Requirement	Mandatory or optional. Global settings.
Repetitive	If true, the value can appear more than once. Global settings.

The following table describes the Fixed Component properties:

Property	Description
Name	Fixed component name. Global settings.
Requirement	Mandatory or optional. Global settings.
Format	Layout of the fixed component. Global settings.

Installing Lookup Tables

Before you can use the SWIFT library, you must install lookup tables for BIC, country, and currency codes.

Sample Lookup Tables

For licensing reasons, the SWIFT library cannot provide production versions of the lookup tables. Instead, the library installs a set of sample lookup tables that you can use for testing. You can find the tables in the following directory:

```
<INSTALL_DIR>\DataTransformation\ServiceDB\SWIFT_Lookup
```

Generating and Installing Production Lookup Tables

The library contains a component that generates production versions of the lookup tables, from directory files that you can download from SWIFT. The production tables replace the sample tables in the `SWIFT_Lookup` directory.

1. Get the BIC directories from the following URL: http://www.swift.com/
2. In the Developer tool, create a Data Processor transformation.
3. In the Data Processor transformation **Objects** view, click **New.**
4. Select **Library** and click **Next.**
5. Browse to select the **Create_SWIFT_Lookups** message type.
6. Click **Finish**.
7. Export the Data Processor transformation as a Data Transformation service.
8. Open a command prompt, and then enter the following command:

   ```
   CM_console <service_name> -f<name of SWIFT directory file>
   ```

   For example, enter:

   ```
   CM_console Create_SWIFT_Lookups -fc:\CT_20090502.TXT
   ```

   An XML file appears in the current directory, for example, `CountryCode_Lookup.xml`.
9. Copy the XML file to the `SWIFT_Lookups` directory in the Data Transformation repository.

   The file overwrites the sample file of the same name.

Message Splitter

A dedicated Script named MTnnn_splitter.tgp splits MT messages 940, 942 and 950 when they are longer then the maximum message length. The Script is available in the restricted Serializer project.

The SWIFT UHB defines the logic used to split the messages. By default, the message is not partitioned. To split a message, set the value of the variable **SplitLargeMessage** to **true** in MTnnn_splitter.tgp.

The output is split into separate output files by the following method:

- The output XML contain a list of message parts, the file names, and the file path. By default, the `Results` folder contains the output files. To change the output path, set the value of **vOutputPath**.
- The output file names are determined by the value of the message field 20, a sequence part number, and a flag that identifies whether the file contains the last part. For example, a message that is split into five parts will have the following file names:
  12345TraRF123456_1_N.txt

  12345TraRF123456_2_N.txt

  12345TraRF123456_3_N.txt

  12345TraRF123456_4_N.txt

  12345TraRF123456_5_Y.txt

  To change the logic for file names, change the rules in the splitter group section **createUniqueFileName**.
- Field 103, the Message User Reference field, is updated by default. If the value is set, it is the same for all message parts. To change the setting, enable the MUR logic in the group section **calculateMessageUserReference**, or create proprietary logic.

Note: If the variable **SplitLargeMessage** is set to **true**, but the message length is within the maximum length limit, the Script splits the message with the splitter logic defined in this section.

To transform the MT940, MT942, or MT950 messages with the Developer tool, ensure that the `SWIFT_Lookups` folder is in the folder named `C:\Users\<user>\AppData\Local\Temp\2\DTTemp`. If needed, copy the folder from the ServiceDB to the folder `C:\Users\<user>\AppData\Local\Temp\2\DTTemp`. If you run the service from outside the ServiceDB, you need to ensure that the `SWIFT_Lookups` folder is in the same path as the project.

If message MT940 generates validation error code C24 during the split procedure, then the splitter fails. The C24 error code has the following description:

C24 error code

> *If field 86 is present in any occurrence of the repetitive sequence, it must be preceded by a field 61. In addition, if field 86 is present, it must be present on the same page or message of the statement as the related field 61.*

SWIFT MT Parsers and Serializers

The `MT` directory of the SWIFT library contains Parsers and Serializers for MT messages.

Parsers

The library parsers use a `ValidateValue` action to validate their input. The action is configured with one or more Validation Rule Language (VRL) files.

For the following message types, SWIFT requires strict validation for Messaging Data Services (MDS) certification. The VRL rules enforce the complete set of validations required by the SWIFT standard:

101, 102+, 102, 103+, 103, 104, 105, 110, 111, 112, 200, 201, 202, 202COV, 203, 204, 205, 205COV, 210, 300, 304, 305, 320, 321, 330, 340, 341, 350, 360, 362, 380, 381, 400, 410, 412, 420, 422, 430, 450, 456, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 513, 514, 515, 516, 517, 518, 519, 524, 526, 530, 527, 535, 536, 537, 538, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 558, 559, 564, 565, 566, 567, 568, 569, 575, 576, 578, 586, 600, 601, 604, 605, 606, 607, 608, 609, 620, 670, 671, 700, 701, 705, 707, 710, 711, 720, 730, 732, 734, 740, 742, 747, 750, 752, 754, 756, 760, 767, 768, 769, 800, 801, 802, 900, 910, 920, 935, 940, 941, 942, 950, 960, 961, 962, 963, 964, 970, 971, 972, 973, n90, n91, n92, n95, n96, n98, n99

For other message types, SWIFT does not require strict validation. The Validation Rules are correspondingly less strict.

If a parser finds a validation error, it does not fail. Instead, it writes data to the following output ports:

- Default output. Returns the transformed SWIFT message.
- Errors. An XML validation-error report.
- ErrorsFound. If the transformation encountered a validation error, returns true. If the transformation did not find an error, returns false.

For more information about `ValidateValue` and Validation Rules, see the *Data Transformation User Guide*.

Enabling or Disabling VRL Rules

By default, the library enables all the VRL rules. In the Developer tool, you can view the rules. You can choose which rules to enable or disable.

To view or disable VRL rules:

1. In the **Objects** view for the Data Processor transformation, double-click a Validation Rule object.

   The Validation Rule editor displays the Validation Rule object structure.

2. Select a rule from the Validation Rule object structure.

   The Validation Rule editor displays the rule properties in the **Properties** pane.

3. To disable the rule, in the **Properties** pane, clear the **Enable** field.

Serializers that Perform Comprehensive Validation

Each SWIFT MT library directories contain serializers having names ending in `_restricted_serializer`. For example, an MT101 project contains a serializer called `MT101_restricted_serializer`.

These serializers perform strict schema validation on their input. If a `_restricted_serializer` finds a validation error, it does not fail. Instead, it writes data to the following output ports:

- **Default output.** Returns the transformed SWIFT message.
- **Errors.** An XML validation-error report.
- **ErrorsFound.** If the transformation encountered a validation error, returns true. If the transformation did not find an error, returns false.

Serializers that Perform Schema Validation

In addition to the `_restricted_serializer` components, the MT library directories contain serializers having names ending in `_serializer`. These serializers perform a lesser degree of validation.

If a `_serializer` finds a validation error, the transformation fails.

MX Validators

For each MX message type, the `MX` directory of the SWIFT library contains a validation project.

Each project contains a parser that processes an input MX XML message. The parser contains a `ValidateValue` action and VRL rules that validate the input.

The parsers have the following output ports:

- Default output. An unaltered copy of the input MX message.
- Errors. An XML validation-error report.
- ErrorsFound. If the transformation encountered a validation error, returns true. If the transformation did not find an error, returns false.

Telekurs VDF Library

Telekurs Financial is a financial information provider headquartered in Switzerland. It provides a Valordata Feed (VDF) service delivering financial data. The Telekurs VDF library processes the message structures used in the VDF service.

The VDF library transformations do not validate their input. If the input is invalid, the transformations might fail or they might generate unexpected output.

For more information about VDF, see the following URL:

http://www.telekurs-financial.com/tkfich_index/tkfich_products/tkfich_products_processing_products/tkfich_products_processing_products_vdf.htm

Telekurs VDF Message Structure

A VDF message contains segments and fields, delimited by characters such as ' and :. The following excerpt illustrates a VDF message:

```
FAD+207:69361602:4::TKFAD:6:1+20071112:093426952'VAI+S:KK+CH:8784173'MGA+S:SSSS
+1+1+20071112'OFI
+S:SSSSSSSSSSSSSSS+CH:472672+P+290+20080215++3472672+75447079+1+1+100++2+CH:472672'FMB
+S:SSSSSS
SSSSSSSSS+8++4+1++3+20071112++++++2+501'VAK+S:KKSSSS+TKN:4+NOKIA P FEB290 08+Stockoption
+Put
```

XML of a Telekurs VDF Message

The Telekurs VDF library processes XML structures such as the following example:

```
<TKFAD>
  <Record>
    <INSTRUMENT_ADMINISTRATION_DOC_HEADER1_1>
      <DEFINITION_OF_DOC_TRANSACTION1_1_1>
        <VDF_Release_number1_1_1_1>207</VDF_Release_number1_1_1_1>
        <DOC_transaction_ID1_1_1_2>69361602</DOC_transaction_ID1_1_1_2>
        <DOC_transaction_type1_1_1_3>4</DOC_transaction_type1_1_1_3>
        <DOC_transaction_flag1_1_1_4 />
        <DOC_ID1_1_1_5>TKFAD</DOC_ID1_1_1_5>
        <DOC_generation1_1_1_6>6</DOC_generation1_1_1_6>
        <DOC_processing_status1_1_1_7>1</DOC_processing_status1_1_1_7>
      </DEFINITION_OF_DOC_TRANSACTION1_1_1>
      <DEFINITION_OF_DOC_TIME_STAMP_GMT1_1_2>
        <Date1_1_2_1>20071112</Date1_1_2_1>
        <Time1_1_2_2>093426952</Time1_1_2_2>
      </DEFINITION_OF_DOC_TIME_STAMP_GMT1_1_2>
    </INSTRUMENT_ADMINISTRATION_DOC_HEADER1_1>
    <INSTRUMENT_IDENTIFICATION1_2>
      <SEGMENT_ELEMENT_QUALIFIER1_2_1>
        <Segment_qualifier1_2_1_1>S</Segment_qualifier1_2_1_1>
        <Element_qualifier1_2_1_2>KK</Element_qualifier1_2_1_2>
      </SEGMENT_ELEMENT_QUALIFIER1_2_1>
```

Thomson Reuters Library

The Thomson Reuters library validates Thomson Reuters reports in XML format. If the report is in a delimited text format, the library converts it to XML and then validates it. If the report is XML, the library modifies the XML before validation.

Reports in text format can be delimited by the following characters:

- Comma
- Pipe
- Semicolon
- Tab

The Thomson Reuters library performs the following types of validation:

- Field datatype and width
- Business rules defined by Informatica
- Codes defined by Thomson Reuters

Validation of Thomson Reuters Corporate Actions reports is based on the SWIFT 2011 specification.

The following table describes the files that the service produces:

File	Description
output.xml	Contains all of the data in the original Thomson Reuters report.
Errors.xml	Contains a detailed description of errors in the input.
ErrorsFound.txt	Boolean flag that indicates whether the service detected errors in the input.

Thomson Reuters Report Structure

A Thomson Reuters report is made up of a header section, a column header section, a data section, and a trailer section. The header section, the column header section, and the trailer section are optional.

The following example of a comma-delimited Thomson Reuters report shows a header section, a column header section, a data section, and a trailer section:

```
06.11.2011 05:05:10
06.11.2011 05:05:14
3
Asset ID,Asset Type,Bridge Symbol,Capitalization Amount,Capitalization Effective
Date,Coupon Rate
0x000019000023013b,CORP,US*;CPK-8.25-0314,,,8.25
0x00038600d5d26def,OTHR,GB*;WIGAN-3-PRP,,,3
0x0003e70125312cdd,CORP,CA*;ALMFC-4.35-0417,,,4.35
First trailer line
Second trailer line
```

XML Version of a Thomson Reuters Report

The following example shows the XML version of a Thomson Reuters report:

```
<?xml version="1.0" encoding="windows-1252"?>
<TR:DataFeed xmlns:TR="http://www.informatica.com/B2B/ThomsonReuters"
current_date="20111124">
  <Header>
    <Row>
      <Start_Time>06.11.2011 05:00:57</Start_Time>
    </Row>
    <Row>
      <End_Time>06.11.2011 05:00:58</End_Time>
    </Row>
    <Row>
    <Row_Count>3</Row_Count>
    </Row>
  </Header>
  <Body>
    <Row>
      <Asset_ID>0x000019000023013b</Asset_ID>
      <Asset_Type>CORP</Asset_Type>
      <Bridge_Symbol>US*;CPK-8.25-0314</Bridge_Symbol>
      <Capitalization_Amount></Capitalization_Amount>
      <Capitalization_Effective_Date></Capitalization_Effective_Date>
      <Coupon_Rate>8.25</Coupon_Rate>
    </Row>
    <Row>
      <Asset_ID>0x00038600d5d26def</Asset_ID>
      <Asset_Type>OTHR</Asset_Type>
      <Bridge_Symbol>GB*;WIGAN-3-PRP</Bridge_Symbol>
      <Capitalization_Amount></Capitalization_Amount>
      <Capitalization_Effective_Date></Capitalization_Effective_Date>
      <Coupon_Rate>3</Coupon_Rate>
    </Row>
    <Row>
```

```
      <Asset_ID>0x0003e70125312cdd</Asset_ID>
      <Asset_Type>CORP</Asset_Type>
      <Bridge_Symbol>CA*;ALMFC-4.35-0417</Bridge_Symbol>
      <Capitalization_Amount></Capitalization_Amount>
      <Capitalization_Effective_Date></Capitalization_Effective_Date>
      <Coupon_Rate>4.35</Coupon_Rate>
    </Row>
  </Body>
  <Trailer>
    <Row>
      <FixedValue>First trailer line</FixedValue>
    </Row>
    <Row>
      <FixedValue>Second trailer line</FixedValue>
    </Row>
  </Trailer>
</TR:DataFeed>
```

Installing a Thomson Reuters Library

Before you create a Thomson Reuters service, you must install the Thomson Reuters library, and then restart the Developer tool.

Note: If you are running a previous version, you can update the library from this release 10 .ZIP file, following the Installing Libraries steps in the *Data Transformation Libraries Guide*.

1. To install the library, copy the Thomson Reuters library .ZIP file to your local file system and unzip the file.
2. From the unzipped folder, copy the file `TR_import.tgp` to the directory `<Install_Dir>\<version>\clients\DT\autoInclude\system`.
3. Copy the folder `CmInternal_Reuters` to the directory `<Install_Dir>\<version>\clients\DT\bin\internalServices`.
4. Copy the folder `Gcodes_Lookups` to the `ServiceDB` directory.

Creating a Thomson Reuters Transformation

Before you create a Thomson Reuters transformation, you must install the Thomson Reuters library, and then restart the Developer tool.

1. In the Developer tool, click **File** > **New** > **Transformation**.
2. Select the Data Processor transformation and click **Next**.
3. Enter a name for the transformation and browse for a Model Repository location to put the transformation.
4. Select **Create a data processor using a wizard** and click **Next**.
5. Select the Thomson Reuters input format. Select one of the following types:
6. Click **Next**.
7. Browse to select a Thomson Reuters DataScope Select definition file. Click **Next**.
8. Select an output format and click **Next**.
9. Click **Finish**.

   The Developer tool creates the transformation in the repository for the Thomson Reuters message type with a Parser and Streamer. The transformation appears in the **Overview** view.
10. In the **Settings** view **Output** panel, select **Disable automatic output**.
11. To run the Streamer, in the **Overview** view, select the Streamer as the startup component.

CHAPTER 4

Generate Library Objects

This chapter includes the following topics:

Generate Library Objects Overview

Advanced users might want to change the way that a message type is transformed. A Library transformation contains a large number of objects and components, such as Scripts with Parsers, Serializers, and XML schemas, that define the transformation. A Library transformation might contain objects for message validation, acknowledgments, and diagnostic displays.

To access and edit the Scripts, XMaps and schemas that you create with the Library editor, you must generate the Library objects. After you generate the Library objects, use the IntelliScript editor to edit Parsers, Serializers, and Mappers. For example, you want to change the output structure to suit your requirements. You generate the Library objects and edit a Parser with the IntelliScript editor.

After you generate Library objects, or if the Library objects were pre-generated, you cannot edit the Library elements with the Library editor. To use the Library editor again, you must discard the generated Library objects for those libraries that you can edit with the Library editor. For example, you might decide that you want to add input fields but not change the structure of the output. Discard the Library objects and after that add custom elements with the Library editor.

Note: Any changes that you made to the generated Library objects are lost when you discard the generated objects.

Generating the Library Objects

Generate the Library objects so that you can access them directly with the Data Processor transformation editors. After you generate the Library objects, you cannot use the Library editor to edit the Library.

You must generate Library objects to access and edit the pre-configured parsers, mappers, serializers, and XML schemas associated with the Library.

1. In the **Objects** view, right-click the Library and select **Generate Library Objects.**

2. To access the Library objects, select **Yes**.

   The Developer tool creates a **Generated Library Objects** folder that contains the Library objects. The startup component is generated.

   Note: If you want to change which Library objects are generated, select a different component.
3. To edit a Script or schema, select it and click **Open**.

   Use the IntelliScript editor to edit the Script.

Discarding the Library Objects

To edit the Library elements with the Library editor, you must remove the Library objects that you generated. When you discard the Library objects, any change that you made to the them is lost.

1. In the **Objects** view, right-click the Library and select **Discard Generate Library Objects.**
2. To discard the Library objects from the Data Processor transformation editors, select **Yes**.

   The **Generated Library Objects** folder that is discarded. The Library components and objects still exist, but are not accessible through the Data Processor transformation editors.

Testing the Transformation

Test the Data Processor transformation in the **Data Viewer** view.

Before you test the transformation, verify that you defined the startup component. You can define the startup component in a Script or you can select the startup component on the **Overview** tab. You also need to have chosen an example input file to test with.

1. Open the **Data Viewer** view.
2. Click **Run**.

   The Developer tool validates the transformation. If there is no error, the Developer tool shows the example file in the **Input** area. The output results appear in the Output panel.
3. Click **Show Events** to show the **Data Processor Events** view.
4. Double-click an event in the **Data Processor Events** view in order to debug the event in the Script editor.
5. Click **Synchronize with Editor** to change the input file when you are testing multiple components, each with a different example input file.

   If you modify the example file contents in the file system, the changes appear in the **Input** area.

INDEX

I

L

M

N

O

P

R

S