IBM InfoSphere DataStage
Version 9 Release 1

XML Pack Guide

IBM
**Contents**

**Chapter 1. Introduction**
- About XML Pack
- Components
- About XML Input
- About XML Output
- About XML Transformer
- Deriving and parsing XML hierarchies
- About the XML Meta Data Importer

**Chapter 2. Using the XML Meta Data Importer**
- Converting XML data types
- Exploring the XML Meta Data Importer window
- Examining parser output
- Getting started with XML Meta Data Importer
- Opening an XML source
- About namespaces
- Processing XML documents
- Processing XML schemas
- Mapping nodes from an XML schema
- Choosing individual items
- Selecting all nodes
- Modifying table definitions
- Modifying XPath expressions
- Accessing the Edit XPath Expression dialog box
- Saving the table definition

**Chapter 3. Using XML Input**
- About transforming XML documents
- Identifying XML sources
- About National Language Support (NLS)
- Validating documents and schemas
- Schema scan and schema validation
- Setting XML schema options
- Mapping transformation errors to InfoSphere DataStage errors
- Using Reject links
- Passing data from input to output links
- Controlling output rows
- Controlling the number of output rows
- Processing NULLs and empty values
- Processing namespaces
- Performing advanced transformations
- Getting started with XML Input
- Creating table definitions with XML Meta Data Importer
- Adding an XML Input stage to your server job
- Setting up properties for the XML Input stage
- Stage properties
- Input Link properties
- Output Link properties

**Chapter 4. Using XML Output**
- About transforming tabular data
- About National Language Support (NLS)
- Supported XPath expressions
- Using XPath expressions
- Parsing XML reserved and special characters
- Validating documents and schemas
- Setting XML schema options
- Mapping validation errors to InfoSphere DataStage errors
- Using Reject links
- Aggregating input rows on output
- Example
- Writing output to your file system
- Generating output files for multiple columns
- Generating output files for rows
- File paths on the output link
- Processing NULLs and empty values
- Selecting items for the XML Output
- Adding elements
- Excluding the generated comment
- Generating XML fragments
- Setting the format of the XML output
- Controlling the repetition of nested elements
- How repetition paths work
- Controlling the order of elements
- Examples
- Using XML Meta Data Importer
- Getting started with XML Output
- Creating table definitions with XML Meta Data Importer
- Adding an XML Output stage to your server job
- Setting up properties for the XML Output stage
- Stage properties
- Input link properties
- Output link properties

**Chapter 5. Using XML Transformer**
- About transforming XML documents
- Identifying XML sources
- Identifying XSLT stylesheets
- About National Language Support (NLS)
- Writing XML output
- Validating documents and schemas
- Setting XML schema options
- Mapping transformation errors to InfoSphere DataStage errors
- Using Reject links
- Passing data from input to output links
- Getting started with XML Transformer
- Adding an XML Transformer stage to your server job
- Setting up properties for the XML Transformer stage
- Stage properties
- Input Link properties
- Output Link properties

© Copyright IBM Corp. 2000, 2013
Chapter 1. Introduction

The XML Pack contains several components.

Extensible Markup Language (XML) is a major standard for representing structured data that flows between database systems and between business partners.

About XML Pack

Using the intuitive interfaces of XML Pack, you can quickly define conversion paths between XML documents and flat relational tables. Your XML sources and targets for data and transformation stylesheets include independent files, URLs, and table columns.

Components

XML Pack consists of active IBM® InfoSphere® DataStage® and QualityStage Designer stages and the standalone XML Meta Data Importer. The stages include:

- XML Input
- XML Output
- XML Transformer

There are two major steps in using XML Input and XML Output.

Step 1: Create mappings between XML and relational data

About this task

You create mappings only for XML Input and XML Output using the XML Meta Data Importer. The output is a table definition that contains a set of XML XPath expressions. For the XML Input stage, these XPath expressions specify how to extract information from the XML document to a relational database format.

For the XML Output stage, XPath expressions specify how to build the XML document from the relational data.

You can also manually create XPath expressions through the XML Input and XML Output stages.

Step 2: Add XML Pack stages to a server job

About this task

Drag-and-drop one or more XML Pack stages to a server diagram, and set up properties within each stage.
About XML Input
Within a server job, use XML Input to convert XML data to flat relational tables.

The major features of XML Input include:
- Data extraction through standard XPath expressions.
- XML schema support.
- Support for complex transformations using custom stylesheets.
- Support for multiple output links using different data sets.
- Robust error handling.
- Optional XML validation.

About XML Output
Use XML Output to convert tabular data, such as relational tables and sequential files, to XML hierarchical structures.

The major features of XML Output include:
- Generating XML output using a subset of XPath expressions.
- Generating multiple documents from the same input.
- XML namespace support.
- Configuring XML document generation, such as one document per input row.
- Generating output documents on disk or through an output link.
- Robust error handling.
- Optional XML validation.
About XML Transformer
Use XML Transformer to convert XML documents to another XML hierarchical format.

The major features of XML Transformer include:
- Generating multiple documents from the same input.
- XML schema validation.
- Robust error handling.
- Support for writing to disk or to an output link.
- Validation of XML input.
- Adherence to XSLT standard stylesheets.

Deriving and parsing XML hierarchies
The following stages use XPath expressions:
- XML Input
- XML Output

The following stages use the Apache Xalan XSLT processor to transform XML documents:
- XML Input
- XML Transformer

About the XML Meta Data Importer
Use the XML Meta Data Importer to create table definitions for XML Input and XML Output. Using the tree view, you quickly point-and-click the elements and attributes that you want derived as XPath expressions. You can process both XML documents and schemas.

The following diagram provides an overview of major features in the XML Meta Data Importer.
XML Schema Excerpt

<xs:element name = "orderperson" type = "xs:string" />
<xs:element name = "name" type = "xs:string" />
<xs:element name = "address" type = "xs:string" />
<xs:element name = "city" type = "xs:string" />
<xs:element name = "country" type = "xs:string" />
<xs:element name = "title" type = "xs:string" />
<xs:element name = "note" type = "xs:string" />

Schema Tree View

Table Definition

<table>
<thead>
<tr>
<th>Columnname</th>
<th>Key</th>
<th>SQL typ</th>
<th>Length</th>
<th>Scale</th>
<th>Null</th>
<th>Display</th>
<th>XPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantity</td>
<td></td>
<td>Decimal</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>/quantity</td>
</tr>
<tr>
<td>address</td>
<td></td>
<td>VarChar</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>/address</td>
</tr>
<tr>
<td>note</td>
<td></td>
<td>VarChar</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>/note</td>
</tr>
<tr>
<td>orderperson</td>
<td></td>
<td>VarChar</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>/orderperson</td>
</tr>
</tbody>
</table>
Chapter 2. Using the XML Meta Data Importer

Use the XML Meta Data Importer to create table definitions from XML sources. You can process both XML schemas and XML documents. Also, you can modify all XPath expressions that are created during an automated mapping from an XML source to a table definition.

For information about the effect of XPaths in the XML Pack, see Using XML input and Using XML output.

Converting XML data types

In creating table definitions, XML Meta Data Importer converts XML data to the equivalent SQL data type.

- For XML schemas, the explicit XML data types are converted to the closest SQL data types.
- For XML documents, in parsing the original string values, heuristics are used to find equivalent SQL data types. Current locale settings are used in evaluating date formats.

For a list of conversions from XML schema to SQL data types, see Advanced transformations.

Exploring the XML Meta Data Importer window

The XML Meta Data Importer window has the following panes:

- Tree View depicts the hierarchical structure in the XML source. This pane is the main view. It is always present and cannot be hidden or docked.
- Source contains the original XML schema or XML document, in read-only mode. To compare the tree view with the XML source, you can dock this pane next to the tree view.
- Node Properties describes XML and XPath information of the selected element.
- Table Definition maps elements that you select in the Tree View.
- Parser Output presents XML syntax and semantic errors.

Examining parser output

XML Meta Data Importer reports any syntax and semantic errors in the Parser Output pane when you open a source file. To find the error in the Source pane, double-click the error in the Parser Output pane.

After correcting the error outside of the XML Meta Data Importer, you can load the revised source file. To reload the file, select File > Refresh.
Getting started with XML Meta Data Importer

The following diagram outlines the steps in deriving table definitions from XML sources using the XML Meta Data Importer.

START

- Open an XML schema or XML document
- Select XML nodes for mapping to table definitions
- Optionally modify the table definitions
- Optionally modify the XPath expressions
- Save the table definition

Starting the XML Meta Data Importer

About this task

To start the XML Meta Data Importer:

Procedure

1. Start the IBM InfoSphere DataStage and QualityStage Designer.
2. Select Import > Table Definitions > XML Table Definitions.

Opening an XML source

About this task

You can process an XML schema file (.xsd) or an XML document (.xml). The file can be located on your file system or accessed with a URL.

About namespaces

This section describes how the XML Meta Data Importer handles namespaces and namespace prefixes in XML documents and XML schemas.
**Processing XML documents**

The XML Meta Data Importer retains namespaces and considers every node in an XML hierarchy to be fully qualified with a namespace prefix. The form is: Prefix:node-name. This approach applies to documents in which the prefixes are included or unspecified.

When prefixes are unspecified, XML Meta Data Importer generates prefixes using the pattern ns#, where # is a sequence number.

**Example**

The following input does not include a namespace prefix.

```xml
<Person xmlns="mynamespace">
  <firstName>John</firstName>
</Person>
```

The following code shows output data.

```xml
<ns1:Person xmlns:ns1="mynamespace">
  <ns1:firstName>John</firstName>
</ns1:Person>
```

**Processing XML schemas**

The XML Meta Data Importer processes namespaces in XML schemas according to three rules:

- General
- Import By Reference
- Target Namespace Unspecified

**General rule**

In general, the XML Meta Data Importer assigns the prefix defns to the target namespace. For example:

```xml
<xsd:schema targetNamespace="mynamespace" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Person">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="firstName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The `firstName` node generates the following XPath expression:

```
/defns:Person/defns:firstName
```

where defns=mynamespace

**Import by reference rule**

If the schema imports by reference other schemas with different target namespaces, the XML Meta Data assigns a prefix in the form ns# to each of them.

To enable this processing, the dependent schema must specify elementFormDefault="qualified". If this is omitted, the elements are considered as belonging to the caller's target namespace.

**Example**

The following example imports by reference the schema mysecondschema.

```xml
<xsd:schema targetNamespace "demonamespace" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:other="othernamespace">
```

Chapter 2. Using the XML Meta Data Importer 7
The schema mysecondschema contains the following statements:

```xml
<schema targetNamespace="othernamespace"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <complexType name="Address">
        <sequence>
            <element name="street" minOccurs="1" maxOccurs="1" />
            <element name="city" minOccurs="1" maxOccurs="1" />
            <element name="state" minOccurs="1" maxOccurs="1" />
            <element name="zip" minOccurs="1" maxOccurs="1" />
        </sequence>
    </complexType>
</schema>
```

The street node generates the following XPath expression:

```
/defns:Person/defns:address/ns2:street
```

where

defns=demonamespace
ns2=othernamespace

**The target namespace unspecified rule**

When the target namespace is unspecified, XML Meta Data Importer omits the prefix defns from XPath expressions.

**Example**

```xml
<schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <element name="Person">
        <complexType>
            <sequence>
                <element name="firstName" type="xsd:string" minOccurs="1" maxOccurs="1" />
            </sequence>
        </complexType>
    </element>
</schema>
```

The firstName tree node generates the following XPath expression:

```
/Person/firstName
```

**Mapping nodes from an XML schema**

You can individually select elements and attributes, or select all leaf nodes except empty ones in one step.

**Choosing individual items**

Select the box that is next to the element or TEXT node that you want to map. If you select an element, you get all the sub-nodes and the actual content of the element.
Your selection is reflected in the Table Definition pane. An asterisk appears after the title Table Definition when you modify the table definition. It disappears when you save the information.

Selecting all nodes
You can simplify selecting all leaf nodes by using the Auto-check command. This command checks leaf nodes.

XML Meta Data Importer ignores leaf nodes in the following circumstances:

• Nodes are empty.
• In a branch in which a node represents a reference to an element or a type defined elsewhere, such as an included schema. To avoid recursive looping, which may be deep in the subschema, the node is not expanded.
  You may manually expand the reference branch down to a specific level, and run the Auto-check command on the top branch node. This action selects all nodes in the branch.
• Node represents a detected recursion. This happens with a schema that has the following form:

```xml
parent = person >children>child = person
```
  You may manually expand the recursive branch and run the Auto-check command to select all nodes in the branch.

To run Auto-check:

Select File > Edit > Auto-check. The nodes appear in the Table Definition pane.

Modifying table definitions
About this task

You can modify each field for a mapped XML node. To make a change, type changes in the field. To commit the change for later saving, move the cursor into a different field or pane.

When you modify an XPath expression in the table definition, XML Meta Data Importer updates the property grid. The opposite action also holds true.

Modifying XPath expressions
About this task

You can modify XPath expressions manually or by using processing instructions.

Accessing the Edit XPath Expression dialog box
About this task

To access the Edit XPath Expression dialog box:

Procedure
1. Click inside the XPath field.
2. Click the ellipses next to the XPath expression.

Saving the table definition
About this task

To save a table definition:
Procedure

1. Select File > Save.
   Table definitions are associated with an InfoSphere DataStage project. XML Meta Data Importer prompts you, if needed, to connect to a project.
   The Save Table Definition dialog box opens.

2. Optionally, change the name of the table definition.
   InfoSphere DataStage requires that you follow this pattern: source\dsn\table_name
   For example: XML\Importer\order
   The default table definition name depends on the XML source name:

<table>
<thead>
<tr>
<th>Source file</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNC-name</td>
<td>Original file name without extension</td>
</tr>
<tr>
<td>URL</td>
<td>The value New</td>
</tr>
<tr>
<td>XML document</td>
<td>Original XML document filename</td>
</tr>
<tr>
<td>XML schema</td>
<td>Original XML schema filename</td>
</tr>
</tbody>
</table>

3. Optionally, supply short and long descriptions.

4. Optionally, change the current project.
   Your change affects only operations in the current XML Meta Data Importer session, not in future sessions or in the InfoSphere DataStage Designer.

5. Click OK.
Chapter 3. Using XML Input

You can transform hierarchical XML data to flat relational tables using the XML Input stage.

You can extract XML data from a variety of repositories, from documents that you edit with a text editor to a single column in a multi-column table.

XML Input supports a single input link and one or more output links.

About transforming XML documents

XML Input requires XPath expressions on output links to identify data in an XML document for extracting to output rows. XPath expressions are used to transform the input XML document into columns and rows. A table definition stores the XPath expressions for each XML document to be parsed.

Use the Description property on the Columns page to record or maintain the XPath expressions.

You have the following options for creating XPath expressions and table definitions:

• Use the XML Meta Data Importer, a visual tool for mapping XML elements and attributes. Start with this tool because it automatically generates the XPath expressions.
• Modify the generated XPath expressions to handle specific operating scenarios.
• Hand-code XPath expressions. This option is suited for advanced users.

Important: In order to process the document, the XML Input stage requires free memory space equivalent of five to seven times the size of the document. There is no direct connection between the memory and the size of the document. However, the memory requirement varies depending on the actual structure of the document, data within the document, and on the XPATH defined in the job. You must ensure that the size of the XML document is 100 MB or less in order to have a stable and serviceable set of jobs. Large files increase the risk of random job failures even when the memory usage is optimally configured.

For more information about using the XML Meta Data Importer, see Using the XML Meta Data Importer.

Identifying XML sources

An input column can contain an XML document, a URL, or a file path.

For more information about identifying the XML source, see Input link properties.

For information about the effect of engine tier host scodepages on accessing a file through a file path, see Local codepages on engine tier host.

About National Language Support (NLS)

This section applies only to running the InfoSphere DataStage Designer in NLS mode.

Preserving the encoding of input documents

An integral part of any XML document is its encoding. If you apply an InfoSphere DataStage map to the document, the document may become corrupted.

To prevent corrupting an XML document, perform one of the following steps:
• Set the stage map to NONE in each upstream stage.
• Set the map for the column that contains the XML input to NONE in each upstream stage.
• Set the SQL type for the column that contains the XML input to VarBinary in each upstream stage and on the input link of the XML Input stage.

Input link
The input link supports XML documents that are encoded using any Internet Assigned Numbers Authority (IANA) character sets. For a complete list of character sets, visit the following IANA web page:
http://www.iana.org/assignments/character-sets

Output link
When the InfoSphere DataStage Designer client runs in NLS mode, output columns are encoded in UTF-8.

When the InfoSphere DataStage Designer client runs in non-NLS mode, output columns are encoded using the local codepage of the system running the engine.

For information about the effect of the engine tier host codepages on accessing a file through a file path, see Local codepages on engine tier hosts.

Validating documents and schemas
XML Input performs two XML validations when the server job runs:
• Checks for well-formed XML.
• Optionally checks that elements and attributes conform to any XML schema that is referenced in the document. You control this option.

Validation against an XML schema is required when you want default values for elements and attributes that are specified in the schema written to an output row. To refer to a schema in your XML document, use the schemaLocation attribute within the root element tag.

For information about activating validation against a schema, see Stage properties.

Schema scan and schema validation
Schema scan and schema validation are performed during the XML Parsing process. The schema defined in the schemaLocation attribute of the XML is used for performing schema scan and validation.

The difference between schema scan and schema validation is:

Schema scan
Reads the default values for elements, attributes, and other objects that are specified in the XML schema and writes them to the output row. Schema scan is performed even if validation is disabled. Schema scan reads the default values from the schema that is referenced in the schemaLocation attribute of the XML document.

Schema validation
Validates the XML document against the XML schema, which is specified in the .xsd file. Schema validation ensures that the XML content conforms to the XML schema that is referenced in the schemaLocation attribute of the XML document.

Setting XML schema options
The XML parser compiles the validating schema to create a schema grammar. While validating the grammar, the parser can apply extra steps called full schema constraint checking, which may increase processing time and be memory-intensive. To enable this checking, use the Strict option. To disable this type of validation, use the Default option.
If your job processes two or more XML documents that use the same schema, you can avoid recompiling the schema by caching the grammar.

For information about activating schema validation, see Stage properties.

**Mapping transformation errors to InfoSphere DataStage errors**

The XML parser reports three types of conditions: fatal, error, and warning.

- Fatal errors are thrown when the XML is not well-formed.
- Non-fatal errors are thrown when the XML violates a validity constraint. For example, the root element in the document is not found in the validating XML schema.
- Warnings may be thrown when the schema has duplicate definitions.

For more information about these conditions, consult the XML and XML Schema specifications on the Worldwide Web Consortium web site [http://www.w3c.org](http://www.w3c.org).

By mapping parsing messages to InfoSphere DataStage error levels, you decide how parsing messages and faulty XML documents are processed.

The following table describes how each InfoSphere DataStage error level is processed.

<table>
<thead>
<tr>
<th>InfoSphere DataStage error level</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reject</td>
<td>Faulty document rows and messages can be written to a Reject link, if one exists. You can also send the messages to the job log. For more information about processing messages and documents, see Using reject links.</td>
</tr>
<tr>
<td>Fatal</td>
<td>The server job terminates, and the messages are written to the job log.</td>
</tr>
<tr>
<td>Warning</td>
<td>A warning message is written to the job log.</td>
</tr>
<tr>
<td>Info</td>
<td>An information message is written to the job log.</td>
</tr>
<tr>
<td>Trace</td>
<td>If the job runs with tracing set on, debug and monitoring information is written to the job log.</td>
</tr>
</tbody>
</table>

For more information about mapping errors and logging them, see Stage properties.

**Using Reject links**

XML Input supports one Reject link, which can store rejection messages and rejected rows.

**Writing rejection messages to the link**

**About this task**

To write rejection messages to a Reject link:

**Procedure**

1. Add a column on the Reject link.
2. Using the General page of the Output Link properties, identify the column as the target for rejection messages.
**Writing rejected rows to the link**  
**About this task**

To write rejected rows to a Reject link:

Add a column on the Reject link that has the same name as the column on the input link that contains or references the XML document.

This is a passthrough operation. Column names for this operation are case-sensitive. Passthrough is available for any input column.

For information about setting up a Reject link, see [Output link properties](#).

**Writing rejection messages to the job log**  
**About this task**

To write rejection messages to the job log:

On the General page of the [Stage properties](#), select the **Log Reject errors** box.

For more information about the General page, see [Stage properties](#).

---

**Passing data from input to output links**

Each output link supports a passthrough mechanism by which data is copied without modification from the input link to the output link. This mechanism works with output columns for which XPath expressions are not supplied.

Passthrough requires an exact match between column names specified in both the output link and the input link. Also, column names are case-sensitive.

**Note:** When using passthrough in a parallel job, the output link column must be a string if the corresponding input link column is a string. Passthrough from a string type to a Unicode string type is not supported.

**Controlling output rows**

To populate the columns of an output row, XML Input uses XPath expressions that are specified on the output link. XPath expressions locate elements, attributes, and text nodes.

**Controlling the number of output rows**

You must designate one column on the output link as the repetition element. A repetition element consists of an XPath expression. For each occurrence of the repetition element, XML Input always generates a row. By varying the repetition element and using a related option, you can control the number of output rows.

**Sample XML document**

Consider the following XML document, which has three address blocks. The final address block does not have a `<city>` element.

```xml
<?xml version="1.0"?>
<customers>
  <customer id="55000">
    <name>Charter Group</name>
    <address>
      <street>100 Main</street>
      <city>Framingham</city>
    </address>
  </customer>
</customers>
```
Sample XPath expressions

Consider the following output columns and their XPath expressions. You can generate these XPath expressions using the XML Meta Data Importer.

<table>
<thead>
<tr>
<th>Column name property</th>
<th>XPath expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>/customers/customer/@id</td>
</tr>
<tr>
<td>name</td>
<td>/customers/customer/name/text()</td>
</tr>
<tr>
<td>street</td>
<td>/customers/customer/address/street/text()</td>
</tr>
<tr>
<td>city</td>
<td>/customers/customer/address/city/text()</td>
</tr>
<tr>
<td>state</td>
<td>/customers/customer/address/state/text()</td>
</tr>
<tr>
<td>zip</td>
<td>/customers/customer/address/zip/text()</td>
</tr>
</tbody>
</table>

If you designate the XPath expression /customers/customer/name/text() as the repetition element, XML Input generates only one output row:

55000, "Charter Group", "100 Main", "Framingham", "MA", "01701"

This happens because the <name> element that the XPath expression selects occurs only one time in the XML document.

If you designate /customers/customer/address/city/text() as the repetition element, XML Input potentially generates three rows. Although there are only two occurrences of the city element found along the path that includes the ancestor elements /customers/customer/address, you can force a third row.

To force an output row when the final element in the XPath expression is not present, disable the option Repetition element required. The missing element is coded as NULL.

Here are three output rows generated from the sample XML document. The city column in the third row is coded as NULL.

55000, "Charter Group", "100 Main", "Framingham", "MA", "01701"
55000, "Charter Group", "720 Prospect", "Framingham", "MA", "01701"
55000, "Charter Group", "120 Ridge", NULL, "MA", "01760"

If the option Repetition element required is activated for this example, two rows are generated.

55000, "Charter Group", "100 Main", "Framingham", "MA", "01701"
55000, "Charter Group", "720 Prospect", "Framingham", "MA", "01701"
Identifying the repetition element

About this task

To identify the repetition element, set the **Key** property to Yes on the output link.

For more information about activating the option Repetition Element Required, see Transformation settings.

Processing NULLs and empty values

XML Input can replace NULLs with empty values, and replace empty values with NULLs. The target columns must use one of the string or binary SQL types.

In an input document, NULLs are missing elements or attributes that are referenced by XPath expressions. Empty values include empty elements (\(<a></a>\)) and empty attributes (\(\text{att}=\"\)\).

For examples using these replacements, see Null and empty value processing.

For information about specifying replacements, see Transformation settings.

Processing namespaces

XML Input requires namespace declarations when namespace prefixes are included in XPath expressions. If the input document uses namespaces, the XPath expression must be qualified. That is, a node that belongs to a namespace must have a prefix even if the namespace of the target node is the default namespace.

If you used the XML Meta Data Importer to create table definitions, you can extract namespace declarations from the generated table definition.

If you did not use the XML Meta Data Importer, you must:

- Write XPath expressions with namespace prefixes in the Description property of the Columns page of the Output Link Properties page.
- List the namespace declarations using the text box in the Transformation Settings page.

For more information about specifying namespace information, see Transformation settings.

Performing advanced transformations

To transform the XML document to columns and rows, XML Input uses an XSLT stylesheet that it generates from the XPath expressions that are specified on the output link.

If the XML document contains nodes whose relationships are not explicit, XML Input may not be able to automatically perform the transformation. As an alternative to the generated XSLT stylesheet, you can substitute your own stylesheet.

For more information about using a custom XSLT stylesheet, see Advanced transformations.

For information about specifying a custom stylesheet in the XML Input stage, see Advanced.
Getting started with XML Input
The following diagram outlines setting up an XML Input stage within a server job.

START

Optionally create table definitions with XML Meta Data Importer

Set up stage properties

Set up input link properties

Set up output link properties

Compile and run the server job

Creating table definitions with XML Meta Data Importer
For information about creating table definitions using XML Meta Data Importer, see Using the XML Meta Data Importer.

Adding an XML Input stage to your server job
About this task
Using the InfoSphere DataStage Designer Client, add an XML Input stage to a server job diagram.

Procedure
1. Select the Real Time category in the Palette pane.
2. From the Palette pane, drag the XML Input icon onto the canvas.
3. Click the Link icon, and connect two stages in your diagram.
4. Repeat Step 2 as needed.
Results

The following diagram shows a sample server job diagram, with renamed stage labels.

Folder Stage

Setting up properties for the XML Input stage

About this task

To set up properties, double-click the XML Input icon on your server job diagram.

Stage properties

The Stage properties are defined on two pages:

- General
- Transformation Settings

General

These properties control:

- Validating the input XML against the XML schema referenced in the XML document.
- Setting the schema validation level.
- Caching a schema grammar for another XML document that uses the same schema.
- Logging rejection errors for an input row into the job log.
• Mapping transformation error messages to InfoSphere DataStage error levels. These include fatal, error (non-fatal), and warning messages.

Transformation settings
These properties control the values that can be shared by multiple output links of the XML Input stage.

They fall into these categories:
• Requiring the repetition element
• Processing NULLs and empty values
• Processing namespaces
• Formatting extracted XML fragments

To use these values with a specific output link, select the Inherit Stage Properties box on the Transformation Settings tab of the output link.

For information about using repetition elements, see Controlling output rows.

For information about NULLs and empty values, see Processing NULLs and empty values.

Processing namespaces
If you used the XML Meta Data Importer to create table definitions, select the Include namespace declaration box and click the Load button to extract namespace declarations from a specific table definition.

If you did not use the XML Meta Data Importer, you must:
• Write XPath expressions with namespace prefixes in the Description property of the Columns page of the Output Link Properties page.
• Specify the namespace declarations in a space-delimited list using the text box.
  The syntax is: xmlns:<prefix>="<namespace_url>"

  Note: When you load a table definition from the Columns page, XML Input does not load namespace declarations.

Formatting extracted XML fragments
When an XPath expression ends with an element node, XML Input extracts and writes an XML fragment. You may want to generate XML fragments to:
• Preserve sections of the input on an output link.
• Split input that has multiple branches with implicit relationships into separate documents for subsequent processing. For an example that illustrates this relationship, see Sample input XML
document

Writing XML
XML Input can write the fragment on the output link as an unformatted or formatted block.

The following XPath ends with an element node:
/customers/customer/address

Unformatted output is the default for writing fragments. For example:
<address><street>1 Main</street><city>Fram</city><state>MA</state><zip>01701</zip></address>
Here is the same output, written as a formatted block:

<address>
  <street>1 Main</street>
  <city>Fram</city>
  <state>MA</state>
  <zip>01701</zip>
</address>

To generate a formatted fragment, select the **Format extracted XML fragments** box.

**Note:** To write tabular data, end XPath expressions with text or attribute nodes.

**Input Link properties**
The Input Link properties are defined on two pages:
- **XML Source**
- **Columns**

**XML source**
**About this task**
Use this page to specify the input column that contains the XML document, its URL, or file path.

For information about the effect of engine tier host codepages on accessing a file through a file path, see [Local codepages on engine tier hosts](#).

**Columns**
**About this task**
Use this standard InfoSphere DataStage grid to describe the input columns.

**Output Link properties**
The Output Link properties are defined on four pages:
- **General**
- **Transformation Settings**
- **Advanced**
- **Columns**

**General**
**About this task**
Use this page to set the following properties:
- Define an output link as a Reject link.
- Specify, for Reject links, the column that will contain rejection errors.

**Transformation settings**
**About this task**
This page controls transformation options for the output link.

To indicate that the output link inherits properties from the Stage page, select the **Input Stage properties** box.

**Note:** The Inherit Stage properties box is not available when the output link is a Reject link.

For information about:
- Repetition elements, see [Controlling output rows](#)
Advanced
About this task

Use this page to specify a custom XSLT stylesheet to transform an XML document.

You can specify a stylesheet in these ways:
• Identify an input column that contains the stylesheet, or its URL or file path.
• Type the stylesheet in the Stylesheet box.
• Type the URL or file path of the stylesheet in the Stylesheet box.
• Load the content or path of a stylesheet that is stored on the engine tier host.

For information about the effect of engine tier host codepages on accessing a file through a file path, see Local codepages on engine tier hosts.

To identify an input column as the source:
Procedure
1. Click the **Use custom stylesheet** box.
2. Click the **Get stylesheet from input column** box.
3. In the list box, select the input column.
4. Using the Source content buttons, identify the column contents. Valid choices are text (the stylesheet itself) and URL/File path.

To type a stylesheet:
Procedure
1. Click the **Use custom stylesheet** box.
2. Click the **Text** button.
3. In the Stylesheet box, type the stylesheet.
   Regarding the XML declaration:

To type a URL or file path:
Procedure
1. Click the **Use custom stylesheet** box.
2. Click the **URL/File path** button.
3. In the Stylesheet box, type the URL or file path.

Results

For information about the effect of engine tier host codepages on accessing a file through a file path, see Local codepages on engine tier host.

To specify a stylesheet on the engine tier host:
Procedure
1. Click the **URL/File path** button.
2. Click the **Load (Server)** button.
   The Browse File dialog box opens.
3. Locate the file and click **OK**.
   The file path appears in the Stylesheet box.
To specify a stylesheet on the InfoSphere DataStage client workstation:

**Procedure**

1. Click the **Text** button.
2. Click the **Load (Client)** button.
   The Open dialog box opens.
3. Locate the file and click **OK**.
   The stylesheet appears in the Stylesheet box.

To paste a stylesheet:

**Procedure**

1. Make sure that the encoding for the stylesheet matches the InfoSphere DataStage Designer client map.
2. Click the **Text** button.
3. Paste a copied stylesheet in the Stylesheet box.
4. Remove the XML declaration from the stylesheet header.

**Columns**

**About this task**

Use this standard InfoSphere DataStage grid to describe the output columns. Use the Description property to supply XPath expressions.

If a column name exactly matches an input column name and the output column’s Description property is empty, the output column is a passthrough column. XML Input copies the contents from the input column to the passthrough column.

To load XPath expressions from the table definition that the XML Meta Data Importer generates:

**Procedure**

1. Click the **Load** button.
   The Table Definitions dialog box opens.
2. Double-click the generated table definition.
   The Select Columns dialog box opens.
3. Click **OK** to select all the columns.
Chapter 4. Using XML Output

You can convert tabular data, such as relational tables and sequential files, to XML hierarchical structures, using the XML Output stage.

XML Output supports a single input link and zero or one output link.

About transforming tabular data

XML Output requires XPath expressions to transform tabular data to XML. A table definition stores the XPath expressions. Using the Description property on the Columns pages within the stage, you record or maintain the XPath expressions.

About National Language Support (NLS)

XML Output supports different character encodings for output documents, depending on the NLS mode.

InfoSphere DataStage in NLS mode

When InfoSphere DataStage runs in NLS mode, all Internet Assigned Numbers Authority (IANA) character sets are supported. For a complete list of character sets, visit the following IANA web page:

http://www.iana.org/assignments/character-sets

For information about selecting the output encoding, see Options page.

Preserving the encoding of output documents

An integral part of any XML document is its encoding. If you apply an InfoSphere DataStage map to the document, the document may become corrupted.

To prevent corrupting an XML document, perform one of the following steps:

- Set the stage map to NONE in each downstream stage.
- Set the map for the column that contains the XML input to NONE in each downstream stage.
- Set the SQL type for the column that contains the XML input to VarBinary in each downstream stage and on the output link of the XML Input stage.

InfoSphere DataStage in non-NLS mode

When InfoSphere DataStage runs in non-NLS mode, note the following information:

- The document is written in UTF-8.
- Input columns are encoded using the local codepage of the machine hosting the engine tier. Therefore, assume that input data to the XML Output stage has been encoded with this codepage. For information about the local codepage, see Local codepages on engine tier hosts.

Supported XPath expressions

The following Backus Naur Form (BNF) diagram describes the subset of XPath expressions that you can use in XML Output.

```
path ::= [/] (element_spec / )+ end_segment
end_segment ::= element_spec[/text()]["0" attribute
element_spec ::= element[ "attr_value ("and" attr_value )=""]
attr_value ::= "0" attribute "</"value"></"`
- Using the XML Meta Data Importer, generate the XPath expressions from an XML schema that you want the XML output to comply with.
- Modify the generated XPath expressions to handle specific operating scenarios.
- Hand-code XPath expressions. This option is suited for advanced users.

For more information about using the XML Meta Data Importer, see Using the XML meta data importer.

**Equivalent XPath expressions**
For an XML Output operation, two types of XPath expressions are equivalent. Both expressions result in the text node being included:
- An expression that ends with an element name: `/a/b`
- An expression that ends with a text node: `/a/b/text()`

**Using XPath expressions**
If a stage has both an input and an output link, XPath expressions are required on both links.

**XPath on input link**
On the input link, the XPath expressions drive the generation of XML. Each XPath expression maps the values of an input column to a node in an XML hierarchy.

The following example illustrates a possible mapping of values in the Customer column.

**XPath on output link**
Each output column that has an XPath expression is a candidate for receiving XML. The source of the XML for an output column are those input columns whose XPath expressions start with and contain the same nodes.

To make the entire XML available to an output column, use one forward slash as the XPath expression. The forward slash identifies the root node.

The following table demonstrates the relationship between XPath expressions on the input and output links. Two output columns use XPath expressions that form the first part of one of more XPath expressions used by input columns. For example, the output column that uses the XPath expression `/orders` receives XML generated using the XPath expressions `/orders/cust` and `/orders/items`. The column that uses the forward slash receives all the XML.

<table>
<thead>
<tr>
<th>Input column XPaths</th>
<th>Output column XPaths</th>
<th>/orders/cust</th>
<th>No</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>/addresses</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>/addresses/orders</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/orders/items</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Mapping related data to different root elements**
You can easily segregate related data in the XML by varying the root element. This feature is available when your XML Output stage has both input and output links. In a stage with only an input link, all XPath expressions must specify the same root element.
Example of XPath expressions
The input consists of addresses and orders for customers. The address data is grouped using the root element /addresses. The order data is grouped using the root element /orders.

<table>
<thead>
<tr>
<th>Input Column XPaths</th>
<th>Output Column XPaths</th>
</tr>
</thead>
<tbody>
<tr>
<td>/orders</td>
<td>/orders/items</td>
</tr>
<tr>
<td>/orders/custom</td>
<td>Yes</td>
</tr>
<tr>
<td>/addresses</td>
<td>No</td>
</tr>
<tr>
<td>/addresses/orders</td>
<td>No</td>
</tr>
<tr>
<td>/orders/items</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The ADDRESSES column receives the following XML structures:

```
<addresses>
  <address street=" " city=" ">
    ...
  </address>
</addresses>
```

The ORDERS column receives the following XML structures:

```
<orders>
  <order id=" ">
    <order item=" ">
      ...
    </order>
  </order>
</orders>
```

Parsing XML reserved and special characters
You can avoid parsing reserved and special XML characters that are already represented by entity references (&entity;) by setting the Data element property on the input link to XML.

If you use a different data element value or omit it, XML Output parses the input to make it XML-safe.

For example, the value &lt; replaces the less-than symbol (<).

Validating documents and schemas
XML Output provides an option to run two XML validation checks at run time:

- Checks for well-formed XML.
- Checks that elements and attributes conform to any XML schema that is referenced in the document.

If you decide to use the option, both validations are performed. Otherwise, no validation occurs.

To refer to a schema in your XML document, use the attribute schemaLocation within the root element tag.

For information about activating validation against a schema, see Stage properties.
Setting XML schema options
To enable validation, use the Strict option. To disable validation, use the Default option.

The XML parser compiles the validating schema to create a schema grammar. While validating the grammar, the parser can apply extra steps called full schema constraint checking, which may increase processing time and be memory-intensive.

If your job produces two or more XML documents that use the same schema, you can avoid recompiling the schema by caching the grammar.

For information about activating schema validation, see Stage properties.

Mapping validation errors to InfoSphere DataStage errors
The XML parser reports three types of conditions: fatal, error, and warning.

- Fatal errors are thrown when the XML is not well-formed.
- Non-fatal errors are thrown when the XML violates a validity constraint. For example, the root element in the document is not found in the validating XML schema.
- Warnings may be thrown when the schema has duplicate definitions.

For more information about these conditions, consult the XML and XML Schema specifications on the Worldwide Web Consortium web site.

By mapping parsing messages to InfoSphere DataStage error levels, you decide how parsing messages and faulty XML documents are processed.

The following table describes how each InfoSphere DataStage error level is processed.

<table>
<thead>
<tr>
<th>InfoSphere DataStage error level</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reject</td>
<td>Faulty document rows and messages can be written to a Reject link, if one exists. You can also send the messages to the job log. For more information about processing messages and documents, see Using Reject links.</td>
</tr>
<tr>
<td>Fatal</td>
<td>The server job terminates, and the messages are written to the job log.</td>
</tr>
<tr>
<td>Warning</td>
<td>A warning message is written to the job log.</td>
</tr>
<tr>
<td>Info</td>
<td>An information message is written to the job log.</td>
</tr>
<tr>
<td>Trace</td>
<td>If the job runs with tracing set on, debug and monitoring information is written to the job log.</td>
</tr>
</tbody>
</table>

For more information about mapping errors and logging them, see Stage properties.

Using Reject links
XML Output supports one Reject link, which can store rejection messages and rejected rows.

Writing rejection messages to the link
About this task
To write rejection messages to a Reject link:
**Procedure**
1. Add a column on the Reject link.
2. Using the General page of the Output Link properties, identify the column as the target for rejection messages.

**Writing rejected rows to the link**
**About this task**

To write rejected rows to a Reject link:

Add a column on the Reject link that has the same name as the column on the target column on the output link.

Column names for this operation are case-sensitive.

For information about setting up a Reject link, see [Output link properties](#).

**Writing rejection messages to the job log**
**About this task**

To write rejection messages to the job log:

On the Validation Settings page of the Stage properties, select the Log Reject Errors box.

For more information about the General page, see [Stage properties](#).

**Aggregating input rows on output**

You have several options for aggregating input rows on output.
- Aggregate all rows in a single output row. This is the default option.
- Generate one output row per input row. This is the Single row option.
- Trigger a new output row when the value of an input column changes.
- Trigger a new output row when the value of a passthrough column changes.
  
  A passthrough column is an output column that has no XPath expression in the Description property and whose name exactly matches the name of an input column.

**Example**

Four columns are involved in the transformation. ZONE values also pass through to an output column called ZONE.

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>DIVISION</th>
<th>CITY</th>
<th>ZONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>Boston</td>
<td>East</td>
</tr>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>New York</td>
<td>East</td>
</tr>
<tr>
<td>Acme</td>
<td>Chemical</td>
<td>Raleigh</td>
<td>East</td>
</tr>
<tr>
<td>Acme</td>
<td>Chemical</td>
<td>St. Louis</td>
<td>Midwest</td>
</tr>
</tbody>
</table>
**XPath expressions**

On the input link, the following XPath expressions are used. The CITY column has the repetition path.

<table>
<thead>
<tr>
<th>Input column</th>
<th>XPath expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER</td>
<td>/directory/customer/@name</td>
</tr>
<tr>
<td>DIVISION</td>
<td>/directory/customer/division/text()</td>
</tr>
<tr>
<td>CITY</td>
<td>/directory/customer/city/text()</td>
</tr>
<tr>
<td>ZONE</td>
<td>/directory/customer/@zone</td>
</tr>
</tbody>
</table>

Each example in **Generating output** uses the output column CUSTOMERS. The CUSTOMERS column uses a forward slash (/) as the XPath expression, which makes the entire XML available to it.

The passthrough example (see **Modes: Using passthrough and aggregate all rows**) adds the ZONE column, which omits an XPath expression. This makes the column available as a passthrough target.

**Generating output**

The section demonstrates the use of various output modes. For more information about setting the output mode, see [Transformation settings page](#).

**Mode: Aggregate all rows**

If you aggregate all input rows, XML Output creates one output row, which contains a single XML document.

```xml
<directory>
  <customer name="Acme" zone="East">
    <division>Toys</division>
    <city>Boston</city>
    <city>New York</city>
  </customer>
  <customer name="Acme" zone="East">
    <division>Chemical</division>
    <city>Raleigh</city>
  </customer>
  <customer name="Acme" zone="Midwest">
    <division>Chemical</division>
    <city>St. Louis</city>
  </customer>
</directory>
```

**Mode: Single row**

If you request one output row per input row, XML Output generates four output rows. Each output row contains the XML chunk generated from a different input row.

Row One

```xml
<directory>
  <customer name="Acme" zone="East">
    <division>Toys</division>
    <city>Boston</city>
  </customer>
</directory>
```

Row Two
Row Three
<directory>
  <customer name="Acme" zone="East">
    <division>Chemical</division>
    <city>Raleigh</city>
  </customer>
</directory>

Row Four
<directory>
  <customer name="Acme" zone="East">
    <division>Chemical</division>
    <city>St. Louis</city>
  </customer>
</directory>

Mode: Use trigger column

If you trigger a new output row based on a change in values in the DIVISION column, there will be two rows with different XML chunks: one for the Toys division and one for the Chemical division.

Row One
<directory>
  <customer name="Acme" zone="East">
    <division>Toys</division>
    <city>Boston</city>
    <city>New York</city>
  </customer>
</directory>

Row Two
<directory>
  <customer name="Acme" zone="East">
    <division>Chemical</division>
    <city>Raleigh</city>
  </customer>
  <customer name="Acme" zone="Midwest">
    <division>Chemical</division>
    <city>St. Louis</city>
  </customer>
</directory>

Modes: Passthrough and Aggregate all rows

The passthrough mechanism works in conjunction with other options. For example, if you use the Aggregate All Rows option, the column ZONE forces two output rows: one for the XML chunks generated from the first three input rows and one for the last input row.

The XML output is written to the CUSTOMERS column. The passthrough data is written to the ZONE column.
**Note:** When using passthrough in a parallel job, the output link column must be a string if the corresponding input link column is a string. Passthrough from a string type to a Unicode string type is not supported.

### Writing output to your file system

You can direct the XML output to files on your file system. With this option, file paths are written to the output link. The number of files depends partly on the number of output columns that contain XPath expressions and the number of input rows.

You select the root name of the output file, such as `acme.xml`. When there is a single output column that has an XPath expression, and you choose the Aggregate All Rows option, XML Output generates a single file, using the root name.

For information about the local codepage on the engine tier host when accessing a file through a file path, see [Local codepages on engine tier host](#).

### Generating output files for multiple columns

When two or more output columns have XPath expressions, XML Output generates a file for each column. You must add a column index flag to the root filename to prevent overwriting. This creates a naming pattern.

Valid flags include:

- `% %` Column position, starting with zero (0)
- `%@` Column names

You can add these flags before, within, or after the root filename.

### Examples

The first output column is CUSTOMERS. The second output column is DIVISIONS.

1. The naming pattern is `acme%%.xml`. XML Output generates two files, called `acme0.xml` and `acme1.xml`.
2. The naming pattern is `acme@.xml`. XML Output generates two files, called `acmeCUSTOMERS.xml` and `acmeDIVISIONS.xml`.

<table>
<thead>
<tr>
<th>CUSTOMERS</th>
<th>ZONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;directory&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;customer name=&quot;Acme&quot; zone=&quot;East&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;division&gt;Toys&lt;/division&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;city&gt;Boston &lt;/city&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;city&gt;New York&lt;/city&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/customer&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;customer name=&quot;Acme&quot; zone=&quot;East&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;division&gt;Chemical&lt;/division&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;city&gt;Raleigh&lt;/city&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/customer&gt;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CUSTOMERS</th>
<th>ZONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;directory&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;customer name=&quot;Acme&quot; zone=&quot;Midwest&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;division&gt;Chemical&lt;/division&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;city&gt;St. Louis&lt;/city&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/customer&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;directory&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;customer name=&quot;Acme&quot; zone=&quot;East&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;division&gt;Toys&lt;/division&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;city&gt;Boston &lt;/city&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;city&gt;New York&lt;/city&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/customer&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;customer name=&quot;Acme&quot; zone=&quot;Midwest&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;division&gt;Chemical&lt;/division&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;city&gt;Raleigh&lt;/city&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/customer&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Note: When using passthrough in a parallel job, the output link column must be a string if the corresponding input link column is a string. Passthrough from a string type to a Unicode string type is not supported.

### Writing output to your file system

You can direct the XML output to files on your file system. With this option, file paths are written to the output link. The number of files depends partly on the number of output columns that contain XPath expressions and the number of input rows.

You select the root name of the output file, such as `acme.xml`. When there is a single output column that has an XPath expression, and you choose the Aggregate All Rows option, XML Output generates a single file, using the root name.

For information about the local codepage on the engine tier host when accessing a file through a file path, see [Local codepages on engine tier host](#).

### Generating output files for multiple columns

When two or more output columns have XPath expressions, XML Output generates a file for each column. You must add a column index flag to the root filename to prevent overwriting. This creates a naming pattern.

Valid flags include:

- `% %` Column position, starting with zero (0)
- `%@` Column names

You can add these flags before, within, or after the root filename.

### Examples

The first output column is CUSTOMERS. The second output column is DIVISIONS.

1. The naming pattern is `acme%%.xml`. XML Output generates two files, called `acme0.xml` and `acme1.xml`.
2. The naming pattern is `acme@.xml`. XML Output generates two files, called `acmeCUSTOMERS.xml` and `acmeDIVISIONS.xml`.
Generating output files for rows

When there are two or more output rows, XML Output appends a row index to the generated filenames. The first index number, zero (0), is omitted from the file for the first row. The second index number, one (1), is appended as _1.

Use a column index flag when there is more than one output column with an XPath expression and you expect multiple output rows.

For information about setting the output filename, see Options page.

Examples using Single row mode

The output mode is Single row, which generates one output row per input row. The first output column is CUSTOMERS. The second output column is DIVISIONS. Both output columns use XPath expressions.

There are three input rows.

In row 1, the naming pattern is acme%%.xml. XML Output generates six files:
- acme0.xml
- acme0_1.xml
- acme0_2.xml
- acme1.xml
- acme1_1.xml
- acme1_2.xml

In row 2, the naming pattern is acme%@.xml. XML Output generates six files:
- acmeCUSTOMERS.xml
- acmeCUSTOMERS_1.xml
- acmeCUSTOMERS_2.xml
- acmeDIVISIONS.xml
- acmeDIVISIONS_1.xml
- acmeDIVISIONS_2.xml

In row 3, the root filename acme.xml is used. Because a column index flag is omitted, XML Output generates and overwrites three files. The final files contain XML chunks for the second column (DIVISIONS).
- acme.xml
- acme_1.xml
- acme_2.xml

To avoid this result, use column index flags when there are multiple columns and rows.

Example using passthrough columns

The output mode is Aggregate all rows, which generates one output row unless there is a passthrough column. ZONE is the passthrough column, and the output column CUSTOMER uses the forward slash (/) as the Description property. Here are the input rows.

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>DIVISION</th>
<th>CITY</th>
<th>ZONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>Boston</td>
<td>East</td>
</tr>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>New York</td>
<td>East</td>
</tr>
<tr>
<td>Acme</td>
<td>Chemical</td>
<td>Raleigh</td>
<td>East</td>
</tr>
</tbody>
</table>
The root filename acme.xml is used because only one output column is involved in the transformation. XML Output generates two output files, acme.xml and acme_1.xml, because the ZONE value changes in the fourth input row. The ZONE values are written on the output link but not to the output files.

File paths on the output link
File paths are written on the output link when output is directed to the file system.

For information about the effect of engine tier host codepages on accessing a file through a file path, see Local codepages on engine tier hosts.

Processing NULLs and empty values
XML Output can replace NULLs with empty values, and replace empty values with NULLs.

For a NULL column (SQL NULL), you have two options:
• Generate no element or attribute. This is the default behavior.
• Replace the NULL with an empty element or attribute.

For an empty column (empty string), you have two options:
• Generate an empty element or attribute. This is the default behavior.
• Replace the empty string with no element or attribute.

The final node in the XPath expression determines whether an element or attribute is involved.

For examples using these replacements, see NULL and empty value processing

For information about configuring replacements, see Transformation settings page

Selecting items for the XML Output
The default XML output includes the following items:
• XML declaration
• Comment, which identifies the generation date and XML Output as the generator
• XML hierarchy

Adding elements
Using the Document Settings options, you can add the following items to the generated output:
• Document type declaration
• Header elements
• Namespace declaration
• Nested XML chunk

The complete list is arranged in the following order:
```xml
<?xml version='1.0'>
<!--[-- comment -->]
<document_type_declaration>
<header_elements>
<root namespace_declaration>
   <nested_XML_chunk>
</root>
</header_elements>
</document_type_declaration>

About document type declarations and nested chunks
A valid XML document conforms to a DTD or XML schema. Using the document type declaration, you can specify a DTD. When using a nested chunk, which can contain any well-formed XML fragment, you can specify an inline schema.

About headers
Headers include comments and processing instructions, such as xml-stylesheet, that are inserted between the XML declaration and the root element. For example:
```xml
<?xml version='1.0'>
<?xml-stylesheet type="text/xsl" href="my.xsl"?>
<root>
```

About namespace declarations
In this context, the namespace declarations that you specify are inserted within the root element tag.

For more information about including these optional items in the final XML, see Document settings page.

Excluding the generated comment
You can exclude generated comment from the output. For more information, see Document settings page.

Generating XML fragments
You can generate an XML fragment, which excludes the XML declaration, the generated comment, headers, and document type declarations. This is useful when your generated XML needs to be combined with other documents.

For information about generating an XML fragment, see Document settings page.

Setting the format of the XML output
You can control the format of the XML output in the following ways:
• Empty element style: single opening/closing tag or separate opening and closing tags.
• Indenting output: spaces or a tab character.
• New line style: operating system defaults, such as line feed (UNIX style) and carriage return/line feed (DOS style).

For information about setting the empty element style, see Transformation Settings Page.

For information about indenting output and setting the new line style, see Options page.

Controlling the repetition of nested elements
In a given XML hierarchy, nested elements may repeat. The following example contains a list of cities in which a customer operates.
On the input link, city values map to the hierarchy using the following XPath expression:

```
/directory/customer/city/text()
```

To enable the `<city>` element to repeat within the `<customer>` node, you need to define this XPath expression as the repetition path. To do this, identify the associated input column as the key.

Repetition applies to each node beneath the root element. Therefore, the `<customer>` element can also repeat. The final output typically depends on the repetition path and the order of the input rows. Two repetition paths can produce the same results.

**How repetition paths work**

The repetition path works by comparing values between input rows. The following rules apply:

**Rule 1**

A change in an input column value triggers the closing of at least one element and the opening of at least one element.

**Rule 2**

When a single input column value changes, the repetition path applies as follows:

Every opened element is closed up to and including the first element that is part of the repetition path.

Repetition path: `/w/x/y/z`

XPath of the affected column: `/w/x/y/a/b`

The `y` element is closed.

New elements are opened, down to the last element of the XPath expression of the column for which the value has changed.

**Rule 3**

When more than one column changes values, elements are closed and opened, starting with the element that is closest to the root element.

**Examples**

The input data consists of four rows, ordered by division:

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>DIVISION</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>Boston</td>
</tr>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>New York</td>
</tr>
<tr>
<td>Acme</td>
<td>Chemical</td>
<td>Boston</td>
</tr>
<tr>
<td>Acme</td>
<td>Chemical</td>
<td>New York</td>
</tr>
</tbody>
</table>

The XPath expressions for each column are as follows:
<table>
<thead>
<tr>
<th>Column</th>
<th>XPath expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER</td>
<td>/directory/customer/@name</td>
</tr>
<tr>
<td>DIVISION</td>
<td>/directory/customer/division/text()</td>
</tr>
<tr>
<td>CITY</td>
<td>/directory/customer/city/text()</td>
</tr>
</tbody>
</table>

**Making /directory/customer/@name the repetition path**

When /directory/customer/@name is the repetition path, Rule 2 applies.

The first input row generates the following fragment:

```xml
<directory>
  <customer name="Acme">
    <division>Toys</division>
    <city>Boston</city>
  </customer>
</directory>
```

In the second row, the CITY value changes. Based on Rule 2, the <customer> element must close, and a new <customer> element must open.

```xml
</customer>
<customer name="Acme">
  <division>Toys</division>
  <city>New York</city>
</customer>
```

In subsequent rows, the CITY or DIVISION value changes. The full XML output is as follows:

```xml
<directory>
  <customer name="Acme">
    <division>Toys</division>
    <city>Boston</city>
    <city>New York</city>
  </customer>
  <customer name="Acme">
    <division>Toys</division>
    <city>New York</city>
  </customer>
  <customer name="Acme">
    <division>Chemical</division>
    <city>Boston</city>
    <city>New York</city>
  </customer>
  <customer name="Acme">
    <division>Chemical</division>
    <city>New York</city>
  </customer>
</directory>
```

**Making /directory/customer/city/text() the repetition path**

When /directory/customer/city/text() is the repetition path, Rules 2 and 3 apply.

Rule 2 forces a new <city> element that contains the New York value to be written immediately after the <city> element that contains the Boston element. This happens because the changed value corresponds to the last element in the repetition path.

In row 3, both the DIVISION and CITY values change. This invokes Rule 3, which closes the <customer> element, which is common among the XPath expressions for the affected columns.

The full XML output is as follows:

```xml
<directory>
  <customer name="Acme">
    <division>Toys</division>
    <city>Boston</city>
    <city>New York</city>
  </customer>
</directory>
```
Making /directory/customer/division/text() the repetition path

When /directory/customer/division/text() is the repetition path, Rules 2 and 3 apply.

Because the city value changes between rows, the <division> element does not repeat within a customer node. The result is the same output as what the repetition path /directory/customer/@name yields.

If you order the input rows by city, the division element repeats in the customer node.

Here is the reordered input:

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>DIVISION</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>Boston</td>
</tr>
<tr>
<td>Acme</td>
<td>Chemical</td>
<td>Boston</td>
</tr>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>New York</td>
</tr>
<tr>
<td>Acme</td>
<td>Chemical</td>
<td>New York</td>
</tr>
</tbody>
</table>

Here is the final output:

```xml
<directory>
  <customer id="Acme">
    <division>Toys</division>
    <division>Chemical</division>
    <city>Boston</city>
  </customer>
  <customer id="Acme">
    <division>Toys</division>
    <division>Chemical</division>
    <city>New York</city>
  </customer>
</directory>
```
### Controlling the order of elements

If you want the XML document to conform to a schema that stipulates the sequence of elements (<xs:sequence>), you must order the XPath expressions on the input link accordingly. XML Output sets the position of an element based on its first occurrence in the set of XPath expressions.

### Examples

The repetition path /directory/customer/city/text() is in effect, and the order of XPath expressions is as follows. Notice that the first occurrence of the node division precedes the first occurrence of the node city.

<table>
<thead>
<tr>
<th>Column</th>
<th>XPath expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER</td>
<td>/directory/customer/@name</td>
</tr>
<tr>
<td>DIVISION</td>
<td>/directory/customer/division/text()</td>
</tr>
<tr>
<td>CITY</td>
<td>/directory/customer/city/text()</td>
</tr>
</tbody>
</table>

The input data consists of four rows:

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>DIVISION</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>Boston</td>
</tr>
<tr>
<td>Acme</td>
<td>Toys</td>
<td>New York</td>
</tr>
<tr>
<td>Acme</td>
<td>Chemical</td>
<td>Boston</td>
</tr>
<tr>
<td>Acme</td>
<td>Chemical</td>
<td>New York</td>
</tr>
</tbody>
</table>

Because the node division precedes city, the output is as follows:

```xml
<directory>
  <customer name="Acme">
    <division>Toys</division>
    <city>Boston</city>
    <city>New York</city>
  </customer>
  <customer name="Acme">
    <division>Chemical</division>
    <city>Boston</city>
    <city>New York</city>
  </customer>
</directory>
```

If you reorder the XPath expressions, the result changes. For example, the XPath expression for CITY is now in the middle position:

<table>
<thead>
<tr>
<th>Column</th>
<th>XPath expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER</td>
<td>/directory/customer/city/text()</td>
</tr>
<tr>
<td>CITY</td>
<td>/directory/customer/city/text()</td>
</tr>
<tr>
<td>DIVISION</td>
<td>/directory/customer/division/text()</td>
</tr>
</tbody>
</table>

This reordering generates the following output:

```xml
<directory>
  <customer name="Acme">
    <city>Boston</city>
    <city>New York</city>
  </customer>
</directory>
```
Using XML Meta Data Importer

About this task

To ensure that the XML document conforms to a schema:

Procedure

1. Use the XML Meta Data Importer to generate XPath expressions from the schema.
2. Load the table definition that contains the XPath expressions. For more information about loading the table definition, see [Input link properties](#).

Getting started with XML Output

The following diagram outlines setting up an XML Output stage within a server job.

```
START
   Optionally create table definitions from an XML schema with XML Meta Data Importer
   Set up stage properties
   Set up input link properties
   Optionally, set up output link properties
   Compile and run the server job
```
Creating table definitions with XML Meta Data Importer

For information about creating table definitions using XML Meta Data Importer, see Using the XML Meta Data Importer.

Adding an XML Output stage to your server job

About this task

Using the InfoSphere DataStage Designer Client, add an XML Output stage to a server job diagram.

Procedure

1. Select the Real Time category in the Palette pane.
2. From the Palette pane, drag the XML Output icon onto the canvas.
3. Click the Link icon, and connect two stages in your diagram.
4. Repeat Step 2 as needed.

Results

The following diagram shows a completed server job diagram, with renamed stage labels.

Setting up properties for the XML Output stage

About this task

To set up properties, double-click the XML Output icon on your server job diagram.

Stage properties

The Stage properties are defined on four pages:

- Document Settings
- Transformation Settings
- Validation Settings
- Options

Note: When there is one output link, use the Output link page to modify the properties for Document Settings, Transformation Settings, and Options. Use the Stage page to modify properties in Validation Settings.
Document settings page

About this task

This page includes up to five tabs, which control the inclusion of following items in the output:

- Comment
- Header
- Namespace declaration
- Document type declaration
- Nested chunk

Generating XML fragments

To indicate that the generated XML includes only XML fragments, select the Generate XML Chunk box.

The number of tabs depends on whether the Generate XML Chunk box is selected. If not selected, all five Document Settings tabs are present. If selected, the tabs include only Comment, Namespace Declaration and Nested Chunk.

Comment Tab

Use this tab to toggle the inclusion of the following XML comment in the generated XML. The comment is inserted after the XML declaration.

<!--
- Generated by IBM Corporation, InfoSphere DataStage - XMLOutput stage -
- Date
-->

Header tab:

About this task

Use this tab to insert processing instructions, comments, and other information between the XML declaration and the root element in the generated XML.

You can access a header from the InfoSphere DataStage client or from the engine tier. In addition, you can type tags in the text box.

To specify a header on the engine tier host::

Procedure
1. Click the Include box.
2. Click the File Path button.
3. Click the Load (server) button.
   The Browse File dialog box opens.
4. Locate the file and click OK.
   The file path appears in the text box.

To specify a header on the InfoSphere DataStage client workstation::

Procedure
1. Click the Include box.
2. Click the Text button.
3. Click the Load (client) button.
   The Open dialog box opens.
4. Locate the file and click OK.
The header tags appear in the text box.

**Namespace Declaration tab:**

**About this task**

Use this tab to insert a space-delimited list of namespace declarations within the generated root element tag.

The syntax is: `xmlns:<prefix>="<namespace_url>`

You can load namespace declarations from only table definitions that XML Meta Data Importer generates. If you did not use XML Meta Data Importer, type the declarations in the text box.

To load namespace declarations:

**Procedure**

1. Click the **Include** box.
2. Click the **Load** button.
   The Table Definitions dialog box opens.
3. Locate the table definition containing the namespace declarations and click **OK**.
   The namespace declarations appear in the text box.

**Document type tab:**

**About this task**

Use this tab to insert a document type declaration in the generated XML.

You can access a document type declaration from the InfoSphere DataStage client or engine tier host. In addition, you can type the declaration in the text box.

*To specify a document type declaration on the engine tier workstation:*

**Procedure**

1. Click the **Include** box.
2. Click the **File Path** button.
3. Click the **Load (server)** button.
   The Browse File dialog box opens.
4. Locate the file and click **OK**.
   The file path appears in the text box.

*To specify a document type declaration on the InfoSphere DataStage client workstation:*

**Procedure**

1. Click the **Include** box.
2. Click the **Text** button.
3. Click the **Load (client)** button.
   The Open dialog box opens.
4. Locate the file and click **OK**.
   The document type declaration appears in the text box.

**Nested Chunk tab:**

**About this task**

Use this tab to insert a nested chunk in the generated XML.
You can access a nested chunk from the InfoSphere DataStage client or engine tier host, or from a web site. In addition, you can type the nested chunk in the text box.

To specify a nested chunk on the engine tier host:

**Procedure**
1. Click the **Include** box.
2. Click the **File Path** button.
3. Click the **Load (server)** button.
   - The Browse File dialog box opens.
4. Locate the file and click **OK**.
   - The file path appears in the text box.

To specify a nested chunk on the InfoSphere DataStage client workstation:

**Procedure**
1. Click the **Include** box.
2. Click the **Text** button.
3. Click the **Load (client)** button.
   - The Open dialog box opens.
4. Locate the file and click **OK**.
   - The nested chunk appears in the text box.

**Transformation settings page**
Use this page to:
- Replace NULLs with empty values.
- Replace empty values with NULLs.
  - For examples using these replacements, see [NULL and empty value processing](#).
- Select an empty tag style:
  - `<tag></tag>`
  - `<tag/>`
- Select a mode for generating output rows:
  - Generate one XML document for all rows (**Aggregate all rows** mode).
  - Generate one XML document per input row (**Single row** mode).
  - Trigger a new output row when the value of an input column changes (**Use trigger column** mode).

**Validation settings**
These properties control:
- Validating the output XML against the XML schema referenced in the XML document.
- Setting the schema validation level.
- Caching a schema grammar for another XML document that uses the same schema.
- Logging rejection errors for an output row into the job log.
- Mapping transformation error messages to InfoSphere DataStage error levels. These include fatal, error (non-fatal), and warning messages.

**Options page**
Use this page to control processing of the following items:
- Output file. Determines whether an output file is written, and provides the file path.
  - For more information about output files, see [Writing output to your file system](#).
- Indentation character. Character to use for indenting nested elements.
- New line style. Choose a new line character from among operating system defaults.
Output encoding for NLS installations. For non-NLS installations, UTF-8 is used. NLS installation, select the encoding from the XML output encoding list. The default encoding is UTF-8.

Input link properties

About this task

Use the standard InfoSphere DataStage grid on the Columns page to describe the data that is extracted from your tabular source. Use the Description property to specify XPath expressions for the XML transformation.

To load XPath expressions from the table definition that the XML Meta Data Importer generates:

Procedure
1. Click the Load button.
   The Table Definitions dialog box opens.
2. Double-click the generated table definition.
   The Select Columns dialog box opens.
3. Click OK to select all the columns.

Output link properties

The Output link properties are defined on five pages:
- General
- Document Settings: for more information, see [Document settings page](#)
- Transformation Settings: for more information, see [Transformation settings page](#)
- Options: for more information, see [Options page](#)
- Columns

Columns page

Use the standard InfoSphere DataStage grid to describe the output columns. Use the Description property to specify XPath expressions that populate columns with generated XML output.

General

Use this page to set the following properties:
- Define an output link as a Reject link.
- Specify, for Reject links, the column that will contain rejection errors.
Chapter 5. Using XML Transformer

You can convert an XML document to another XML hierarchical format using the XML Transformer stage.

XML Transformer supports one input link and zero or more output links.

About transforming XML documents
XML Transformer converts documents using an XSLT stylesheet that you specify.

Identifying XML sources
An input column can contain an XML document, or its URL or file path.

For more information about identifying the XML source, see Input link properties.

For information about the effect of engine tier host codepages on accessing a file through a file path, see Local codepages on engine tier hosts.

Identifying XSLT stylesheets
You specify the XSLT stylesheet by its file path or URL.

For more information about identifying the XSLT stylesheet, see Transformation settings.

About National Language Support (NLS)
You can use character encodings for the input and output links.

Input link
The input link supports XML documents that are encoded using any Internet Assigned Numbers Authority (IANA) character sets. For a complete list of character sets, visit the following IANA web page:
http://www.iana.org/assignments/character-sets

Output link
XML Transformer supports all Internet Assigned Numbers Authority (IANA) character sets. For a complete list of character sets, visit the following IANA web page:
http://www.iana.org/assignments/character-sets

In your XSLT stylesheet, specify the encoding of the output document using the <xsl:output> element.
For example:
<xsl:output encoding="ISO-10646-J-1"/>

InfoSphere DataStage in NLS mode
This section applies only to running InfoSphere DataStage in NLS mode.

Preserving the encoding of input documents
The engine can corrupt well-formed XML documents through transcoding the contents. Transcoding involves applying an encoding scheme that may be incompatible with the one that is listed in the XML declaration.

To prevent transcoding, perform one of the following steps:
• Set the stage map to NONE in each upstream stage.
• Set the map for the column that contains the XML input to NONE in each upstream stage.
• Set the SQL type for the column that contains the XML input to VarBinary in each upstream stage and on the input link of the XML Transformer stage.

Using XSLT stylesheets

If you want to paste a stylesheet through the Transformation Settings tab of the Stage page, follow these guidelines:
• Make sure that the encoding for the stylesheet matches the InfoSphere DataStage client map.
• Do not specify the XML declaration in the stylesheet header.

Preserving output encoding

To preserve the output encoding when the document is written to the output link, you must set the SQL type of the target column to a binary type. If you use a character type, the document is written in UTF-8.

Writing XML output

If the XML Transformer stage has only an input link, the XML output is written only to the file system.

Naming output files

The output filename that you specify as a Stage property works as a root name. If the input consists of multiple rows, XML Transformer creates a file for each input row. XML Transformer appends a row index to the filename. The first index number, zero (0), is omitted from the file for the first row. The second index number, one (1), is appended as _1.

Examples

Three output files are generated using the root name myfile.xml:
• First row file is myfile.xml
• Second row file is myfile_1.xml
• Third row file is myfile_2.xml

For information about setting the filename as a Stage property, see Transformation settings.

Using output links

If the XML Transformer stage has an output link, you can select one of these output choices:
• Write the generated XML to output files, and send the file paths to an output column.
• Send the generated XML to an output column.

For more information about directing output as an Output property, see Output settings.

Validating documents and schemas

XML Transformer performs two XML validations when the server job runs:
• Checks for well-formed XML.
• Optionally checks that elements and attributes conform to any XML schema that is referenced in the document. You control this option.

Validation against an XML schema is required when you want default values for elements and attributes that are specified in the schema to be written to an output row. To refer to a schema in your XML document, use the schemaLocation attribute within the root element tag.

For information about activating validation against a schema, see Stage properties.
**Setting XML schema options**

The XML parser compiles the validating schema to create a schema grammar. While validating the grammar, the parser can apply extra steps called full schema constraint checking, which may increase processing time and be memory-intensive. To enable this checking, use the Strict option. To disable this type of validation, use the Default option.

If your job processes two or more XML documents that use the same schema, you can avoid recompiling the schema by caching the grammar.

For information about activating schema validation, see [Stage properties](#).

**Mapping transformation errors to InfoSphere DataStage errors**

The XML parser reports three types of conditions: fatal, error, and warning.

- Fatal errors are thrown when the XML is not well-formed.
- Non-fatal errors are thrown when the XML violates a validity constraint. For example, the root element in the document is not found in the validating XML schema.
- Warnings may be thrown when the schema has duplicate definitions.

For more information about these conditions, consult the XML and XML Schema specifications on the Worldwide Web Consortium web site.

By mapping parsing messages to InfoSphere DataStage error levels, you decide how parsing messages and faulty XML documents are processed.

The following table describes how each InfoSphere DataStage error level is processed.

<table>
<thead>
<tr>
<th>InfoSphere DataStage error level</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reject</td>
<td>Faulty document rows and messages can be written to a Reject link, if one exists. You can also send the messages to the job log. For more information about processing messages and documents, see <a href="#">Using reject links</a>.</td>
</tr>
<tr>
<td>Fatal</td>
<td>The server job terminates, and the messages are written to the job log.</td>
</tr>
<tr>
<td>Warning</td>
<td>A warning message is written to the job log.</td>
</tr>
<tr>
<td>Info</td>
<td>An information message is written to the job log.</td>
</tr>
<tr>
<td>Trace</td>
<td>If the job runs with tracing set on, debug and monitoring information is written to the job log.</td>
</tr>
</tbody>
</table>

For more information about mapping errors and logging them, see [Stage properties](#).

**Using Reject links**

XML Transformer supports one Reject link, which can store rejection messages and rejected rows.

**Writing rejection messages to the link**

**About this task**

To write rejection messages to a Reject link:
**Procedure**

1. Add a column on the Reject link.
2. Using the General page of the Output Link properties, identify the column as the target for rejection messages.

**Writing rejected rows to the link**

*About this task*

To write rejected rows to a Reject link, add a column on the **Reject** link that has the same name as the column on the input link that contains or references the XML document. This is a passthrough operation. Column names for this operation are case-sensitive.

Passthrough is available for any input column.

For information about setting up a Reject link, see [Output link properties](#).

**Writing rejection messages to the job log**

*About this task*

To write rejection messages to the job log:

On the General page of the Stage properties, select the Log Reject errors box.

For more information about the General page, see [Stage properties](#).

**Passing data from input to output links**

Each output link supports a passthrough mechanism by which data is copied without modification from the input link to the output link. This mechanism works with an output column that is not the target for the transformation.

Passthrough requires an exact match between column names specified in both the output link and the input link. Also, column names are case-sensitive.

**Note:** When using passthrough in a parallel job, the output link column must be a string if the corresponding input link column is a string. Passthrough from a string type to a Unicode string type is not supported.
Getting started with XML Transformer

The following diagram outlines setting up an XML Transformer stage within a server job.

START

Create an XSLT stylesheet

Set up stage properties

Set up input link properties

Optionally, set up output link properties

Compile and run the server job

Adding an XML Transformer stage to your server job

About this task

Using the InfoSphere DataStage Designer Client, add an XML Transformer stage to a server job diagram.

Procedure

1. Select the Real Time category in the Palette pane.
2. From the Palette pane, drag the XML Transformer icon onto the canvas.
3. Click the Link icon, and connect two stages in your diagram.
4. Repeat Step 2 as needed.
Results

The following diagram shows a sample server job diagram, with renamed stage labels.

---

Setting up properties for the XML Transformer stage

About this task

To set up properties, double-click the XML Transformer icon on your server job diagram.

Stage properties

The Stage properties are defined on two pages:

- General
- Transformation Settings

General

These properties control:

- Validating the input XML against the XML schema referenced in the XML document.
- Setting the schema validation level.
- Caching a schema grammar for another XML document that uses the same schema.
- Logging rejection errors for an input row into the job log.
• Mapping transformation error messages to InfoSphere DataStage error levels. These include fatal, error (non-fatal), and warning messages.

**Transformation settings**

**About this task**

These properties control the default values that can be shared by multiple output links.

Use the Transformation Settings page to specify:

• Output path for the target XML document. The filename is used as naming pattern when XML Transformer generates multiple output files.
• XSLT stylesheet used in the transformation.

To use these values with a specific output link, click the Inherit properties from stage box on the General page of the output link.

**Specifying the stylesheet**

You can specify a stylesheet in these ways:

• Load the content or path of a stylesheet that is stored on an engine tier host.
• Type the stylesheet.
• Paste the stylesheet.
• Type the URL or file path of the stylesheet.

For information about the effect of engine tier host codepages on accessing a file through a file path, see [Local codepages on engine tier host](#).

**To specify a stylesheet on the engine tier host workstation:**

**Procedure**

1. Click the **URL/File Path** button.
2. Click the **Load (server)** button.
   The Browse File dialog box opens.
3. Locate the file and click **OK**.
   The file path appears in the **Stylesheet** box.

**To specify a stylesheet on the InfoSphere DataStage client workstation:**

**Procedure**

1. Click the **Text** button.
2. Click the **Load (client)** button.
   The Open dialog box opens.
3. Locate the file and click **OK**.
   The stylesheet appears in the **Stylesheet** box.

**To paste a stylesheet:**

**Procedure**

1. Make sure that the encoding for the stylesheet matches the InfoSphere DataStage client map.
2. Click the **Text** button.
3. Paste a copied stylesheet in the **Stylesheet** box.
4. Remove the XML declaration from the stylesheet header.

**To type a URL or file path:**
Procedure
1. Click the URL/File path button.
2. Type the URL or file path in the text box.

**Input Link properties**
The Input Link properties are defined on two pages:
- XML Source
- Columns

**XML source**
Use this page to specify the input column that contains the XML document, its URL, or file path.

**Columns**
Use this standard InfoSphere DataStage grid to describe the input columns.

**Output Link properties**
The Output Link properties are defined on four pages:
- General
- Stylesheet Settings
- Output Settings
- Columns

**General**
Use this page to set the following properties:
- Define an output link as a Reject link.
- For Reject links, specify the column that will contain rejection messages.
- Indicate that the output link inherits properties from the Stage page.

*Note:* The Inherit properties from stage box is not available when the output link is a Reject link.

**Stylesheet settings**
Use this page to identify an XSLT stylesheet (text or URL) that you will use to convert the XML source.
For information about identifying the stylesheet, see [Specifying the stylesheet](#).

*Note:* All controls are not available if the properties are inherited from the Stage page.

**Output settings**
**About this task**

Use this page to direct the XML output to a file or an output column. If you direct the XML output to a file, the output column receives a file path.

To direct the XML output to an output column::
**Procedure**
1. In the Target column list, select the output column.
2. Click the Text button.

To direct the XML output to a file::
**Procedure**
1. In the Target column list, select the output column that receives the file path.
2. Click the URL/File path button.
3. Type the file path in the Output file path box.
Note: The Output file path box is not available when the output link inherits properties from the Stage page.

Columns
Use this standard InfoSphere DataStage grid to describe the output columns, including the one to contain the XML text, URL, or file path.

If a column name exactly matches an input column name and the output column is not the target for the transformation, the output column is a passthrough column.

XML Transformer copies the contents from the input column as is to the passthrough column.
Chapter 6. XML to SQL data type conversions

The following table indicates how XML data types are converted to SQL data types.

<table>
<thead>
<tr>
<th>XML data type</th>
<th>SQL data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyURL</td>
<td>VarChar</td>
</tr>
<tr>
<td>base64Binary</td>
<td>LongVarChar</td>
</tr>
<tr>
<td>Boolean</td>
<td>VarChar</td>
</tr>
<tr>
<td>byte</td>
<td>SmallInt</td>
</tr>
<tr>
<td>date</td>
<td>Timestamp</td>
</tr>
<tr>
<td>decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>default</td>
<td>Unknown</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>LongVarChar</td>
</tr>
<tr>
<td>ENTITY</td>
<td>VarChar</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>gDay</td>
<td>Timestamp</td>
</tr>
<tr>
<td>gMonthDay</td>
<td>Timestamp</td>
</tr>
<tr>
<td>gYear</td>
<td>Timestamp</td>
</tr>
<tr>
<td>gYearMonth</td>
<td>Timestamp</td>
</tr>
<tr>
<td>hexBinary</td>
<td>LongVarChar</td>
</tr>
<tr>
<td>ID</td>
<td>VarChar</td>
</tr>
<tr>
<td>IDREF</td>
<td>VarChar</td>
</tr>
<tr>
<td>IDREFS</td>
<td>LongVarChar</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>integer</td>
<td>Decimal</td>
</tr>
<tr>
<td>language</td>
<td>VarChar</td>
</tr>
<tr>
<td>long</td>
<td>Numeric</td>
</tr>
<tr>
<td>Name</td>
<td>VarChar</td>
</tr>
<tr>
<td>NCName</td>
<td>VarChar</td>
</tr>
<tr>
<td>negativeInteger</td>
<td>Decimal</td>
</tr>
<tr>
<td>NMToken</td>
<td>VarChar</td>
</tr>
<tr>
<td>NMTokens</td>
<td>LongVarChar</td>
</tr>
<tr>
<td>nonNegativeInteger</td>
<td>Decimal</td>
</tr>
<tr>
<td>nonPositiveInteger</td>
<td>Decimal</td>
</tr>
<tr>
<td>normalizedString</td>
<td>VarChar</td>
</tr>
<tr>
<td>NOTATION</td>
<td>VarChar</td>
</tr>
<tr>
<td>positiveInteger</td>
<td>Decimal</td>
</tr>
<tr>
<td>QName</td>
<td>VarChar</td>
</tr>
<tr>
<td>short</td>
<td>Numeric</td>
</tr>
<tr>
<td>string</td>
<td>VarChar</td>
</tr>
<tr>
<td>XML data type</td>
<td>SQL data type</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>time</td>
<td>Timestamp</td>
</tr>
<tr>
<td>token</td>
<td>VarChar</td>
</tr>
<tr>
<td>unsignedByte</td>
<td>SmallInt</td>
</tr>
<tr>
<td>unsignedInt</td>
<td>numeric</td>
</tr>
<tr>
<td>unsignedLong</td>
<td>numeric</td>
</tr>
<tr>
<td>unsignedShort</td>
<td>numeric</td>
</tr>
</tbody>
</table>
Chapter 7. Advanced transformations

To transform the XML document to columns and rows, XML Input uses an XSLT stylesheet that it generates from the XPath expressions that are specified on the input link.

If the XML document contains nodes whose relationships are not explicit, XML Input may not be able to automatically perform the transformation. As an alternative to the generated XSLT stylesheet, you can substitute your own stylesheet.

The following provides the following information about transforming XML documents using custom XSLT stylesheets in the XML Input stage:
• Document Type Definition (DTD) to which the output of an XSLT stylesheet must conform
• Procedure for accessing the generated XSLT stylesheet
• Sample input document for which you might use a custom XSLT stylesheet
• Sample XSLT stylesheet that converts the input document to an XML document that conforms to the required DTD
• Generated internal XML document
• Generated output rows, which are available on an output link

Required DTD

The output of your custom XSLT stylesheet must conform to the following DTD:

```
<!ELEMENT table (row*)>
<!ELEMENT row (column*)>
<!ELEMENT column (#PCDATA | NULL)>
<!ATTLIST column
  name CDATA #REQUIRED>
```

The declaration `<!ELEMENT NULL>` is used to set an output column to `NULL`.

Accessing the generated XSLT stylesheet

About this task

You can access the generated XSLT stylesheet through the job log. This stylesheet is a good starting point for creating your own.

To write the internal stylesheet to the log:

**Procedure**

1. Open the InfoSphere DataStage Director.
2. In the Job Run Options dialog box, select the **Tracing** tab.
3. In the Stage names list, select the XML Input stage for the job.
4. Select the **Subroutines calls** box.
Results

To access the stylesheet:

In the Job Log view, open event entries that start with an XML declaration.

Sample input XML document

In the following example, the relationship between orders and addresses is not explicit. If you want to associate each order with every address, use a custom stylesheet. This creates a cross-product.

```xml
<?xml version="1.0" ?>
<customers>
  <customer id="1" name="Acme, Inc.">
    <orders>
      <order order_no="1">
        <items>
          <item item_no="1" quantity="10" />
          <item item_no="2" quantity="2" />
          <item item_no="3" quantity="1" />
        </items>
      </order>
      <order order_no="2">
        <items>
          <item item_no="123" quantity="2" />
          <item item_no="123" quantity="2" />
          <item item_no="321" quantity="3" />
        </items>
      </order>
    </orders>
    <addresses>
      <address street="192 Prospect St" city="Auburn" state="MA" zip="01501" />
      <address street="50 Washington St" city="Westborough" state="MA" zip="01581" />
    </addresses>
  </customer>
  <customer id="2" name="Bubba Gump Shrimp Co, Inc.">
    <orders>
      <order order_no="3">
        <items>
          <item item_no="654" quantity="1" />
        </items>
      </order>
    </orders>
    <addresses>
      <address street="1099 18th St, Suite 2500" city="Denver" state="CO" zip="80202" />
    </addresses>
  </customer>
</customers>
```

Sample XSLT stylesheet

The following XSLT stylesheet forces a relationship between orders and addresses by computing all combinations. The output document conforms to the required DTD.

Note: If you use an XSLT stylesheet that generates an XML document that does not conform to the required DTD, XML Input may produce unusable rows or no rows at all.

```xml
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <table>
      <xsl:apply-templates select="/customers/customer/orders/order"/>
    </table>
  </xsl:template>
</xsl:stylesheet>
```
<xsl:template match="order">
  <xsl:variable name="id" select="../../@id" />
  <xsl:apply-templates select="/customers/customer/addresses/address[../../@id =$id]">
    <xsl:with-param name="id" select="$id" />
    <xsl:with-param name="order_no" select="@order_no" />
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="address">
  <xsl:param name="id" />
  <xsl:param name="order_no" />
  <row>
    <column name="CUSTOMER_ID">
      <xsl:value-of select="$id" />
    </column>
    <column name="ORDER_NO">
      <xsl:value-of select="$order_no" />
    </column>
    <column name="STREET">
      <xsl:value-of select="@street" />
    </column>
    <column name="CITY">
      <xsl:value-of select="@city" />
    </column>
    <column name="STATE">
      <xsl:value-of select="@state" />
    </column>
    <column name="ZIP">
      <xsl:value-of select="@zip" />
    </column>
  </row>
</xsl:template>
</xsl:stylesheet>

---

**Generated XML document**

The sample XSLT stylesheet transforms the input document into the following output document, which conforms to the required DTD. Use this example as a guide in testing your stylesheet before using it in the XML Input stage.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<table>
  <row>
    <column name="CUSTOMER_ID">1</column>
    <column name="ORDER_NO">1</column>
    <column name="STREET">192 Prospect St</column>
    <column name="CITY">Auburn</column>
    <column name="STATE">MA</column>
    <column name="ZIP">01501</column>
  </row>
  <row>
    <column name="CUSTOMER_ID">1</column>
    <column name="ORDER_NO">1</column>
    <column name="STREET">50 Washington St</column>
    <column name="CITY">Westborough</column>
    <column name="STATE">MA</column>
    <column name="ZIP">01581</column>
  </row>
  <row>
    <column name="CUSTOMER_ID">1</column>
    <column name="ORDER_NO">2</column>
    <column name="STREET">192 Prospect St</column>
    <column name="CITY">Auburn</column>
    <column name="STATE">MA</column>
    <column name="ZIP">01501</column>
  </row>
</table>
```
Output table

The following output rows are generated using the output XML document:

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
<th>ORDER_NO</th>
<th>STREET</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>192 Prospect St</td>
<td>Auburn</td>
<td>MA</td>
<td>01501</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>50 Washington St</td>
<td>Westborough</td>
<td>MA</td>
<td>01581</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>192 Prospect St</td>
<td>Auburn</td>
<td>MA</td>
<td>01501</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>50 Washington St</td>
<td>Westborough</td>
<td>MA</td>
<td>01581</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1099 18th St, Suite 2500</td>
<td>Denver</td>
<td>CO</td>
<td>80202</td>
</tr>
</tbody>
</table>
Chapter 8. NULL and empty value processing

The XML Input and XML Output stages process input NULLs and empty values based on property settings.

**XML Input**

**Sample input XML**

```xml
<customers>
  <customer id="0">
    <name>IBM Corporation</name>
    <address city="">
      <street>50 Washington St</street>
    </address>
  </customer>
  <customer id="1">
    <name>Acme, Inc.</name>
    <address>
      <street>10 Broadway</street>
    </address>
  </customer>
  <customer id="2">
    <name>ABC, Corp.</name>
    <address city="Denver">
      <street>1234 Main St</street>
    </address>
  </customer>
</customers>
```

**Output columns**

The sample input XML maps to four output columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>XPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>/customers/customer/@id</td>
</tr>
<tr>
<td>NAME</td>
<td>/customers/customer/name/text()</td>
</tr>
<tr>
<td>STREET</td>
<td>/customers/customer/address/street/text()</td>
</tr>
<tr>
<td>CITY</td>
<td>/customers/customer/address/@city</td>
</tr>
</tbody>
</table>

Repetition path: /customers/customer/@id

**Processing NULLs and empty values**

There are four possible combinations for setting two properties:

- Replace NULLs with empty values
- Replace empty values with NULLs

You set these properties on the Transformation Settings page in the XML Input Stage dialog box.

**One: Replace NULLs (unselected). Replace empty values (unselected)**

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>NAME</th>
<th>STREET</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IBM Corporation</td>
<td>50 Washington St</td>
<td></td>
</tr>
</tbody>
</table>
Two: Replace NULLs (selected). Replace empty values (unselected)

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>NAME</th>
<th>STREET</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IBM Corporation</td>
<td>50 Washi</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>Acme, Inc.</td>
<td>10 Broadway</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ABC, Corp.</td>
<td>1234 Main St</td>
<td>Denver</td>
</tr>
</tbody>
</table>

Three: Replace NULLs (unselected). Replace empty values (selected)

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>NAME</th>
<th>STREET</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IBM Corporation</td>
<td>50 Washington St</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>Acme, Inc.</td>
<td>10 Broadway</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>ABC, Corp.</td>
<td>1234 Main St</td>
<td>Denver</td>
</tr>
</tbody>
</table>

Four: Replace NULLs (selected). Replace empty values (selected)

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>NAME</th>
<th>STREET</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IBM Corporation</td>
<td>50 Washington St</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>Acme, Inc.</td>
<td>10 Broadway</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>ABC, Corp.</td>
<td>1234 Main St</td>
<td>Denver</td>
</tr>
</tbody>
</table>

XML Output

Sample input

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>NAME</th>
<th>STREET1</th>
<th>STREET2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IBM Corporation</td>
<td>50 Washi</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Acme, Inc.</td>
<td>10 Broadway</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>ABC, Corp.</td>
<td>1234 Main St</td>
<td>Suite 4321</td>
</tr>
</tbody>
</table>

Input columns

The sample input XML maps to four output columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>XPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>/customers/customer/@id</td>
</tr>
<tr>
<td>NAME</td>
<td>/customers/customer/name/text()</td>
</tr>
<tr>
<td>STREET1</td>
<td>/customers/customer/address/street1/text()</td>
</tr>
<tr>
<td>STREET2</td>
<td>/customers/customer/address/street2/text()</td>
</tr>
</tbody>
</table>
Repetition path: /customers/customer/@id

**Processing NULLs and empty values**
There are four possible combinations for setting two properties:
- Replace NULLs with empty values
- Replace empty values with NULLs

You set these properties on the Transformation Settings page in the XML Output Stage dialog box.

**One: Replace NULLs (unselected). Replace empty values (unselected)**
```
<customers>
  <customer id="0">
    <name>IBM Corporation</name>
    <address>
      <street1>50 Washington St</street1>
      <street2/>
    </address>
  </customer>
  <customer id="1">
    <name>Acme, Inc.</name>
    <address>
      <street1>10 Broadway</street1>
    </address>
  </customer>
  <customer id="2">
    <name>ABC, Corp.</name>
    <address>
      <street1>1234 Main St</street1>
      <street2>Suite 4321</street2>
    </address>
  </customer>
</customers>
```

**Two: Replace NULLs (selected). Replace empty values (unselected)**
```
<customers>
  <customer id="0">
    <name>IBM Corporation</name>
    <address>
      <street1>50 Washington St</street1>
    </address>
  </customer>
  <customer id="1">
    <name>Acme, Inc.</name>
    <address>
      <street1>10 Broadway</street1>
    </address>
  </customer>
  <customer id="2">
    <name>ABC, Corp.</name>
    <address>
      <street1>1234 Main St</street1>
      <street2>Suite 4321</street2>
    </address>
  </customer>
</customers>
```

**Three: Replace NULLs (unselected). Replace empty values (selected)**
```
<customers>
  <customer id="0">
    <name>IBM Corporation</name>
    <address>
```
<street1>50 Washington St</street1>
</address>
</customer>
</customers>

**Four: Replace NULLs (selected). Replace empty values (selected)**

<customers>
  <customer id="0">
    <name>IBM Corporation</name>
    <address>
      <street1>50 Washington St</street1>
    </address>
  </customer>
  <customer id="1">
    <name>Acme, Inc.</name>
    <address>
      <street1>10 Broadway</street1>
    </address>
  </customer>
  <customer id="2">
    <name>ABC, Corp.</name>
    <address>
      <street1>1234 Main St</street1>
      <street2>Suite 4321</street2>
    </address>
  </customer>
</customers>
Chapter 9. Local codepages on engine tier hosts

Data is encoded using the local codepage of the engine tier host when:

- The engine runs in non-NLS mode. This applies to reading from an XML file and to writing to an XML file.
- A file that resides on an engine tier host machine is accessed through a file path. This applies to both NLS modes.

To ensure uniform and expected results when using the XML Pack, the local codepage of the engine tier host should be set for the locale that you want.

Setting the local codepage

The system administrator sets the codepage for the locale on the engine tier host.

The method used to select the local codepage depends upon the operating system on which engine is running. Consult your system administrator.

Setting a locale in a POSIX system

This section describes the method for setting a local codepage in a POSIX system, and provides an example.

Method

About this task

- Set the locale using the LANG environment variable. This variable is typically in the dsenv file.
- To display the current locale, use the locale command.
- To display a list of available locales (possible values for the LANG variable), use the following command:
  
  `locale -a`

Example of setting the locale

The locale is set to UTF-8:

```bash
set LANG=en_US.UTF-8
```

Character data are interpreted as UTF-8 values.

Sample File Path

```
/tmp/data/direcci\179\195\19n
```

where:

\179\195 is interpreted as ó (letter o with an acute accent).

The complete path is interpreted as:

```
/tmp/data/dirección
```

If you change the locale to ISO-8859-1, the sequence \179\195 is interpreted as ³Å (superscript 3 followed by a capital A with a tilde)

Note: Not all characters have representations in all character sets. For example, values above \127 have no equivalents in the ASCII character set (en_US).
Appendix A. Product accessibility

You can get information about the accessibility status of IBM products.

The IBM InfoSphere Information Server product modules and user interfaces are not fully accessible. The installation program installs the following product modules and components:
• IBM InfoSphere Business Glossary
• IBM InfoSphere Business Glossary Anywhere
• IBM InfoSphere DataStage
• IBM InfoSphere FastTrack
• IBM InfoSphere Information Analyzer
• IBM InfoSphere Information Services Director
• IBM InfoSphere Metadata Workbench
• IBM InfoSphere QualityStage®

For information about the accessibility status of IBM products, see the IBM product accessibility information at http://www.ibm.com/able/product_accessibility/index.html.

Accessible documentation

Accessible documentation for InfoSphere Information Server products is provided in an information center. The information center presents the documentation in XHTML 1.0 format, which is viewable in most Web browsers. XHTML allows you to set display preferences in your browser. It also allows you to use screen readers and other assistive technologies to access the documentation.

The documentation that is in the information center is also provided in PDF files, which are not fully accessible.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.
Appendix B. Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You also can provide feedback to IBM about products and documentation.

The following table lists resources for customer support, software services, training, and product and solutions information.

Table 2. IBM resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description and location</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Support Portal</td>
<td>You can customize support information by choosing the products and the topics that interest you at <a href="http://www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server">www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server</a></td>
</tr>
<tr>
<td>Software services</td>
<td>You can find information about software, IT, and business consulting services, on the solutions site at <a href="http://www.ibm.com/businesssolutions/">www.ibm.com/businesssolutions/</a></td>
</tr>
<tr>
<td>My IBM</td>
<td>You can manage links to IBM Web sites and information that meet your specific technical support needs by creating an account on the My IBM site at <a href="http://www.ibm.com/account/">www.ibm.com/account/</a></td>
</tr>
<tr>
<td>Training and certification</td>
<td>You can learn about technical training and education services designed for individuals, companies, and public organizations to acquire, maintain, and optimize their IT skills at <a href="http://www.ibm.com/software/sw-training/">http://www.ibm.com/software/sw-training/</a></td>
</tr>
</tbody>
</table>
Appendix C. Accessing and providing feedback on the product documentation

Documentation is provided in a variety of locations and formats, including in help that is opened directly from the product client interfaces, in a suite-wide information center, and in PDF file books.

The information center is installed as a common service with IBM InfoSphere Information Server. The information center contains help for most of the product interfaces, as well as complete documentation for all the product modules in the suite. You can open the information center from the installed product or from a Web browser.

Accessing the information center

You can use the following methods to open the installed information center.

• Click the Help link in the upper right of the client interface.

  Note: From IBM InfoSphere FastTrack and IBM InfoSphere Information Server Manager, the main Help item opens a local help system. Choose Help > Open Info Center to open the full suite information center.

• Press the F1 key. The F1 key typically opens the topic that describes the current context of the client interface.

  Note: The F1 key does not work in Web clients.

• Use a Web browser to access the installed information center even when you are not logged in to the product. Enter the following address in a Web browser: http://host_name:port_number/infocenter/topic/com.ibm.swg.im.iis.productization.iisinfsv.home.doc/ic-homepage.html. The host_name is the name of the services tier computer where the information center is installed, and port_number is the port number for InfoSphere Information Server. The default port number is 9080. For example, on a Microsoft® Windows® Server computer named iisdocs2, the Web address is in the following format: http://iisdocs2:9080/infocenter/topic/com.ibm.swg.im.iis.productization.iisinfsv.nav.doc/dochome/iisinfsrv_home.html.

A subset of the information center is also available on the IBM Web site and periodically refreshed at http://pic.dhe.ibm.com/infocenter/iisinfsv/v9r1/index.jsp

Obtaining PDF and hardcopy documentation

• A subset of the PDF file books are available through the InfoSphere Information Server software installer and the distribution media. The other PDF file books are available online and can be accessed from this support document: https://www.ibm.com/support/docview.wss?uid=swg27008803&cwv=1

• You can also order IBM publications in hardcopy format online or through your local IBM representative. To order publications online, go to the IBM Publications Center at http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss.

Providing comments on the documentation

Your feedback helps IBM to provide quality information. You can use any of the following methods to provide comments:

• To comment on the information center, click the Feedback link on the top right side of any topic in the information center.

• Send your comments by using the online readers’ comment form at www.ibm.com/software/awdtools/rcf/
• Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and part number of the information (if applicable). If you are commenting on specific text, include the location of the text (for example, a title, a table number, or a page number).

• You can provide general product feedback through the Consumability Survey at [http://www.ibm.com/software/ucd/consumability/](http://www.ibm.com/software/ucd/consumability/)
Notices and trademarks

This information was developed for products and services offered in the U.S.A.

Notices

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
Licensees of this program who wish to have information about it for the purpose of enabling: (i) the 
exchange of information between independently created programs and other programs (including this 
one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, 
payment of a fee.

The licensed program described in this document and all licensed material available for it are provided 
by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or 
any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the 
results obtained in other operating environments may vary significantly. Some measurements may have 
been made on development-level systems and there is no guarantee that these measurements will be the 
same on generally available systems. Furthermore, some measurements may have been estimated through 
extrapolation. Actual results may vary. Users of this document should verify the applicable data for their 
specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their 
published announcements or other publicly available sources. IBM has not tested those products and 
cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM 
products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of 
those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without 
notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the 
products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate 
them as completely as possible, the examples include the names of individuals, companies, brands, and 
products. All of these names are fictitious and any similarity to the names and addresses used by an 
actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming 
techniques on various operating platforms. You may copy, modify, and distribute these sample programs 
in any form without payment to IBM, for the purposes of developing, using, marketing or distributing 
application programs conforming to the application programming interface for the operating platform for 
which the sample programs are written. These examples have not been thoroughly tested under all 
conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these 
programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be 
liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright 
notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © 
Copyright IBM Corp. _enter the year or years_. All rights reserved.
If you are viewing this information softcopy, the photographs and color illustrations may not appear.

**Trademarks**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

The following terms are trademarks or registered trademarks of other companies:

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The United States Postal Service owns the following trademarks: CASS, CASS Certified, DPV, LACSLink, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACSLink licensee of the United States Postal Service.

Other company, product or service names may be trademarks or service marks of others.
## Index

### Special characters
- `@` column index flag 30
- `%%` column index flag 30

### A
- aggregating input rows, XML Output 27, 42
- Apache Xalan 3
- attributes, missing replacing 16, 19
- Auto-check command, XML Meta Data Importer 9

### C
- character encodings input documents 11, 45
- column index flags 30
- column names, in output files 30
- comment tag, for XML output 33, 40
- consolidating input rows, XML Output 27, 42
- customer support contacting 69

### D
- data type conversions 55
defns namespace prefix 7
dsenv file (UNIX), InfoSphere DataStage Server locale 65
- DTD

### E
- elementFormDefault attribute 7
elements, missing replacing 16, 19, 61
- elements, order of 37
- empty element style 33, 42
- empty values, replacing
  - XML Input 16, 19, 61
  - XML Output 32, 42, 63
- encodings 65
- entity references 25

### F
- fatal errors, XML parsing 13, 19, 26, 42, 47, 51
- Fatal, InfoSphere DataStage error level 13, 26, 47
- file paths, and local codepages 65
- file paths, on output links
  - XML Output 32
  - XML Transformer 46
- full schema constraint checking 26

### H
- header elements for XML output 33, 40
- hierarchies. See XML hierarchies 5

### I
- indenting output 33
- Info, InfoSphere DataStage error level 13, 26, 47
- InfoSphere DataStage
  - NLS modes 11, 45
- InfoSphere DataStage Server locale (UNIX), dsenv file 65
- inheriting link properties XML Transformer 52
- input columns
  - passthrough to output 14, 48
  - XML Input 20
- input link properties
  - XML Input 20
  - XML Output 43
  - XML Transformer 52
- input rows aggregating on output 27, 42
- inserting in XML output 41

### J
- job diagrams
  - XML Input 18
  - XML Output 39
  - XML Transformer 50
- job log
  - and rejection messages 14, 18, 27, 42, 48, 50
  - and stylesheets 57

### K
- Key property
  - and repetition element 16

### L
- language settings 65
- legal notices 73
- links, supported
  - XML Input 11
  - XML Output 23
  - XML Transformer 45
- local codepages 65
- locale 65

### M
- metadata, importing 5

### N
- namespace declarations
  - in root element tags 33, 41
  - XML Input requirement 16, 19
  - XML Output option 33, 41
- namespace prefixes generating with XML Meta Data Importer 7
- namespaces
  - processing in XML Meta Data Importer 7
  - National Language Support
    - for input documents 11, 45
    - for output documents 45
- nested chunks
  - for XML output 33, 41
- new line character 33, 42
- nodes, and repeating elements 33
- non-fatal errors, XML parsing 13, 19, 26, 47, 51
- ns# namespace prefixes 7
- NULLs, replacing
  - XML Input 16, 19, 61
  - XML Output 32, 42, 63

### O
- ordering output elements 37
- output columns
  - column index for 30
  - output link properties
    - XML Input 20
    - XML Output 43
    - XML Transformer 52
- output link, file paths on
  - XML Output 32
  - XML Transformer 46
- output rows
  - and XML Input 14
  - and XML Output 27, 42
  - and XML Transformer 46
- column triggers for 27, 42
- one per input row 31, 42
- passthrough mechanism 14, 31, 48
- row index for 31
- output rows, writing to disk
  - XML Output 30, 42
  - XML Transformer 46, 51, 52

### P
- parallel job
  - passthrough 14, 30, 48
- parser errors 13, 26, 47
parser output, and XML Meta Data Importer 5
passthrough mechanism 14, 31, 48, 53
product accessibility
accessibility 67
product documentation
accessing 71

R
recursive XML 9
Reject links, creating
XML Input 20, 43
XML Transformer 52
Reject links, overview
XML Input 13, 26
XML Transformer 47
Reject, DataStage error level 13, 47
Reject, InfoSphere DataStage error level 26
rejected rows
writing to Reject link 14, 27, 48
rejection messages
writing to job log 14, 18, 27, 42, 48, 50
writing to Reject link 13, 26, 47
rejection messages, output column
XML Input 20, 43
XML Transformer 52
repeating elements, in XML output 33
repetition element
Key property 16, 34
XML Input 14
XML Output 33
root element tags
and namespace declarations 33, 41
row generation
XML Input 14
XML Output 31
XML Transformer 46

S
SAX parser errors 13, 26, 47
schema files. See XML schema files 6
schemaLocation attribute 12, 25, 46
semantic errors
and XML Meta Data Importer 5
sequence of elements 37
software services
contacting 69
SQL data types 55
SQL NULL, replacing 32
stage properties
XML Input 18
XML Output 39
XML Transformer 50
stylesheets
and output encoding 45
and XML Input 21, 57
and XML Transformer 51
sample 58
support
customer 69
syntax errors
and XML Meta Data Importer 5

T
table definitions
loading meta data from 22, 43
mapping schema nodes 8
modifying in XML Meta Data Importer 9
saving in XML Meta Data Importer 9
target namespaces
assigning prefixes to 7
missing 8
Trace, InfoSphere DataStage error level 13, 26, 47
trademarks
list of 73
transforming XML documents 45, 51, 52

U
UTF-8 encoding, output columns 12, 45

W
Warning, InfoSphere DataStage error level 13, 26, 47
warnings, XML parsing 13, 19, 26, 42, 47, 51
well-formed XML
XML Input 12, 25
XML Transformer 46
writing XML to disk
XML Output 30, 42
XML Transformer 46, 51, 52

X
Xalan XSLT processor 3
XML branches
mapping to columns 24
XML chunk 41
generating as output 33, 40
inserting in XML output 33
XML data types 55
XML documents
generating 30, 46
validating against schema 12, 18, 42, 46, 50
XML fragment
generating as output 33, 40
inserting in XML output 33, 41
XML fragments, formatting 19
XML hierarchies
tree view 5
XML Input, overview 2
XML Meta Data Importer
overview 3
starting 6
XML Output
overview 2
supported XPath expressions 23
XML Pack, overview 1
XML schema files
and namespaces 7
for XML output 33, 41
mapping nodes to table definitions 8
XML schema files (continued)
opening in XML Meta Data Importer 6
validating documents against 12, 46
XML source, browsing
in XML Meta Data Importer 5
XML source, identifying
XML Input 11, 20
XML Transformer 45, 52
XML Transformer, overview 3
XML-safe 25
XPath expressions
overview 1
XPath expressions, editing
and XML Meta Data Importer 9
XPath expressions, on input links
XML Output 24, 43
XPath expressions, on output links
XML Input 11, 22
XML Output 24, 43
xsd files, importing metadata 6
XSLT processor 3
XSLT stylesheets
and job log 57
and output encoding 45
and XML Input 21
and XML Transformer 45, 51, 52
default in XML Input 57
sample 58