IBM InfoSphere DataStage
Version 9 Release 1

XML Transformation Guide

IBM
IBM InfoSphere DataStage
Version 9 Release 1

XML Transformation Guide

IBM
Before using this information and the product that it supports, read the information in "Notices and trademarks" on page 259.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XML transformation</strong></td>
<td>1</td>
</tr>
<tr>
<td>Schema management</td>
<td>1</td>
</tr>
<tr>
<td>Opening the Schema Library Manager</td>
<td>2</td>
</tr>
<tr>
<td>Working with libraries and resources</td>
<td>3</td>
</tr>
<tr>
<td>Creating an example schema library</td>
<td>4</td>
</tr>
<tr>
<td>Using the XML stage</td>
<td>6</td>
</tr>
<tr>
<td>Adding an XML stage to a job</td>
<td>7</td>
</tr>
<tr>
<td>Configuring runtime properties for the XML stage</td>
<td>7</td>
</tr>
<tr>
<td>The assembly</td>
<td>8</td>
</tr>
<tr>
<td>Overview</td>
<td>8</td>
</tr>
<tr>
<td>Input step</td>
<td>9</td>
</tr>
<tr>
<td>Output step</td>
<td>9</td>
</tr>
<tr>
<td>Assembly Editor</td>
<td>10</td>
</tr>
<tr>
<td>Opening the Assembly Editor</td>
<td>11</td>
</tr>
<tr>
<td>Modifying items in the step Output</td>
<td>11</td>
</tr>
<tr>
<td>Testing assemblies</td>
<td>12</td>
</tr>
<tr>
<td>Parsing large schemas</td>
<td>13</td>
</tr>
<tr>
<td>Icons in the schema tree</td>
<td>13</td>
</tr>
<tr>
<td>Details Inspector</td>
<td>16</td>
</tr>
<tr>
<td>Mapping data</td>
<td>18</td>
</tr>
<tr>
<td>Working with the mapping table</td>
<td>18</td>
</tr>
<tr>
<td>Determining mapping candidates</td>
<td>19</td>
</tr>
<tr>
<td>Configuring how mapping candidates are determined</td>
<td>24</td>
</tr>
<tr>
<td>XML Parser step</td>
<td>25</td>
</tr>
<tr>
<td>XML Parser validation rules</td>
<td>26</td>
</tr>
<tr>
<td>Controlling how specific items are parsed</td>
<td>32</td>
</tr>
<tr>
<td>Setting default values for types</td>
<td>32</td>
</tr>
<tr>
<td>XML Composer step</td>
<td>33</td>
</tr>
<tr>
<td>XML Composer validation rules</td>
<td>34</td>
</tr>
<tr>
<td><strong>JSON transformation</strong></td>
<td>37</td>
</tr>
<tr>
<td>Schema management</td>
<td>38</td>
</tr>
<tr>
<td>JSON Parser step</td>
<td>40</td>
</tr>
<tr>
<td>JSON Composer step</td>
<td>43</td>
</tr>
<tr>
<td>Large Schema</td>
<td>46</td>
</tr>
<tr>
<td>Auto Chunking</td>
<td>46</td>
</tr>
<tr>
<td>Schema views</td>
<td>50</td>
</tr>
<tr>
<td>Parsing an XML file using schema views</td>
<td>51</td>
</tr>
<tr>
<td>Composing an XML file using schema views</td>
<td>53</td>
</tr>
<tr>
<td>Transformation steps for the XML stage</td>
<td>60</td>
</tr>
<tr>
<td>Aggregate step</td>
<td>60</td>
</tr>
<tr>
<td>H-Pivot step</td>
<td>61</td>
</tr>
<tr>
<td>HJoin step</td>
<td>61</td>
</tr>
<tr>
<td>Order Join step</td>
<td>61</td>
</tr>
<tr>
<td>Regroup step</td>
<td>62</td>
</tr>
<tr>
<td>Sort step</td>
<td>62</td>
</tr>
<tr>
<td>Switch step</td>
<td>62</td>
</tr>
<tr>
<td>Union step</td>
<td>64</td>
</tr>
<tr>
<td>V-Pivot step</td>
<td>64</td>
</tr>
<tr>
<td>Examples of transforming XML data</td>
<td>64</td>
</tr>
<tr>
<td>Example 1: Parsing XML data</td>
<td>64</td>
</tr>
<tr>
<td>Example 2: Using the XML Composer and Regroup steps</td>
<td>75</td>
</tr>
<tr>
<td>Example 3: Using the XML Composer and HJoin steps</td>
<td>84</td>
</tr>
<tr>
<td>Example 4: Using the XML Parser and Switch steps</td>
<td>92</td>
</tr>
<tr>
<td>Example 5: Using the XML Parser and Union steps</td>
<td>106</td>
</tr>
<tr>
<td>Example 6: Using the XML Composer and H-Pivot steps</td>
<td>121</td>
</tr>
<tr>
<td>Example 7: Using the XML Parser and Aggregate steps</td>
<td>129</td>
</tr>
<tr>
<td>Example 8: Using the XML Composer and Sort steps</td>
<td>140</td>
</tr>
<tr>
<td>Example 9: Using the XML Composer and OrderJoin steps</td>
<td>156</td>
</tr>
<tr>
<td>Example 10: Using the XML Parser and V-Pivot steps</td>
<td>168</td>
</tr>
<tr>
<td>Examples of transforming JSON data</td>
<td>180</td>
</tr>
<tr>
<td>Example 1: Parsing JSON data</td>
<td>180</td>
</tr>
<tr>
<td>Example 2: Composing JSON data by using the JSON Composer and HJoin steps</td>
<td>190</td>
</tr>
<tr>
<td>Reference</td>
<td>200</td>
</tr>
<tr>
<td>Supported XML types and type mappings</td>
<td>200</td>
</tr>
<tr>
<td>Files for XML stage examples</td>
<td>204</td>
</tr>
<tr>
<td>Messages</td>
<td>205</td>
</tr>
<tr>
<td>Working with IBM Software Support to diagnose problems</td>
<td>217</td>
</tr>
<tr>
<td>Links to developer articles</td>
<td>217</td>
</tr>
<tr>
<td><strong>Appendix A. Product accessibility</strong></td>
<td>219</td>
</tr>
<tr>
<td><strong>Appendix B. Reading command-line syntax</strong></td>
<td>221</td>
</tr>
<tr>
<td><strong>Appendix C. How to read syntax diagrams</strong></td>
<td>223</td>
</tr>
<tr>
<td><strong>Appendix D. Contacting IBM</strong></td>
<td>225</td>
</tr>
<tr>
<td><strong>Appendix E. Accessing and providing feedback on the product documentation</strong></td>
<td>227</td>
</tr>
<tr>
<td>Notices and trademarks</td>
<td>229</td>
</tr>
<tr>
<td>Index</td>
<td>233</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 2011, 2013
**XML transformation**

Use the XML stage to create powerful hierarchical transformations, as well as parse and compose XML data with high performance and scalability.

Many industries have standardized on Extensible Markup Language (XML) as the mechanism to use to exchange information between organizations. With the broader acceptance and use of these standards, companies have increasingly looked to XML to also satisfy requirements for the exchange of information between different IT units within their organization. The business projects for which XML has been adopted have generated specific requirements for IT. In some cases, the data volume that is represented in an XML document is minimal. For example, the data might represent a single transaction, but it might have many layers of hierarchical complexity. Other projects require that multi-gigabyte files with relatively simple XML schemas be transformed into a new format that prescribes to an industry standard. When the data volume and complex hierarchy requirements meet, they often present their own challenges to traditional IT tools.

The XML stage includes capabilities that easily manage the design and processing requirements presented by the most challenging XML sources. The IT developer can leverage the schema library manager to register the XML metadata in its native form. This metadata forms the basis which guide the design activities. The XML stage uses an integrated user interface component called the assembly editor to facilitate the transformation of XML data from hierarchical to relational or other hierarchical formats, or from relational to hierarchical formats. After the logic is constructed, the job runtime leverages unique components that provide various forms of parallelism that are built specifically for hierarchical data formats, such XML. With these mechanisms IBM® InfoSphere® DataStage® scales to meet very high volumes and manage system resources efficiently.

**Choosing an XML solution**

InfoSphere DataStage provides two XML solutions: the XML pack and the XML stage. The XML pack, which includes the XML Input, XML Output, and XML Transformer stages, is useful if you have already made an investment in using this technology or if you want to perform only very simple transformations that do not involve a large amount of data. The XML stage the best choice if you have not yet created an XML solution and want perform complex transformations on large amounts of data.

**Schema management**

Use the Schema Library Manager to import schemas into the metadata repository and to organize them into libraries that can be shared across all DataStage projects.

Before you can use the XML stage to produce or consume data, you must import the XML schemas that describe the data into the metadata repository. To import the schemas, you use the Schema Library Manager, which is available from the IBM InfoSphere DataStage and QualityStage® Designer via the menu choice Import > Schema Library Manager.

After you import a schema, you can browse the type structure that the schema defines. You use these types to define the processing in the XML stage. The
schemas that you import are available for use with any data. To use a schema with specific data, you bind the physical location of the data to the schema. For example, to read an XML file, you provide a path to the XML file and select the schema for the data that the file contains. When you work with XML data, you perform the binding when you configure the XML Parser or XML Composer step in an assembly.

You can add related schemas to the same library. Schemas that are in the same library can refer to each other. For example, one XML schema can use an XML include element or an import element to refer to another schema that is in the same library. Schemas that are in one library cannot refer to external schemas or to schemas that are in another library.

When you create a library, you specify a unique name for the library and an optional category name. Library names must be unique across all categories. Organizing libraries into categories ensures that you can later locate a specific library.

After you add schemas to a library, the library is automatically validated to determine if the schemas contain any errors. If the validation is successful, the library contains all of the element and type definitions from the schemas. If the library is invalid, you are notified that there are errors. To view the list of errors, click Validate. Whenever you modify a schema, delete a schema from the library, or add a new schema to the library the library is automatically re-validated. Schemas are used only at design time, not at runtime. Therefore, modifying a schema or deleting a schema from a library has no effect on existing jobs that use the schema.

If you modify a schema that is already being used by a job, the schema modifications are not automatically passed on to the job. If you want to apply a modified schema to an XML stage, you must edit the XML stage to retrieve the modified schema.

Within a library, you cannot repeat the same type definition; however, two different libraries can have the same type definition. Having the same type definition in two different libraries is useful if you want to have two versions of the same type definition.

**Opening the Schema Library Manager**

You use the Schema Library Manager to import the schemas to use in jobs that include the XML stage.

**About this task**

You can open the Schema Library Manager from IBM InfoSphere DataStage and QualityStage Designer or from the Libraries tab, which is available from the Assembly Editor.

**Procedure**

Do one of the following:

- From the InfoSphere DataStage and QualityStage Designer, choose Import > Schema Library Manager.
- From the Assembly Editor, click the Libraries tab.
Working with libraries and resources

Use the Schema Library Manager to import resources, such as XML schemas, and to create and manage libraries of resources.

Use the istool command line to transfer contract libraries between metadata repositories of IBM InfoSphere Information Server. Libraries are stored in the metadata repository and are available for use in any job that uses the XML stage. In the metadata repository, a contract library is represented by the library name, followed by the extension .cl. When you work with multiple installations of InfoSphere Information Server, you might want to transfer a contract library from one installation to another installation, for example, from a test environment to a production environment.

You can import and export contract libraries using the istool functionality. For more information, see . For more information about exporting and importing common metadata assets by using the command line, see http://publib.boulder.ibm.com/infocenter/iisinfsv/v8r7/topic/com.ibm.swg.im.iis.iisinfsv.assetint.doc/topics/pdrassets.html.

Table 1. Working with libraries

<table>
<thead>
<tr>
<th>Task</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating libraries</td>
<td>1. Click New Library.</td>
</tr>
<tr>
<td></td>
<td>2. Enter a name.</td>
</tr>
<tr>
<td></td>
<td>3. Enter a description and the name of a new or existing category.</td>
</tr>
<tr>
<td>Refreshing library lists</td>
<td>Click the Refresh icon at the top of the Libraries pane.</td>
</tr>
<tr>
<td>Removing libraries</td>
<td>Select a library, and click Remove.</td>
</tr>
<tr>
<td>Opening libraries</td>
<td>Select a library, and click Open. You can open a library only if it has been validated and has no errors. The Types tab displays a list of namespaces and the belonging types. To view the schema for a type, select a type from the Types tab. To view the facets and attributes for a node, select the node from the Schema tab.</td>
</tr>
<tr>
<td>Recategorizing libraries</td>
<td>To move a library to a different category, edit the Category field in the library details.</td>
</tr>
</tbody>
</table>

Table 2. Working with resources

<table>
<thead>
<tr>
<th>Task</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importing resources</td>
<td>1. Select the name of a library, and click Import new resource.</td>
</tr>
<tr>
<td></td>
<td>2. Select the file to import, and click Open. You can select a single .xsd file or a .zip file that contains multiple .xsd files.</td>
</tr>
<tr>
<td></td>
<td>3. After importing all of the required resources into the library, the library is automatically validated. If the library is valid, a green checkmark displays beside its name. If it is not valid, a red exclamation point displays beside its name.</td>
</tr>
<tr>
<td></td>
<td>4. If the library is invalid, click Validate to display the list of errors.</td>
</tr>
</tbody>
</table>
Table 2. Working with resources (continued)

<table>
<thead>
<tr>
<th>Task</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displaying the types in schemas</td>
<td>Double-click the name of a library to display the <strong>Types</strong> and <strong>Schema</strong> tabs. On the <strong>Types</strong> tab, click the plus sign beside a namespace to display a list of types that belong to the namespace. Click a type to display the schema for the type on the <strong>Schemas</strong> tab. Browsing the types and their structures is useful for a variety of reasons. You can determine which type corresponds to a specific document instance by matching the top-level element name with the type name. You can determine which pieces of information are optional but could be included in similar documents. You can view the characteristics of the data, so that you can understand how to set transformation options.</td>
</tr>
<tr>
<td>Exporting resources</td>
<td>Select the resource, and then click <strong>Export</strong> to save a copy of the resource on the local computer.</td>
</tr>
<tr>
<td>Updating resources</td>
<td>If you modify a resource and want to update it in the library, select the resource, and then click <strong>Replace Selected Resource</strong>. The updated resource has no effect on jobs that used the earlier version of the resource. To use the updated version in an existing job, you must edit the assembly.</td>
</tr>
<tr>
<td>Deleting resources</td>
<td>Click the <strong>Delete</strong> icon that displays beside the resource name. Deleting a resource has no effect on existing jobs that use the resource.</td>
</tr>
<tr>
<td>Displaying the resources in a library</td>
<td>Click the name of a library. For the XML stage, the Resources view displays a file location and a namespace for each schema in the library. Click the name of a schema to display additional details about each schema.</td>
</tr>
<tr>
<td>Editing resource details</td>
<td>Only the <strong>File Location</strong> and <strong>Description</strong> fields can be updated. After you update the file location, the library is automatically validated.</td>
</tr>
</tbody>
</table>

Related information:


Creating an example schema library

Create an example of a library and import the schemas.

**About this task**

The library can have more than one schema. The schemas can be interdependent. If the schemas are interdependent and if you import only one schema, the library displays an error. To ensure that the library is valid you must import the dependent schema.

You can download the files, `department.xsd` and `definition.xsd` using the link, [http://www.ibm.com/support/docview.wss?uid=swg27019894](http://www.ibm.com/support/docview.wss?uid=swg27019894). Open the xml examples zip file and select the folder, `example_library` to download these files.

**Procedure**

1. Start the IBM InfoSphere DataStage and QualityStage Designer.
2. Choose **Import > Schema Library Manager**.
3. On the Libraries tab, click **New Library**.
4. In the New Contract Library window, enter Example_library for the name of the library.
5. Click **OK** to save the library.
6. Select the library that you created, Example_library, and then click **Import New Resource** from the **Resource View**.
7. Select the schema, department.xsd.
8. The File Upload Complete window is displayed.
9. Click **OK**.
   - The library is not valid and the error message, “One or more resources failed validation. Click Validate to view the errors” is displayed.
10. Click **Validate** to view the actual error. Following figure shows the error.
   - The error indicates that the schema department.xsd has a dependency on another schema, definition.xsd. The schema department.xsd contains an element named dept_id that has the type dept_id1. Because the type dept_id1 is defined in the schema definition.xsd, you need to import it, too.
11. Import definition.xsd and click **Validate**. The error still exists because the file location for definition.xsd is incorrect.
12. Perform the following steps to specify the file location:
   a. Open department.xsd in a notepad or a wordpad to get the schema location. The department.xsd is shown below:
   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <xs:schema xmlns="http://www.ibm.com/departments"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/departments">
     <xs:include schemaLocation="http://ibm.com/definitions/definition.xsd” />
     <xs:element name="Info">
       <xs:complexType>
         <xs:sequence>
           <xs:element name="dept_id" type="dept_id1"/>
         </xs:sequence>
       </xs:complexType>
     </xs:element>
   </xs:schema>
   ```
   b. Copy the schema location, http://ibm.com/definitions/definition.xsd from the include statement.
   c. In the **Resources View**, select definition.xsd and paste the schema location below in the **File Location** field as shown in the following figure.
13. Click on any other field to refresh the view. The error gets resolved and the library becomes valid. The following figure shows the resolved view.

14. Click OK to close the Libraries window.

**Using the XML stage**

Use the XML stage to parse, compose, and transform XML data.

The XML stage is available in the Real Time section of the palette in the IBM InfoSphere DataStage and QualityStage Designer. You can use the XML stage in server jobs and in parallel jobs. In parallel jobs, the XML stage can have multiple input and output links. In server jobs, the XML stage can have only one input link, but it can have multiple output links. You can use the XML stage as source stage, which has only output links; a middle stage, which has both input and output links; or a target stage, which has only input links.

The XML stage page displays a preview window that shows the XML stage and any input or output links. The page displays a set of runtime properties that you can configure, and the **Edit assembly** button. When you select the stage in the
preview window and then click Edit assembly, the Assembly Editor opens. You use the Assembly Editor to create the assembly for the XML stage. When you close the Assembly Editor, the assembly is saved as part of the stage.

To create the columns on an input or output link, click the link in the preview window to display the links view. Then click the Columns tab and create the columns on the link.

Adding an XML stage to a job

Use the InfoSphere DataStage and QualityStage Designer client to add an XML stage to a job.

Procedure

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and select New > Parallel job.
3. Open the Real Time section of the palette, and drag XML stage to the canvas.
4. Open the File section of the palette, and Sequential File stage to the canvas. Position these stages to the right of the XML stage.
5. Create a link from the XML stage to each sequential file stage.
6. Modify the default name of the stage and the link.

   Note: When designing a DataStage job with XML Stage, decide the name of the DataStage input and output link before configuring the mapping within the XML stage. If you try to rename the input or output link, then it invalidates all the configured mapping within the XML stage. Similarly, when copying an XML stage, you must reconfigure the mapping for the copied XML stage.

7. Choose File > Save, and name the job.

Configuring runtime properties for the XML stage

To modify how the XML stage runs in the job, configure these properties.

Heap Size (MB)

Enter the maximum heap size, in MB, for the Java™ Virtual Machine. This property corresponds to the -Xmx command line option. Specify at least 256 MB. Consider specifying a larger heap size if the assembly includes a large schema and large documents.

Stack Size (KB)

Enter the size of the stack, in KB, for the Java Virtual Machine. This property corresponds to the -Xss command line option.

Other Options

Enter additional command line arguments to the Java Virtual Machine. For example, to set a system property, enter DSystemProperty=Value.

Use Scratch Disk

Yes (default) specifies that at runtime, data might be written to disk if processing does not fit into main memory. If set to No, the job might fail at runtime if processing does not fit into main memory. Performance is faster if you do not use a scratch disk; however, you risk not having enough memory for the job.

Limit Output Rows

Enter Yes for Limit output rows, and then enter a value for Maximum
output rows. Limiting output rows is useful when you are debugging a job and do not want to wait for the results of the entire job.

Maximum Output Rows
Enter Yes for Limit output rows, and then enter a value for Maximum output rows. Output rows are calculated cumulatively across all output links. The XML stage stops processing after generating the maximum number of output rows.

Enable logging
Enter Yes to enable logging and set the log level to Warning. By default, logging is not enabled. Enabling logging affects performance.

Log level
Specify the level of detail to log. Choosing a high level of logging affects performance.

Assembly Date
The date that the assembly was last saved.

The assembly
To configure the XML stage, you create an assembly.

An assembly is a series of steps that perform enrichments and transformations on hierarchical data. By default, a new assembly always contains an assembly Overview, an Input step, and an Output step. Between the Input step and the Output step, you insert additional steps that parse, compose, or transform the data from the previous step.

The assembly Overview provides creation and modification information about the assembly and includes an optional field for the assembly description. This field documents the purpose of the assembly and is helpful when you or others later need to modify it.

The Input step transforms the relational structure – the links and columns – into a hierarchical structure. In the hierarchical data structure, an input link is transformed into a list item, and each column on the link is transformed into a child content item of that list. The step Output schema displays the schema tree that represents this hierarchical data structure. This data structure is passed to the next step in the assembly, where it becomes the step Input. At each step, the step Input is modified by the action of the step. For example, if you use an Aggregate step to perform a calculation, the result of the calculation is added as a branch to the step Output. Additions to the step Output are highlighted in the schema tree.

The Output step receives the results of all of the previous step transformations. In the Output step, you map the hierarchical data structure back to a relational data structure. That is, you map schema items to output links and to the columns on those links.

Overview
The Overview of an assembly provides creation and modification information about the assembly and includes an optional description field that you can modify.

On the Overview, you can enter an optional description for the assembly. This field documents the purpose of the assembly and is helpful when you or others later need to modify the assembly.
Input step

The Input step describes how the links and columns of a relational data structure are represented as a hierarchical data structure.

The Configuration tab of the Input step provides two views: a Links view and a Tree view.

The Links view displays the same column definition table that you see in on the Columns tab of a stage. From the Links view, you can edit the column definitions and then view the result in the step Output. When you save the assembly, the edited column definitions are saved in the job properties.

The Tree view displays the relational links structure after it has been transformed into a hierarchical structure. The root element is Top, and it contains the entire input data. Each input link is transformed into a child list item of the InputLinks node. Each column is transformed into a content item and assigned a primitive type. For example, a column that is assigned the SQL Varchar type is transformed into an a content item that has the String data type.

When waves are used in a job that uses the XML stage, each wave is a new item in the Top list, and each item contains all of the data that belongs to a particular wave. When waves are not used, the Top list always contains a single item that contains all of the data that passed through the stage.

The step Output of the Input step becomes the step Input for the next step in the assembly. This process of taking the transformed data structure from the previous step; transforming, parsing, or composing data; and then producing a modified data structure for the next step continues through each step in the assembly. At each step, you can look at the step Input, which displays the result of the previous step, and the step Output, which displays the output results of the current step, to assess the changes that each step makes to the data structure.

Note that not all assemblies use an Input step. For example, if the XML stage is used as a source stage in a job, there are no input links and the Input step is empty. Even if the Input step is empty, it remains a permanent part of an assembly; you cannot remove it. Parallel jobs can receive input from multiple input links, while server jobs can only receive input from a single input link.

Output step

The Output step describes how the hierarchical data structure is mapped to a relational data structure.

The Configuration tab of the Output step provides two windows: Output and Mappings.

In the Output window, you can display two views: a Links view and a Tree view.

The Links view displays the same column definition table that you see on the Columns tab of a stage. From the Links view, you can edit the column definitions and then view the result in the step Output. When you save the assembly, the edited column definitions are saved in the job properties.

OK
The Tree view displays the relational links structure after it has been transformed into a hierarchical structure. The root element is Top, and it contains the entire output data. Each output link is transformed into a child list item of the OutputLinks node. Each column is transformed into a content item and assigned a primitive type. For example, a column that is assigned the SQL Varchar type is transformed into an a content item that has the String data type.

In the Mappings window, you map the hierarchical data structure to a relational data structure.

You must map a source item to every item that is in the target structure and that you want to include in the output. Start at the top of the target structure and map items from the top down. Start by mapping target list nodes to source lists nodes. As soon as you map a target list node, the target content nodes of that list are available for mapping. Map the target content nodes of each mapped list node to source content nodes.

In the Mappings window, you can delete multiple columns from the Mappings table by selecting the columns and clicking Delete button from the keyboard.

For more information about mapping, see Mapping data.

Not all assemblies use an Output step. For example, if the XML stage is used as a target stage in a job and doesn't have any output links, the Output step is empty. Even if the Output step is empty, it is a permanent part of an assembly; you cannot remove it.

**Assembly Editor**

Use the Assembly Editor to create an assembly.

Each XML stage that you add to a job requires an assembly. An assembly defines a series of steps that parse, compose, and transform hierarchical data.

The Assembly Editor has three panes: the Assembly Outline, the Step pane, and the step Input/step Output pane. The Assembly Outline lists the names of the assembly steps in the order in which they are performed.

To add a step to the assembly, you click the Palette button. Then you drag a step from the palette to the location in the assembly where you want that step to be performed or double-click a step to add it below the currently selected step in the assembly. Steps are available for a variety of transformations, including sorting, aggregating, joining, and pivoting.

When you click the name of a step in the Assembly Outline, the step displays in the Step pane. Each step has three tabs: Configuration, Information, and Test Data. The Configuration tab always contains mandatory fields that you must complete to configure the step. The step Information tab is similar to the assembly Overview; it displays creation and modification information, as well as optional fields that you can use to document the name and purpose of the step. You use the Test Data tab when you test the assembly. Testing is optional but extremely useful. By testing, you can see the incremental change that each step makes. You can evaluate how the step transformations affect the data and confirm that the assembly is performing exactly the transformation that you want.
As you configure each step in the assembly, the Assembly Editor monitors your work and maintains a running total of the issues that you need to address. An issue is not necessarily an error. In many cases, an issue describes a specific field that requires input or an item that requires mapping. To see all of the issues in the assembly, click View All. The issues display, sorted by step. For each step, a handy link is available, so that you can quickly jump to the step and start addressing the issues. The issues are also displayed on the Configuration tab of the step that has issues.

The step Input/step Output pane displays the step input and the step output as schema trees. The step Input is the result of the enrichments and transformations that were made by all of the previous steps in the assembly. The step Output describes the output of the current step. For example, if the step adds a new node to the data structure, the step Output highlights the new node. Within the assembly, the output of one step is always the input for the next step in the outline.

On the Output step, you can right-click on an item to display a menu of actions that you can perform on that step. You can rename items so that they have more meaningful names, delete unnecessary sub-trees, and perform other modifications.

**Opening the Assembly Editor**

After you add the XML stage to a job, use the Assembly Editor to create the assembly.

**Procedure**

1. From the Real Time section of the palette in the IBM InfoSphere DataStage and QualityStage Designer, drag to add an XML stage to a job.
2. Double-click the XML stage to open the stage editor.
3. Click Edit assembly to display the Assembly Editor.

   **Note:** The Edit assembly button provides the only way to open the Assembly Editor; there is no menu choice for the Assembly Editor.

**What to do next**

After you finish working on the assembly, close the Assembly Editor to return to the XML stage editor and to the job. If you plan to work for an extended period of time on an assembly, be sure to close the Assembly Editor occasionally, so that your changes are saved.

**Modifying items in the step Output**

Rename or drop an item, change a list item into a group item, or change a group item into a list item.

**About this task**

Each step in an assembly transforms or enriches the schema and then presents the result of all of the changes in the step Output. You can make minor modifications to items in the step Output. These modifications are passed to the next step in the assembly as part of the step Input.
Procedure

Select an item in the step Output, and then right-click to display a menu of choices that are appropriate for the selected item.

**Table 3. Item modifications**

<table>
<thead>
<tr>
<th>Choice</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revert</td>
<td>Removes all modifications from the selected item.</td>
</tr>
<tr>
<td>Rename</td>
<td>Changes the name of the item. The previous item name displays when you hover over the new item name.</td>
</tr>
<tr>
<td>Drop</td>
<td>Removes an item and all of its children.</td>
</tr>
<tr>
<td>Group-to-List</td>
<td>Changes a group item to a list item. Creates a list item that has the selected list item as the only element. This option is available only for an item that has children but that is not a list item. This option is useful when a list transformation such as a join or a union should be conducted on an optional group. When the group exists, it will have a list size of 1. When the group does not exist, it will be an empty list.</td>
</tr>
<tr>
<td>List-to-Group</td>
<td>Changes the list item to a group item by taking only the first element of the list. This option is available only for a list item. This action is useful when a list item contains only one child item, and you want to remove the complexity of dealing with the list. If the list contains more than one instance, however, only the first instance is selected and the rest of the instances are ignored.</td>
</tr>
</tbody>
</table>

Testing assemblies

As you work on an assembly, you can use randomly generated data or XML source data to test each step and view the output data.

About this task

Testing the assembly allows you to run the assembly in design mode without compiling and running the job. Unlike a job, which you can test only when it is complete and has no errors, you can test an assembly at any time during the design process. Even if the assembly is incomplete and has errors, you can test the steps in the assembly that do not have errors.

To test an assembly, you provide source data or use randomly generated data. Then after you run the test, you can view the output from each step and see the log messages that the stage would produce if it were run as part of the job.

You can test an assembly as soon as you add a step that is after the Input step and that does not contain errors. Testing an assembly processes the data in the Input step and in all of the subsequent steps. The test stops if it reaches a step that contains errors. Incrementally test an assembly to determine if the steps in the assembly are performing the transformations that you expect. If a step is not performing as you expect, you can modify that step and then retest the assembly. When you are sure that the assembly is working just as you want it to, you are ready to run the job.

To test the assembly, you must provide test input data to the XML Parser step. If the XML Parser step obtains its data from an upstream stage, you do not need to provide test data. For any other case, you must either generate random test data or provide actual source data to use in the test. If you do not provide input data, the test automatically uses randomly generated data.

Procedure

1. To provide input test data for the Input step, click the **Test Data** tab and then do one of the following:
In the **Input Links** field, select the link that provides the real source data to use.

Click **Generate Random Data** to produce sample data to use in the test.

2. To provide input test data for the XML Parser step, click the **Test Data** tab and then do one of the following:

   - In the **Disk Input Test Data** window, select the link that provides the real source data to use.
   - Click **Generate Random Data** to produce sample data to use in the test.

3. Above the **Test Data** tab, click **Test Assembly** and then perform the following steps:

   a. Optional: If you created job parameters for the job, enter the values for the parameters in the Test Values column of the Job Parameter Values table.

   b. Click **Run Test**.

   The **Test Execution Log** displays messages that describe the test.

4. Open each step that was included in the test. The **Downstream Output Test Data** shows the data that the step produces for the next step. The **Disk Output Test Data** shows the data that will be sent to an external resource.

### Parsing large schemas

To parse a schema that has over 2,000 nodes, you use configure multiple XML Parser steps.

When you configure an XML Parser step and select a schema that has over 2,000 nodes, the first 2,000 nodes are available to be parsed. However, the remaining nodes are automatically put into a chunk. To parse the chunk, you add a second XML Parser step to the assembly and select **String set** for the **XML Source**. Then select the chunk as the **String set** source item. On the **Document Root** tab, select the global element that corresponds to the chunked sub-tree. If the global element does not exist, modify the schema by creating global elements from in-line elements.

### Icons in the schema tree

The following table shows the icons that represent the types of items in the schema tree.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>All</td>
<td>A complex type. The child elements can appear 0 or 1 time, in any order, in the data.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Any simple type</td>
<td>A simple type. The type can contain the data for any concrete simple type.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Base-64 binary</td>
<td>A binary type. The type contains base-64-encoded data.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Boolean</td>
<td>A logical type. The legal values are true or 1 and false or 0.</td>
</tr>
<tr>
<td>Icon</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Byte</td>
<td>Numeric data that is in the range from -128 to 127.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Byte string</td>
<td>Unparsed binary data.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Choice</td>
<td>A complex type. Only one child can appear in the data. If any child is optional, the type can be empty.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Date</td>
<td>A date type that is in the format CCYY-MM-DD. A trailing time zone is optional.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Date time</td>
<td>A datetime type that is in the format CCYY-MM-DDThh:mm:ss.sss. A trailing time zone is optional.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Decimal</td>
<td>An arbitrary precision decimal number.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Double</td>
<td>Double-precision 64-bit floating-point number.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Duration</td>
<td>A time period that is in the format PnYNnMnDTnHnMnS, where n is one or more digits.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Empty</td>
<td>A simple type that does not contain any content.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Float</td>
<td>A single-precision 32-bit floating-point number.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Hex binary</td>
<td>A simple binary type that contains hexadecimal-encoded data.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>ID</td>
<td>A unique identifier.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>ID reference</td>
<td>A reference to a unique identifier.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Integer</td>
<td>Numeric data that is in the range from -2147483648 to 2147483647</td>
</tr>
<tr>
<td>![Icon]</td>
<td>List</td>
<td>An array of a simple type.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>List all</td>
<td>An array of the complex type all.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>List choice</td>
<td>An array of the complex type choice.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>List sequence</td>
<td>An array of the complex type sequence.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Locale</td>
<td>The locale.</td>
</tr>
<tr>
<td>Icon</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Long integer</td>
<td>Numeric data that is in the range from -9223372036854775808 to 9223372036854775807.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Name</td>
<td>A valid XML name. The name starts with a letter, underscore, or colon; and it can contain letters, underscores, colons, hyphens, and periods.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Namespaces</td>
<td>Absolute or relative URI references.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>NCname</td>
<td>A non-colonized name. The name starts with a letter or an underscore; and it can contain letters, underscores, colons, hyphens, and periods.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>NMtoken</td>
<td>A single token that does not contain any whitespace.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>NMtokens</td>
<td>Multiple tokens that are separated by whitespace.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>QName</td>
<td>A simple type for a qualified name. Qualification is optional.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Recursive XML</td>
<td>Recursive, unparsed XML data.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Sequence</td>
<td>A complex type. The child elements must appear in the specified order in the data.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Short integer</td>
<td>Numeric data that is in the range from -32768 to 32767.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>String</td>
<td>A string that has unmodified whitespace.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Time</td>
<td>A time type that is in the format hh:mm:ss.sss. A trailing timezone is optional.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Token</td>
<td>A string in which all whitespace is replaced by a space and multiple spaces are replaced by a single space.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Typeschema</td>
<td>A graphical view of the schema.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Unknown</td>
<td>A new or unknown type.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>URI</td>
<td>An absolute or relative URI reference.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Unparsed XML</td>
<td>Unparsed XML data.</td>
</tr>
</tbody>
</table>
The following table shows the icons that indicate the state of specific items in the schema tree.

**Table 5. Icons that indicate the state of items in the schema tree**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Checkmark]</td>
<td>Indicates the selected item.</td>
</tr>
<tr>
<td>![ Rename ]</td>
<td>Indicates that the item was renamed.</td>
</tr>
<tr>
<td>![ Remove ]</td>
<td>Indicates that the item was removed.</td>
</tr>
<tr>
<td>![ Multiple Actions ]</td>
<td>Indicates that multiple actions were performed on the item – for example, renaming it and changing it from a group to a list.</td>
</tr>
<tr>
<td>![ Group to List ]</td>
<td>Indicates that the item was changed from a group to a list or from a list to a group.</td>
</tr>
<tr>
<td>![ Enrichment Root ]</td>
<td>Indicates the enrichment root item – that is, the addition to the schema that occurs as a result of the current step.</td>
</tr>
<tr>
<td>![ Rename Enrichment Root ]</td>
<td>Indicates that the enrichment root item was renamed.</td>
</tr>
<tr>
<td>![ Remove Enrichment Root ]</td>
<td>Indicates that the enrichment root item was removed.</td>
</tr>
<tr>
<td>![ Multiple Actions Enrichment Root ]</td>
<td>Indicates that multiple actions were performed on the enrichment root item.</td>
</tr>
<tr>
<td>![ Group to List Enrichment Root ]</td>
<td>Indicates that the enrichment root item was changed from a group to a list or from a list to a group.</td>
</tr>
</tbody>
</table>

**Details Inspector**

The Details Inspector displays information about the attributes and facets of a selected item and about the definition of DataStage columns.

**Table 6. Attributes in the Details Inspector**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the item that was selected for inspection.</td>
</tr>
<tr>
<td>Type</td>
<td>The XML type of the item.</td>
</tr>
<tr>
<td>Type Name</td>
<td>The base type of the item.</td>
</tr>
<tr>
<td>Optional</td>
<td>Indicates if the item is optional. For example, in XML, an item is optional when it has minOccurs=“0” or when it has the attribute xs:use=“optional”.</td>
</tr>
<tr>
<td>List</td>
<td>Indicates if the item is an array.</td>
</tr>
<tr>
<td>Recursive</td>
<td>Indicates if the item is recursive.</td>
</tr>
<tr>
<td>TypeName Namespace URI</td>
<td>The namespace URI of the type.</td>
</tr>
<tr>
<td>Item Namespace URI</td>
<td>The namespace URI of the item.</td>
</tr>
<tr>
<td>Type Documentation Annotation</td>
<td>An annotation that is defined in the XML schema for the type or an explanation of a created type.</td>
</tr>
</tbody>
</table>
Table 6. Attributes in the Details Inspector (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Documentation Annotation</td>
<td>An annotation that is defined in the XML schema for the item.</td>
</tr>
<tr>
<td>Nillable</td>
<td>Indicates if the item can have nil as its value.</td>
</tr>
<tr>
<td>Max Occurs</td>
<td>The maximum times that the item can occur.</td>
</tr>
<tr>
<td>Min Occurs</td>
<td>The minimum times that the item can occur.</td>
</tr>
</tbody>
</table>

Table 7. Facets in the Details Inspector

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whitespace</td>
<td>Indicates how whitespace is handled: preserve all whitespace, replace each whitespace with a space, or replace all whitespace with a single space.</td>
</tr>
<tr>
<td>Fraction Digits</td>
<td>The maximum number of fractional digits.</td>
</tr>
<tr>
<td>Total Digits</td>
<td>The maximum number of significant digits.</td>
</tr>
<tr>
<td>Max Length</td>
<td>The length of the value must be less than or equal to maxLength.</td>
</tr>
<tr>
<td>Min Length</td>
<td>The length of the value must be greater than or equal to minLength.</td>
</tr>
<tr>
<td>Max Exclusive</td>
<td>The value must be less than maxExclusive.</td>
</tr>
<tr>
<td>Min Exclusive</td>
<td>The value must be greater than minExclusive.</td>
</tr>
<tr>
<td>Max Inclusive</td>
<td>The value must be less than or equal to maxInclusive.</td>
</tr>
<tr>
<td>Min Inclusive</td>
<td>The value must be greater than or equal to minInclusive.</td>
</tr>
<tr>
<td>Enumeration Value</td>
<td>The value must match one of the enumerated values.</td>
</tr>
<tr>
<td>Length</td>
<td>The length of the item.</td>
</tr>
<tr>
<td>Pattern</td>
<td>The pattern that is defined for the item.</td>
</tr>
</tbody>
</table>

When you select an item in the Output schema of the Input step or when you select an item in the Input schema of the Output step, the Details Inspector displays information about the selected DataStage column.

Table 8. DataStage column definition information in the Details Inspector

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Indicates if the item selected represents a link or a column.</td>
</tr>
<tr>
<td>SQL Type</td>
<td>The SQL type that is defined for the column.</td>
</tr>
<tr>
<td>Key Position</td>
<td>Indicates if the column is defined as a key.</td>
</tr>
<tr>
<td>Extended</td>
<td>The extended type of the column.</td>
</tr>
<tr>
<td>Precision</td>
<td>The precision value defined in the column.</td>
</tr>
<tr>
<td>Scale</td>
<td>The scale value defined in the column.</td>
</tr>
<tr>
<td>Nullable</td>
<td>Indicates if the column value can be null.</td>
</tr>
</tbody>
</table>
Table 8. DataStage column definition information in the Details Inspector (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level Number</td>
<td>The level number defined in the column.</td>
</tr>
<tr>
<td>Display Size</td>
<td>The display size defined in the column.</td>
</tr>
<tr>
<td>Data Element</td>
<td>The data item defined in the column.</td>
</tr>
<tr>
<td>Description</td>
<td>The description defined in the column.</td>
</tr>
</tbody>
</table>

Mapping data

Understanding how to map source data to target data is key to obtaining the results that you want when you transform hierarchical data.

Working with the mapping table

Some steps, such as the Output step and the XML Composer step, require that you create mappings that define how to create the target nodes.

For these steps, the Mappings tab displays the mapping table. This table contains three columns: Source, Result, and Target. The Target column displays the elements in the target structure. The Source column contains the mapping for each mapped element in the target structure. The Result column displays the status of the current mapping. The table contains one row for every level in the target data structure. While every level is shown, you only map target list nodes to source list nodes and target content nodes to source content nodes. The mapping table does not allow you to map a list node to a content node or to map a content node to a list node.

Content items have a primitive value and correspond to actual values. List items repeat in the data. Elements that have maxOccurs>1, xs:list items, and links are all represented by list items.

You must map a source item to every required item in the target structure. Start at the top of the target structure and map items from the top down. Start by mapping target list nodes to source lists nodes. As soon as you map a target list node, the target content nodes of that list are available for mapping. Map the target content nodes of each mapped list node to source content nodes.

A list mapping defines how instances in the target list are created. For each item in the source list, a target item is created, and the content mapping is computed in the context of that source item. The context is defined as all of the parent items that contain the item.

For each target node, the mapping table provides a list of valid mapping candidates. The valid candidates are evaluated based on the target position in the target tree, its parent mappings, and its data type. The valid candidates are then scored and ordered based on name similarity to the target node and path.

There are four ways to map an item. First, you select the row in the mapping table, and then perform one of the following steps:

1. Click the down-arrow in the Source column to display a partial list of valid mapping candidates. Then select an item from that list.
2. Click **Auto Map** to automatically map the source item that received the highest mapping score. Automatic mapping is context-sensitive. If you select a list item and then click **Auto Map**, the list item and all of its descendent items are automatically mapped.

3. Click the down-arrow in the source column, and then click **More** to display the entire source structure as tree. In the tree, you can select any node, even if it is not a valid mapping candidate. If you select an invalid node, a message displays to describe why the mapping is invalid.

   To see the source structure as a list, click **Switch to List View**. Then select an item to map. The list view displays a **Target Similarity** button. To control how many mapping candidates display, select a target score. Only the candidates that have a score higher than the one that you select display in the list.

4. To assign a fixed value to the target content item, click the down-arrow in the source column, click **Constant**, and then enter a fixed value to map to the target item. Note that the **Constant** option is not available for list items.

   As you perform the mappings on the Output step, you might determine that the target contains one or more items that you do not need. To remove an item from the target structure, select the item, and then click the delete icon in the rightmost column of the table. This action is similar to removing columns from the output link.

   If you decide to change a mapping, select the item and then choose a different mapping candidate.

   If you use any method other than automatic mapping to map an item, a Lock icon displays in the first column of the table to indicate that the mapping is locked.

   If you decide that you want to remove one or more mappings, select an item and click **Clear Mappings**.

**Related reference:**

“JSON Composer step” on page 43

Use a JSON schema or a view created from the JSON schema to compose JSON data.

**Determining mapping candidates**

For each item in the target structure, the mapping table automatically presents a list of valid mapping candidates.

To determine the candidates to display in the mapping candidate list and to determine which candidate to use for the **Auto Map** option, the mapping algorithm uses the following process:

1. Create a list that contains all of the valid mapping candidates. Valid candidates are unambiguous.

2. Select only the candidates that meet the specified level of conversion accuracy. By default, the level for conversion accuracy is lossy; candidates must have a data type that can be converted to the target data type without creating a possible runtime error.

3. Evaluate the similarity of the name and of the data type of the source and target items. Assign a score to each candidate, based on the similarity of the names and data types.

4. Select the highest scoring candidate to use for the **Auto map** option.
5. Select the highest scoring candidates to present on the mapping candidate list, which displays when you press the down-arrow on the source item in the mapping table. By default, the mapping candidate list displays the top 5 candidates.

To change the level of conversion accuracy and the number of mapping candidates to present on the candidate list, choose Administration > Mapping Configuration.

**Mapping example**

To understand why certain mapping candidates are valid or invalid, review this example. The following figure shows the step Input. The step Input contains the following items that are peers of each other:

- An item named documentID.
- A list named mySourceList, which contains an item named sourceChildField and a list item named sourceChildList, which contains an item named sourceDescendentField.
- A list named Peer-List, which contains an item named peerField.

The following figure shows the instance data that the step receives and transforms to the target structure:
The following is the XML data that this example uses. As you read about valid and invalid mapping candidates, refer to this data so that you can understand how the data values are determined:

```xml
<SourceDoc>
  <documentID>ID001</documentID>
  <mySourceList>
    <sourceChildField>A1</sourceChildField>
    <sourceChildList>
      <sourceDescendentField>A11</sourceDescendentField>
    </sourceChildList>
    <sourceChildList>
      <sourceDescendentField>A12</sourceDescendentField>
    </sourceChildList>
  </mySourceList>
  <mySourceList>
    <sourceChildField>A2</sourceChildField>
    <sourceChildList>
      <sourceDescendentField>A21</sourceDescendentField>
      <sourceDescendentField>A22</sourceDescendentField>
    </sourceChildList>
  </mySourceList>
  <Peer-List>
    <peerField>peer1</peerField>
  </Peer-List>
  <Peer-List>
    <peerField>peer2</peerField>
  </Peer-List>
</SourceDoc>
```
The following figure of the mapping table shows that list item mySourceList is mapped to list item myTargetList, and all of the child items of myTargetList are correctly mapped.

<table>
<thead>
<tr>
<th>Source</th>
<th>Result</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td></td>
<td></td>
</tr>
<tr>
<td>top/..result/..SourceDoc/mySourceList</td>
<td></td>
<td>myTargetList</td>
</tr>
<tr>
<td>top/..result/..SourceDoc/mySourceList/sourceChildField</td>
<td>targetDocumentID</td>
<td></td>
</tr>
<tr>
<td>top/..result/..SourceDoc/mySourceList/targetChildList</td>
<td>targetChildList</td>
<td></td>
</tr>
<tr>
<td>top/..result/..SourceDoc/mySourceList/targetDescendentField</td>
<td>targetDescendentField</td>
<td></td>
</tr>
</tbody>
</table>

**Valid mappings**

List mapping determines how the items of a target list are created. During item creation, the source list is iterated; and for each item in the source list, a new item is created in the target list. A child list is similarly iterated in a nested loop. Content mappings are evaluated in the context of the iteration. In other words, the target value of an item depends on the source list item that is currently being iterated and on the mapping.

The following mappings are valid for the example:

**List mapping**

For each item in mySourceList, one item in myTargetList is created. Therefore, two myTargetList items are created for the two mySourceList items that are in the input data.

**Simple content mapping**

To create the first myTargetList item, the mapping top/result/SourceDoc/mySourceList/sourceChildField evaluates to A1. Therefore, the value of targetField is A1. To create the second myTargetList item, the mapping top/result/SourceDoc/mySourceList/sourceChildField evaluates to A2. Therefore, the value of targetField is A2.

**Ancestor content mapping**

To create the first myTargetList item, the mapping top/result/SourceDoc/documentID evaluates to ID001. When looking for the value of documentID, the containment relationship is taken into account. Because mySourceList is contained within a single document that has only a single documentID value, the mapping is unambiguous; and the value of targetDocumentID is ID001. The second myTargetList item has the same containment relationship. Therefore, the mapping top/result/SourceDoc/documentID evaluates to ID001. Consequently, both target items have the same documentID value.

**Child list mapping**

Valid mappings always preserve containment relationships. Therefore, targetChildList must be mapped to a descendent list of mySourceList. Each child list has two items because the sourceChildList has two items in the input data.

To modify or create a containment relationship, add a transformation step, such as an HJoin step or a Union step, before the step that contains the mapping table.
Invalid mappings

The mapping table prevents you from creating ambiguous, invalid mappings. An ambiguous mapping occurs when you map an item that, when evaluated, returns multiple values. The following mappings are invalid:

Mapping the content child of a peer list to a child content item
When iterating on the first mySourceList and trying to evaluate the mapping top/result/SourceDoc/Peer-List/peerField, the two values peer1 and peer2 are returned. The containment relationship for the mySourceList item does not help to resolve which value should be returned. Therefore, the mapping is ambiguous and is not allowed.

Mapping content child of a descendent list to a child content item
When iterating on the first mySourceList and trying to evaluate the mapping top/result/SourceDoc/sourceChildList/sourceDescendentField, the two values A11 and A12 are returned. The containment relationship for the mySourceList item does not help to solve which value to return because both values are contained in the mySourceList. Therefore, this mapping is ambiguous and it not allowed.

Mapping peer list to a child list
List mapping can be ambiguous. When iterating on the first mySourceList and trying to evaluate the mapping top/result/SourceDoc/Peer-List, the mapping cannot determine whether to return the first or the second
Peer-List item. Without a containment relationship between the items of Peer-List and mySourceList, the ambiguity cannot be resolved. Therefore, this mapping is invalid.

The following figures show the valid choices:

**Configuring how mapping candidates are determined**

Configure how mapping candidates are calculated and displayed.

**Procedure**

1. Click **Administration** and then select **Mapping Configuration**.
2. Perform one or more of the following steps:
In the **Strings to Use as Keys** field, enter one or more strings to use to recognize good key items. Items with names that contain one of the common key identifiers are scored higher than other items and presented as choices in key fields.

In the **Data Type Conversion Accuracy** field, select the default conversion accuracy to use to select the mapping candidates that display in the Source column of the Mappings table. The choices are:

- **Same type**
  Only source items that have the same type of the target item are presented as mapping candidates.

- **Lossless**
  Only source items that have a type that can be converted without loss of data to the target item type are presented as mapping candidates.

- **Lossy** (Default) Only source items that have a type that can be converted to the target item type without causing a possible runtime error are presented as mapping candidates.

- Select **Map optional lists** to automatically map lists that are optional.
- Select **Map optional primitive data types** to automatically map primitive data types that are optional.

- In the **Number of mapping candidates to display** field, select the number results to display as mapping candidates.

---

### XML Parser step

Use the XML Parser step to parse one or more documents that have the same structure.

### XML Source

For the source of the XML data, specify one of the following:

- **String set**
  Select the input schema item that contains the document string. Only items that have the String, normalizedString, byteString, or XML data types are available for selection.

- **Single file**
  Enter the path and file name, or click **Insert Parameter** and then select the name of the parameter. The parameters that are available are those that you previously defined in the job and the built-in macros that are in IBM InfoSphere DataStage. Only items that have the String, normalizedString, or byteString data types are available for selection.

- **File set**
  A file set option is used to read multiple xml files that are based on the same xsd. Select the input schema item that will contain in runtime the absolute paths (example, c:\test.xml) of the xml files. Only items that have the String, normalizedString, or byteString data types are available for selection.

### Parallel Parsing

Open the section, and then click **Compute** to analyze the job and the input XML document and then display the XML path to the body of the
document that will be parsed. For complete information about configuring parallel parsing, see Configuring Parallel Parsing.

Enable Filtering

Enable filtering to apply an XSLT style sheet to the document before parsing it. The document root must reflect the document that is created from the result of the XSLT transformation. This option is not recommended for large documents because the entire processing is done in memory. For large documents, use transformation steps. This feature cannot be used with the parallel parsing feature.

Document Root

Select the top-level element that describes the documents that you are parsing. The types that display under the library’s namespace are top-level element definitions. Following the XML Schema standard, only top-level elements can describe documents. The name of the element that you select must match the top-level element name in the instance documents. For example, if you are parsing Order documents, you select the Order element. When you select the element, you can view its structure and verify that the structure is correct for the documents that you want to parse.

Note: The elements from which you select the document root are from the resources that were previously imported into the schema libraries. If you need to import the resource that contains the document root for the XML Parser step, click the Libraries tab and import the resource that you need. Then return to the Assembly Editor and configure the document root.

Validation

By default, when the XML Parser step runs, it uses minimal validation, which disables all of the validation rules and provides better performance than strict validation does. Strict validation is initially configured so that each validation rule is set to Fatal, and the job stops as soon as it parses the first occurrence of invalid data. To customize validation, specify the action to perform when a violation occurs.

For more information about validation rules, see “XML Parser validation rules.”

For an example that includes the XML Parser step, see Example 1.

XML Parser validation rules

By default, the XML Parser uses minimal validation and ignores violations. To customize validation, specify the action to perform when a violation occurs.

The following tables describe the validation rules and the applicable actions.
<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type has illegal value</td>
<td>The value must match the value rules for the data type of the corresponding item.</td>
<td><strong>Ignore</strong> Type-checking and type-conversion are not performed. The types of the items as they are defined by the schema are replaced by the String type. For example, if the schema includes an item named startDate that is of type Date, the type of that item in the Step Output becomes String, not Date.</td>
</tr>
<tr>
<td>Log per Occurrence; Log per Document</td>
<td>Type-checking is performed. If a type fails the check, a log error is reported either once for each occurrence or once for the entire document; however, parsing continues. Type-conversion is not performed. The real types of the items in the Step Output are replaced by string types.</td>
<td><strong>Reject</strong> Type-checking and type-conversion are performed. Only invalid values fail the invalid document parsing. The ParsingStatus item is added as a new group to the Step Output for the XML Parser step and is set to False and contains the corresponding error message.</td>
</tr>
<tr>
<td>Log per Document</td>
<td></td>
<td><strong>Fatal</strong> Type-checking and type-conversion are performed. An invalid value causes the job to fail.</td>
</tr>
<tr>
<td>Convert to Default</td>
<td></td>
<td><strong>Convert to Default</strong> Type-checking and type-conversion are performed. An invalid value is replaced with its default value.</td>
</tr>
<tr>
<td>Rule</td>
<td>Description</td>
<td>Actions</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Value fails facet constraint</td>
<td>The value is checked against the facets of its corresponding item type.</td>
<td><strong>Ignore</strong> Facet-checking is not performed.</td>
</tr>
<tr>
<td></td>
<td>Note: If you enable this rule, the <strong>Datatype has illegal value</strong> rule is also set with the same value that is chosen for Value fails facet constraint rule.</td>
<td><strong>Log per Occurrence; Log per Document</strong> Facet-checking is performed. Log errors are issued for the violations, and parsing continues.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Reject</strong> Facet-checking is performed. Invalid values will fail only the invalid document parsing, and the ParsingStatus item, which is added as a new group to the step Output for the XML Parser step, is set to False with the corresponding error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Fatal</strong> Facet-checking is performed. An invalid value causes the job to fail.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Convert to Default</strong> Facet-checking is performed. An invalid value is replaced with its default value.</td>
</tr>
<tr>
<td>Rule</td>
<td>Description</td>
<td>Actions</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------------------------------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>Item than cannot be null has a null value</td>
<td>Non-nullable items are checked to ensure that they do not contain null values.</td>
<td><strong>Ignore</strong> Null-checking is not performed. All items in the step Output for the XML Parser step become nullable. All items in the step output become nullable. <strong>Log per Occurrence; Log per Document</strong> Null-checking is performed, and log errors are issued. Null values are set. As a result, all items in the step Output for the XML Parser step become nullable. <strong>Reject</strong> Null-checking is performed. Invalid values will fail only the invalid document parsing, and the ParsingStatus item, which is added as a new group to the step Output for the XML Parser step, is set to False and includes the corresponding error message. <strong>Fatal</strong> Null-checking is performed. An invalid value causes the job to fail. <strong>Convert to Default</strong> Null-checking is performed. An invalid value is replaced with its default value.</td>
</tr>
<tr>
<td>Use global default values for missing values</td>
<td>Missing values are replaced with their default values.</td>
<td><strong>False</strong> If the schema defines a default value for the item, that default value is used. Otherwise, the value is set to nil. <strong>True</strong> If the schema defines a default value for the item, that default value is used. Otherwise, the value is set to the default value that is specified in the Assembly Administration panel.</td>
</tr>
<tr>
<td>Trim values</td>
<td>Trim the white space before and after a value before performing type-checking and type-conversion.</td>
<td><strong>False</strong> Trimming is not performed. <strong>True</strong> Trimming is performed on both sides of the value.</td>
</tr>
<tr>
<td>Rule</td>
<td>Description</td>
<td>Actions</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Document is malformed</td>
<td>The document is malformed.</td>
<td>Ignore: No error message is logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Document: An error message is logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reject: The ParsingStatus item, which is added as a new group to the step Output for the XML Parser step, is set to False and includes the corresponding error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal: A malformed document causes the job to fail.</td>
</tr>
<tr>
<td>Items are not declared in the schema</td>
<td>Items in the instance document must be declared in the schema. Violations might occur when a job uses an out-of-date version of a schema.</td>
<td>Ignore: No error message is logged. Items are parsed based on best guess.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log per Document: Error messages are logged. Items are parsed based on best guess.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reject: The ParsingStatus item, which is added as a new group to the step Output for the XML Parser step, is set to False and includes the corresponding error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal: An item that is not declared in the schema causes the job to fail.</td>
</tr>
<tr>
<td>Nil element has a value</td>
<td>Nullable items are checked to ensure that they do not contain data values. This applies only to those items where nullable is set to true in the xml document.</td>
<td>Ignore: No error message is logged. Items are parsed based on best guess.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log per Document: Error messages are logged. Items are parsed based on best guess.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reject: The ParsingStatus item, which is added as a new group to the step Output for the XML Parser step, is set to False and includes the corresponding error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal: A null element having a value causes the job to fail.</td>
</tr>
<tr>
<td>Rule</td>
<td>Description</td>
<td>Actions</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Mandatory items are missing</td>
<td>Items must appear correctly in the instance document. Violations might occur if a required attribute is missing or if an element appears out of order in a sequence content.</td>
<td>Ignore Checking for mandatory items does not occur. No error message is logged. As a result, all items in the step Output become optional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error messages are logged. Parsing continues, even though data is missing. As a result, all items in the step Output become optional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reject The ParsingStatus item, which is added as a new group to the step Output for the XML Parser step, is set to False and includes the corresponding error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal A missing mandatory item causes the job to fail.</td>
</tr>
<tr>
<td>List has invalid number of occurrences</td>
<td>The number of occurrences must be between the value of the MinOccurs attribute and the MaxOccurs attribute, as defined in the schema.</td>
<td>Ignore No error message is logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error messages are logged. Parsing continues, even though data is missing. As a result, all items in the step Output become optional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reject The ParsingStatus item, which is added as a new group to the step Output for the XML Parser step, is set to False and includes the corresponding error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal If the list has invalid number of occurrences, the job fails.</td>
</tr>
<tr>
<td>ID values are not unique</td>
<td>Each document must have a unique ID.</td>
<td>Ignore No error message is logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error messages are logged. Parsing continues, even though data is missing. As a result, all items in the step Output become optional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reject The ParsingStatus item, which is added as a new group to the step Output for the XML Parser step, is set to False and includes the corresponding error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal Duplicate ID values cause the job to fail.</td>
</tr>
</tbody>
</table>
Controlling how specific items are parsed

Pass data as a chunk, disable type derivation, or change a recursive item into a list item.

About this task

By default, the XML Parser step applies the schema that you specify to all items in the schema. However, there might be situations when you do not want to apply the schema to a specific item. To control how a specific item is parsed, you apply one of the parsing options: Chunk, Disable Type Derivation, or Recursive to List.

To prevent data from being parsed, use the Chunk option. When you apply this option to a schema item, all of the data for the item and its child items, if there are any, is concatenated into a single string; and that string is passed as a chunk to the next step or stage. The Chunk option is useful when the data that you want to parse is inconsistent with the schema. In this case, you can pass the data as a chunk and in a subsequent step or stage, parse the chunk. Another use for the Chunk option is when you want to apply validation rules to only a specific section of a document. In this case, you pass the data as a chunk, and then in a subsequent XML Parser step, you configure validation rules that apply specifically to that chunk.

To prevent the schema tree from becoming unnecessarily complex and to optimize processing, use the Disable Type Derivation option. An XML schema can contain derivations. For example, a derivation occurs when the schema defines a base element and also defines a child element that inherits, or derives, attributes from the base element. By default, when the XML Parser step creates the schema tree, the tree includes all of the attributes that all of the child elements inherit from the base elements. This schema enrichment ensures that no information about the element is lost. If you know that you do not need the child elements in the data, you can select the base element in the schema and disable type derivation.

To turn a recursive type into a list of items, use the Recursive to List option. The attributes recursiveParentReferenceID and recursiveID are added to the schema to maintain the relationship via unique keys that the parser generates. The recursiveID is a unique key for each parent of the recursion. The recursiveParentReferenceID attribute stores the key for the parent of each child.

Procedure

1. In the Document Root window of the XML Parser step, right-click the item to which you want to apply a parsing option.
2. From the menu that displays, select the parsing option.

Setting default values for types

Specify a default value for the XML Parser to use when the schema does not define a default value for a type.

Procedure

1. From the Assembly Editor, click Administration and then select Default Data Type Values.
2. Enter default values.
3. On the XML Parser step, open the Validation window and set the Use global default values for missing values rule to True.
**XML Composer step**

Use a specified structure to compose XML content.

**XML Target**

**Write to File**

Enter the output directory and the file name prefix for the files that will be composed, or click Insert Parameter and then select the name of the parameter for the output directory and file name prefix. The parameters that are available are those that you previously defined in the job and the built-in macros that are in IBM InfoSphere DataStage.

**Pass as String**

Pass the composed XML string to a downstream step or stage for further processing.

**Pass as Large Object**

Pass the composed XML string as a large object. The final target stage – that is the last stage in the job – must be a LOB-aware stage, such as the DB2 connector, Oracle connector, ODBC connector, Teradata connector, or Websphere MQ connector. These stages use the LOB locator string to obtain the XML data and then write it to the target database or message queue. The job can contain non-LOB-aware stages, but these stages must not modify the LOB locator string. If the last stage in the job is not LOB-aware, the LOB locator is written out as data, rather than being interpreted as a locator.

**Document Root**

Select the top-level element that describes the documents that you are composing. The types that display under the library’s namespace are top-level element definitions. Following the XML Schema standard, only top-level elements can describe documents. The name of the element that you select must match the top-level element name in the instance documents. For example, if you are composing Order documents, you select the Order element. When you select the element, you can view its structure and verify that the structure is correct for the documents that you want to compose.

**Note:** The elements from which you select the document root are from the resources that were previously imported into the schema libraries. If you need to import the resource that contains the document root for the XML Composer step, click the Libraries tab and import the resource that you need. Then return to the Assembly Editor and configure the document root.

**Validation**

By default, the XML Composer uses strict validation, and the job fails if a violation occurs. To customize validation, specify the action to perform when a violation occurs.

For more information about validation rules, see "XML Composer validation rules" on page 34.
Mappings

Create a mapping to the document_collection item. How you map this item determines whether one document or multiple documents are created. To produce only one document, map the root of the Input (top) to document_collection. To produce multiple documents, map a list item to the document_collection item. Then one file will be created for each item in the list. For more information about mapping, see “Working with the mapping table” on page 18.

Header

Specify additional optional information to include at the beginning of the XML output.

Generate XML fragment
Do not include the XML declaration, comments, and processing instructions.

Include XML declaration
Include the XML declaration, for example, <?xml version="1.0" encoding="UTF-8">.

Include comments
Include the comments that you enter in the Comments field.

Include processing instructions
Include the processing instructions that you enter in the Processing Instructions field. Enclose each processing instruction in the <? and ?> tags.

Format

Encoding type
Select the encoding to use for the document. The default encoding is UTF-8.

Format style
Check the box to apply the following format options to the XML output:

- Indentation length – Select the number of characters to use for each indentation level in the XML output.
- New line style – Select the type of new line. Choices are UNIX (LF), DOS(CRLF) or MAC(CR).
- Omit attributes – For attributes that have default values and are in the data or for attributes that have fixed values, omit the values from the XML output.
- Omit null elements – For nullable elements that are not in the data, omit the elements from the XML output.

For examples the use the XML Composer step, see Example 2 and Example 3.

XML Composer validation rules
By default, the XML Composer uses strict validation, and the job fails if a violation occurs. To customize validation, specify the action to perform when a violation occurs.

The following tables describe the validation rules and the applicable actions.
<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type has illegal value</td>
<td>The value must match the value rules for the data type of the corresponding item.</td>
<td>Ignore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type-checking and type-conversion are not performed. The data types of the items in the Target column of the mapping table are replaced with the String data type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type-checking and type-conversion are performed. An invalid value causes the job to fail.</td>
</tr>
<tr>
<td>Value fails facet constraint</td>
<td>The value is checked against the facets of its corresponding item type.</td>
<td>Ignore</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you enable this rule, the “Datatype has illegal value” rule is also set to the same action that you choose for this rule.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** If you enable this rule, the “Datatype has illegal value” rule is also set to the same action that you choose for this rule.
<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item than cannot be null has a null value</td>
<td>Non-nullable items are checked to ensure that they do not contain null values.</td>
<td>Ignore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal (Default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trim values</td>
<td>Trim the white space before and after a value before performing type-checking and type-conversion.</td>
<td>False Trimming is not performed.</td>
</tr>
</tbody>
</table>
Table 12. Structure validation rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory items are missing</td>
<td>Items must appear correctly in the instance document. Violations might occur if a required attribute is missing or if an element appears out of order in a sequence content.</td>
<td>Ignore</td>
</tr>
<tr>
<td></td>
<td>Violations might occur if a required attribute is missing or if an element appears out of order in a sequence content.</td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td></td>
<td>Error messages are not logged. As a result, all items in Target column of the mapping table are optional and you do not need to provide a mapping for them.</td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td>Log per Occurrence; Log per Document</td>
<td>Error messages are logged. Missing values are not filled in. As a result, all items in the Target column of the mapping table are optional.</td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td>Fatal</td>
<td>A missing mandatory item causes the job to fail.</td>
<td>Fatal</td>
</tr>
<tr>
<td>List has invalid number of occurrences</td>
<td>The number of occurrences must be between the value of the MinOccurs attribute and the MaxOccurs attribute, as defined in the schema.</td>
<td>Ignore</td>
</tr>
<tr>
<td></td>
<td>No error message is logged.</td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td></td>
<td>Error messages are logged.</td>
<td>Log per Occurrence; Log per Document</td>
</tr>
<tr>
<td>Reject</td>
<td>The ComposingStatus item, which is added as a new group to the step Output for the XML Composer step, is set to False and includes the corresponding error message.</td>
<td>Fatal</td>
</tr>
<tr>
<td></td>
<td>(Default) If the list has invalid number of occurrences, the job fails.</td>
<td></td>
</tr>
</tbody>
</table>

**JSON transformation**

Use the XML stage to create powerful hierarchical transformations, as well as parse and compose JSON data with high performance and scalability.
**Schema management**

Before you can use the XML stage to parse or compose JSON data, you must import a JSON data instance that describes the data into the metadata repository. Use the Schema Library Manager to import JSON files into the metadata repository and to organize them into libraries that can be shared across all InfoSphere DataStage projects.

After you import a JSON data instance, the schema library manager automatically validates the instance to determine if the data contains any errors. If the data instance is valid, the schema library manager creates a schema in the metadata repository to describe the data types and structures in the instance. Click **Browse** to open the library to view the imported schema and to browse the type structure of the schema. You use these types to define the JSON processing in the XML stage.

When you create a library, you specify a unique name for the library and an optional category name. Library names must be unique across all categories. Organizing libraries into categories ensures that you can later locate a specific library.

Within a library, you cannot repeat the same global type definition; however, two different libraries can have the same type definition. Each library can be used to import only one JSON data instance.

You can create views on top of a generated JSON schema by selecting or de-selecting elements and performing chunking operations.

**Opening the Schema Library Manager**

You use the Schema Library Manager to import the schemas to use in jobs that include the XML stage.

**About this task**

You can open the Schema Library Manager from IBM InfoSphere DataStage and QualityStage Designer or from the Libraries tab, which is available from the Assembly Editor.

**Procedure**

Do one of the following:

- From the InfoSphere DataStage and QualityStage Designer, choose Import > Schema Library Manager.
- From the Assembly Editor, click the Libraries tab.

**Working with libraries and resources**

Use the Schema Library Manager to import resources, such as JSON files, and to create and manage libraries of resources.
Use the istool command line to transfer contract libraries between metadata repositories of IBM InfoSphere Information Server. Libraries are stored in the metadata repository and are available for use in any job that uses the XML stage. In the metadata repository, a contract library is represented by the library name, followed by the extension .cl. When you work with multiple installations of InfoSphere Information Server, you might want to transfer a contract library from one installation to another installation, for example, from a test environment to a production environment.

You can import and export contract libraries using the istool functionality. For more information, see . For more information about exporting and importing common metadata assets by using the command line, see [http://publib.boulder.ibm.com/infocenter/iisinfsv/v8r7/topic/com.ibm.swg.im.iis.iisinfsv.assetint.doc/topics/pdrassets.html](http://publib.boulder.ibm.com/infocenter/iisinfsv/v8r7/topic/com.ibm.swg.im.iis.iisinfsv.assetint.doc/topics/pdrassets.html).

**Table 13. Working with libraries**

<table>
<thead>
<tr>
<th>Task</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating libraries</td>
<td>1. Click <strong>New Library</strong>.</td>
</tr>
<tr>
<td></td>
<td>2. Enter a name.</td>
</tr>
<tr>
<td></td>
<td>3. Enter a description and the name of a new or existing category.</td>
</tr>
<tr>
<td>Refreshing library lists</td>
<td>Click the <strong>Refresh</strong> icon at the top of the Libraries pane.</td>
</tr>
<tr>
<td>Removing libraries</td>
<td>Select a library, and click <strong>Remove</strong>.</td>
</tr>
<tr>
<td>Opening libraries</td>
<td>Select a library, and click <strong>Open</strong>. You can open a library only if it has been validated and has no errors. The <strong>Types</strong> tab displays a list of namespaces and the belonging types. To view the schema for a type, select a type from the <strong>Types</strong> tab. To view the facets and attributes for a node, select the node from the <strong>Schema</strong> tab.</td>
</tr>
<tr>
<td>Recategorizing libraries</td>
<td>To move a library to a different category, edit the <strong>Category</strong> field in the library details.</td>
</tr>
</tbody>
</table>

**Table 14. Working with resources**

<table>
<thead>
<tr>
<th>Task</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importing resources</td>
<td>1. Select the name of a library, and click <strong>Import new resource</strong>.</td>
</tr>
<tr>
<td></td>
<td>2. Select the file to import, and click <strong>Open</strong>. You can select a single <strong>json</strong> file. You cannot import <strong>json</strong> files from a <strong>zip</strong> file.</td>
</tr>
<tr>
<td></td>
<td>3. After importing all of the required resources into the library, the library is automatically validated. If the library is valid, a green checkmark displays beside its name. If it is not valid, a red exclamation point displays beside its name.</td>
</tr>
<tr>
<td></td>
<td>4. If the library is invalid, click <strong>Validate</strong> to display the list of errors.</td>
</tr>
</tbody>
</table>
Creating an JSON schema in the schema library
You can create an library and import the JSON file to create a JSON schema.

About this task
If you select a JSON instance, the contract library derives an XML schema from the JSON instance and imports the derived schema into the contract library.

Procedure
1. Start the IBM InfoSphere DataStage and QualityStage Designer.
2. Click Libraries tab from the Assembly Editor to open the Contract Libraries window.
4. In the New Contract Library window, enter the name of the library, and then click OK.
5. Select the library that you created, and then click Import New Resource from the Resource View to import the JSON file.
6. Select the JSON file from the list of files. The file type can be .json, .jsn or .jsd
7. Click OK to close the Libraries window. The contract library derives an XML schema from the JSON file.

JSON Parser step
Use the JSON Parser step to parse one or more documents that have the same structure.

<table>
<thead>
<tr>
<th>Task</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displaying the types in schemas</td>
<td>Double-click the name of a library to display the Types and Schema tabs. On the Types tab, click the plus sign beside a namespace to display a list of types that belong to the namespace. Click a type to display the schema for the type on the Schemas tab. Browsing the types and their structures is useful for a variety of reasons. You can determine which type corresponds to a specific document instance by matching the top-level element name with the type name. You can determine which pieces of information are optional but could be included in similar documents. You can view the characteristics of the data, so that you can understand how to set transformation options.</td>
</tr>
<tr>
<td>Exporting resources</td>
<td>Select the resource, and then click Export to save a copy of the resource on the local computer.</td>
</tr>
<tr>
<td>Updating resources</td>
<td>If you modify a resource and want to update it in the library, select the resource, and then click Replace Selected Resource. The updated resource has no effect on jobs that used the earlier version of the resource. To use the updated version in an existing job, you must edit the assembly.</td>
</tr>
<tr>
<td>Deleting resources</td>
<td>Click the Delete icon that displays beside the resource name. Deleting a resource has no effect on existing jobs that use the resource.</td>
</tr>
<tr>
<td>Displaying the resources in a library</td>
<td>Click the name of a library. For the XML stage, the Resources view displays a file location and a namespace for each schema in the library. Click the name of a schema to display additional details about each schema.</td>
</tr>
<tr>
<td>Editing resource details</td>
<td>Only the File Location and Description fields can be updated. After you update the file location, the library is automatically validated.</td>
</tr>
</tbody>
</table>
**JSON source**

In this section, specify the source of the JSON data.

**String set**
Select the node that contains the JSON data. The node can be a field which comes from the previous input step in the DataStage or a chunked node from the previous parser step. You can select only items that contains the JSON data.

**Single file**
Enter the path and file name, or click Insert Parameter, and then select the name of the parameter. You can select parameters that you defined in the job and built-in macros that are in IBM InfoSphere DataStage.

**File set**
Select the input schema item that contains the full file paths of the JSON files at run time. For example, the full file path might be C:\test.json. Use this option to read multiple JSON files that are based on the same schema.

**Document root**
Select the top-level element or a view of a JSON schema that describes the JSON data that you want to parse. The JSON schema is a schema generated while importing a JSON data instance into the schema library.

When you select the document root, the list of elements includes only elements available in the resource selected as document root.

**Validation**
By default, when the JSON Parser step runs, it uses minimal validation, which disables some of the validation rules and provides better performance than strict validation does. Strict validation is configured so that each validation rule is set to Fatal, and the job aborts if the input file has invalid data. To customize validation, specify the action to be taken when a violation occurs.

**Related tasks:**
“Example 1: Parsing JSON data” on page 180
You can create a simple job that uses the JSON Parser step to parse contact data, which is stored in one JSON data file, into two flat files.

**JSON Parser validation rules**
By default, the JSON Parser uses minimal validation and ignores violations. To customize validation, specify the action to perform when a violation occurs.

The following tables describe the validation rules and the actions that the JSON Parser can complete based on the setting of the rule.
<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type has illegal</td>
<td>The value must conform to the value rules for the data type of the corresponding item.</td>
<td><strong>Ignore</strong> Type checking is not performed. The data types of the items as they are defined by the schema are replaced by the String type. For example, if the schema includes an item named income that has the decimal data type, the data type of the item in the step output becomes String.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Log per Occurrence; Log once per Document</strong> Log per occurrence logs error for each occurrence and log once per document logs error for the first occurrence of all errors of the entire document.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Reject</strong> Type checking is performed. Only invalid values fail the invalid document parsing. The ParsingStatus item is added as a new group to the step output for the JSON Parser step and is set to False. The ParsingStatus item contains the corresponding error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Fatal</strong> Type checking is performed. An invalid value causes the job to fail.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Using global default value</strong> If the value is not present for any key, then it takes the value that is specified in the administration page.</td>
</tr>
<tr>
<td>Trim values</td>
<td>Trim the white space before and after a value before performing type checking and type conversion.</td>
<td><strong>False</strong> Trimming is not performed. <strong>True</strong> Trimming is performed on both sides of the value.</td>
</tr>
</tbody>
</table>
**Table 16. Structure validation rules**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document is malformed</td>
<td>The document is malformed.</td>
<td>Ignore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No error message is logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log once per Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An error message is logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reject</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The ParsingStatus item,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>which is added as a new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>group to the step Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for the JSON Parser step,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is set to False and includes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the corresponding error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A malformed document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>causes the job to fail.</td>
</tr>
<tr>
<td>Items are not declared in</td>
<td>Items in the instance document must be declared in the schema. Violations might occur when a</td>
<td>Ignore</td>
</tr>
<tr>
<td>the schema</td>
<td>job uses a version of a schema that is outdated.</td>
<td>No error message is logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log once</td>
</tr>
<tr>
<td></td>
<td></td>
<td>per Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error messages are logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Items are parsed based on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>best guess.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reject</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The ParsingStatus item,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>which is added as a new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>group to the step Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for the JSON Parser step,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is set to False and includes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the corresponding error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An item that is not declared</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in the schema causes the job</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to fail.</td>
</tr>
</tbody>
</table>

**JSON Composer step**

Use a JSON schema or a view created from the JSON schema to compose JSON data.

**JSON Target**

**Write to file**

Enter the output directory and the file name prefix for the files to compose, or click Insert Parameter and then select the name of the parameter for the output directory and file name prefix. The parameters that are available are those that you previously defined in the job and the built-in macros that are in IBM InfoSphere DataStage.

**Pass as string**

Pass the composed JSON string to a downstream step or stage for further processing.

**Pass as large object**

Pass the composed JSON string as a large object. The last stage in the job must be a LOB-aware stage, such as the DB2 connector, Oracle connector,
ODBC connector, Teradata connector, or Websphere MQ connector. These stages use the LOB locator string to obtain the JSON data and then write it to the target database or message queue. The job can contain stages that are not LOB-aware, but the stages must not modify the LOB locator string. If the last stage in the job is not LOB-aware, the LOB locator is written as data and is not interpreted as a locator.

**Document Root**

Select the top-level element or a view of a JSON schema that describes the JSON data that you want to compose. The JSON schema is a schema generated while importing a JSON data instance into the schema library.

**Validation**

By default, the JSON Composer uses strict validation, and the job fails if a violation occurs. To customize validation, specify the action to perform when a violation occurs.

**Mappings**

Create a mapping to the document_collection item. How you map this item determines whether one document or multiple documents are created. To produce only one document, map the root of the input (top) to the document_collection item. If you want each item in the list to be in a separate document, then map a list to the document_collection item. If you produce multiple documents, one file is created for each item in the list. For more information about mapping, see "Working with the mapping table" on page 18.

An JSON array is modelled as multiple occurrences of mixed types. The choiceDescriminator indicates which choice branch you should select for the particular array element. Below is an example of JSON array.

```json
[
    {"type": "home", "number": "212 555-1234"},
    [116, 943, 234],
    true
]
```

In the above example, the choiceDescriminator has the value of objectValue for the first array element which is an object. The choiceDescriminator has the value of arrayValue because the second array element is a nested array. The choiceDescriminator has the value of booleanValue. The choiceDescriminator value contains stringValue, numberValue, booleanValue, nullValue, objectValue, and arrayValue. When composing the JSON data you need to choose which value should be used from the available values. If you want the value as objectValue for the choiceDescriminator, then from the Source column, select **Constant** from the drop down list and enter the Constant Value as objectValue and click **OK**. In a similar way, if you want the value as stringValue, then the Constant value should be **stringValue**

**Format**

**Encoding type**

Select the encoding to use for the document. The default encoding is UTF-8.
Format style

Apply the following format options to the JSON output:

**Indentation length**
Select the number of characters to use for each indentation level in the JSON output.

**New line style**
Select the type of new line. Choices are UNIX (LF), DOS(CRLF) or MAC(CR).

Related concepts:
“Working with the mapping table” on page 18

Some steps, such as the Output step and the XML Composer step, require that you create mappings that define how to create the target nodes.

Related tasks:
“Example 2: Composing JSON data by using the JSON Composer and HJoin steps” on page 190

Build this job that uses the JSON Composer and HJoin steps in the XML stage to create a hierarchical structure.

**JSON Composer validation rules**

By default, the JSON Composer uses strict validation, and the job fails if a violation occurs. To customize validation, specify the action to perform when a violation occurs.

The following tables describe the validation rules and the applicable actions.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type has illegal value</td>
<td>The value must conform to the value rules for the data type of the corresponding item.</td>
<td>Ignore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log once per Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fatal</td>
</tr>
<tr>
<td>Ignore</td>
<td>Type checking is not performed. The data types of the items in the Target column of the mapping table are replaced with the String data type.</td>
<td></td>
</tr>
<tr>
<td>Log per Occurrence; Log once per Document</td>
<td>Log per occurrence logs error for each occurrence and the log once per document logs error for the first occurrence of all errors of the entire document.</td>
<td></td>
</tr>
<tr>
<td>Fatal</td>
<td>(Default) Type checking is performed. An invalid value causes the job to fail.</td>
<td></td>
</tr>
</tbody>
</table>
Table 17. Value validation rules (continued)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item that cannot be null has a null value</td>
<td>Items that cannot be null are checked to ensure that they do not contain null values.</td>
<td>Ignore: Null checking is not performed. All items in the Target column of the mapping table become nullable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Log per Occurrence; Log once per Document: Null checking is performed, and errors are logged. Null values are set. As a result, all items in the Target column of the mapping table become nullable.</td>
</tr>
<tr>
<td>Trim values</td>
<td>Trim the white space before and after a value before performing type checking and type conversion.</td>
<td>False: Trimming is not performed. True: (Default) Trimming is performed on both sides of the value.</td>
</tr>
</tbody>
</table>

Large Schema

Use the XML Stage to parse and compose XML files based on large schemas.

Auto Chunking

The auto-chunk feature provides an automated and quick approach to limit the size of the schema tree for an XML parser step and parse the XML data with large schema.

The auto-chunk algorithm performs a breadth-first search, counts the number of encountered nodes, and starts auto-chunk operations based on the configurable soft and hard limits. For more information about configuring the soft and hard limits, see “Configuring the auto-chunked elements in the assembly” on page 48.

For large schemas, some of the XML elements are automatically chunked by a parser step. Chunking refers to converting all child XML elements to a single XML element.

Figure 1 shows the actual schema representation. When a large schema is loaded as the document root for an XML parser, the element name is automatically chunked as shown in the Figure 2. The auto-chunked element is represented with the greater than symbol and less than symbol (<>), and all the child elements of the
element name are contained within this single element.

*Figure 1. Example of actual schema representation*

*Figure 2. Example of schema representation after chunking*
Configuring the auto-chunked elements in the assembly

You can use a configuration screen to set the thresholds to trim the schema tree.

Procedure

1. Specify the value in the Schema Trimming start at and Maximal Schema Tree size fields.
2. The XML stage starts to chunk the top-level XML elements when the tree size reaches the value for the schema trimming property.

3. When the tree size reaches the value specified for the maximum size, the XML Stage automatically chunks all the qualified XML elements (both the top-level XML elements and the nested XML elements).
   a. For schemas with fewer than 500 nodes the XML stage does not automatically chunk any schema elements. To select a large schema as a document root for the XML parser step, use multiple parsers to extract the auto-chunked elements.

4. Each parser step can have different auto-chunking values set. After defining the schema in the Document Root of the Parser Step, the values can be changed. The changed values cannot affect the schema representation in the Parser Steps where the Document Root is already defined.

**Parsing XML data with a large schema**

To parse XML in a large schema, you must configure multiple XML Parser steps in the assembly for each auto-chunked set of elements.

**About this task**

The parser step does not parse the XML data for the descendants of an auto-chunked element. The data for the auto-chunked element and its descendants are passed to next step as an XML data chunk. You need to configure multiple XML Parser steps to parse the data chunk for the descendants.

**Procedure**

1. Import the schema in to the schema library.
2. Add an XML Parser step to the assembly outline.
3. In the XML source field, enter the source of the XML data.
4. On the **Document Root** tab, click **Browse** and select the schema element that describes the documents that the step parses.

5. Add a second Parser step to the assembly.

6. In the second Parser step, select the **String Set** option in the **XML source** tab and select an auto-chunked node in the **Document Root** tab.

7. After configuring the second Parser Step, the child items of the auto-chunked element are available for mapping in the **Output Step**.

**Related concepts:**

“Parsing an XML file using schema views” on page 51

By using the schema views, you can parse an XML file with only a single Parser step.

**Design Consideration in the Assembly Editor**

When the auto-chunked element is selected in the second Parser Step, then the **Document Root** is automatically defined. This method holds true if the first and second Parser Steps are defined one after another without saving the assembly in between. But in case the Assembly Editor is closed after defining the first Parser Step, then the **Document Root** will not be automatically populated in the second Parser Step when it is added to the assembly on reopening the Assembly Editor.

To address this issue, you need to go to the first parser and set the document root again. You must note the following points:

- There is no need to delete or reconfigure the first parser, you need to reset the **Document Root**.
- This also applies to the following scenarios where you have several parsers in the assembly:
  - When the first Parser is used to import the whole schema and the rest of the Parsers are used to parse some auto-chunked elements from the output of the first parser.
  - If you reopen the job and want to add another parser to parse another auto-chunked schema element, then you need to reset the document root in the first parser

**Schema views**

A schema view is a defined subset of a schema.

When a schema contains a large number of nodes, you create schema views to reduce the size of schema tree, enhance the design experience, and improve the efficiency of the parsing and composing processes.

Schema views are saved in the schema library manager so that they can be reused for different steps and different job designs.

**Creating and modifying schema views**

To reduce the size of the schema tree of a large schema, create a schema view from any XML element.

**Procedure**

1. Start the IBM® InfoSphere® DataStage® and QualityStage™ Designer.
2. On **Libraries** tab, click **New Library** or select an existing schema library.
3. Select the node in the XML schema that you want to create a schema view for.
4. Click **Create View**.
5. Enter the Schema library name and the description.
6. Use the Find option to locate the nodes by a name. You can also search the nodes by specifying the node level. Click Next after entering the node name and specifying the node level.
7. Specify the properties for the schema view.

Table 18. Specifying the values for creating view

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include in View</td>
<td>Select this check box for the optional nodes that you want to include in the view. Mandatory nodes are selected automatically. You need to manually select the nodes that are optional.</td>
</tr>
<tr>
<td>Include All Descendants</td>
<td>Select this check box to include all XML elements that are under the selected node.</td>
</tr>
<tr>
<td>Chunk</td>
<td>Select this check box to chunk schema tree of a selected XML element as one single XML node with the XML type.</td>
</tr>
</tbody>
</table>

8. Click OK to save the view.

Parsing an XML file using schema views

By using the schema views, you can parse an XML file with only a single Parser step.

When parsing a small XML file by using the XML stage, it requires a large number of Parser steps as there will be a large number of auto-chunked nodes in the schema that needs separate Parser steps to parse it. For example, consider parsing a XML file for the FPML 4.9 schema. Assume that the XML file has only certain number of elements and not all the elements are defined in the schema. To parse such a small XML file, XML stage requires a large number of Parser steps. The following figure displays the auto-chunked node in the schema and the assembly design.
The schema view allows you to simplify the assembly when parsing such small XML files. You can create the view on top of the original schema to include only the required elements which are present in the XML file. Hence, you can reduce the schema size and there will be no auto-chunked nodes in the schema. You can also import the view into the Parser step. Now you require only a single Parser step to parse the file as there will not be any auto-chunked node as the size of the schema will be smaller.

**Note:** If the schema view created for the schema is also very large, then the auto-chunked nodes can be present in the view.

**Related tasks:**
"Parsing XML data with a large schema" on page 49

To parse XML in a large schema, you must configure multiple XML Parser steps in the assembly for each auto-chunked set of elements.

**Composing an XML file using schema views**

By creating a schema view from a large schema, you can use the composer step to compose XML data based on the structures and types defined in the schema view, rather than in the original schema.

XML schemas can be very complicated. They can contain XML types, elements, structures, and definitions for many XML documents. They can contain thousands of XML nodes and provide large amounts of information. You can create views from those complex schemas to reduce the size of schema tree and simplify your composing design process.

You can use schema views to compose an XML document in the following cases:
- If the XML schema contains the definitions of many types of XML documents and you want to compose XML data for one particular type, you can create a view for the XML document type from your XML schema and remove all the features that are not related to your document type from the view. You can then select the view as the document root for the XML composer step, define the mappings, and compose your XML documents.

- If the schema that describes the XML data contains large amounts of information, you can define multiple views on top of your schema; each view describing one part of your XML schema. You can add one XML composer step for each view to build one part of your XML document. You can assemble all the generated parts by using the additional views and composer steps.

The following figure shows an example of using schema views to compose XML data from multiple parts of an XML document.

**Creating Schema Views**

Schema views help you to reduce the size of the schema tree. Schema views are saved in the schema library manager.

**Procedure**

1. Create two views based on the original department schema as shown in the below figure.
2. The **employee_view** contains only the types and elements that describe the employee information.

3. The **departments_view** is created to combine the information described in the **employee_view** with the department information. As shown in the below figure, the employee in the **departments_view** is chunked and represented by a
single node. The types and elements of the chunked node "employee" are described by the `employee_view`.

---

**Creating an assembly**

Using the Assembly Editor, you can create an assembly. By default, an assembly contains an Overview, an Input step, and an Output step. You add additional steps to the assembly, based on the type of transformations that you want to perform.

**About this task**

In this example, you need to add multiple steps to compose an XML document.

**Procedure**

1. Create two HJoin steps and two composer steps for the below described schema.
2. The **HJoin_Employee_Address Step** joins the “Employee” list with the “Address” list.
3. The **XML_Composer_Employee Step** composes an XML data described by the "employee_view". As shown in the following figure, the "employee_view" is selected as the document root to describe the XML data produced by the **XML_Composer_Employee Step**.

4. The **HJoin_Dept_Employee Step** joins the "DeptInfo" list with the "Employee" list.
5. The **XML_Composer_depts Step** combines the XML data generated from the **XML_Composer_Employee Step** with the department information and generates the final XML data described by the “departments_view”. As shown in the following figure, the “departments_view” is selected as the document root to describe the XML data produced by the **XML_Composer_depts Step**.

6. The result-string generated from the **XML_Composer_Employee Step** is mapped to the value for the chunked node “employee”. 
Transformation steps for the XML stage

Transformation steps for the XML stage are available from the palette in the Assembly Editor.

Aggregate step

Perform hierarchical aggregations on the items in a list.

In the example below, the aggregate function, average, is computed for the salary of all employees in each department. The departmentID is used as the key to group Employee items and the averageSalary is computed for each group. The result list contains an item for each distinct departmentID/grouping.

```xml
<Company[]>
  <Employee[]>
    <departmentID/>
    <salary/>
  </Employee[]>
  <result[]>
    <keys>
      <departmentID/>
      <aggregate>
        <averageSalary/>
      </aggregate>
    </keys>
  </result[]>
</Company[]>
```

List to Aggregate

Specify the list that contains the elements that will be iterated on.

Scope

Select the item that defines when to produce the result of the aggregation.

Aggregation Item and Aggregation Function

Select an items and the function to use for the aggregation. The function that you select must be applicable to the data type of the selected item. The following table describes the available aggregation functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>Calculates the average value in the list.</td>
</tr>
<tr>
<td>Count</td>
<td>Counts the elements in the list. Ignore null values.</td>
</tr>
<tr>
<td>Concatenate</td>
<td>Concatenates all strings in the list, starting with the first element.</td>
</tr>
</tbody>
</table>
Table 19. Aggregation functions (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>Selects first value in the list.</td>
</tr>
<tr>
<td>Last</td>
<td>Selects last value in the list.</td>
</tr>
<tr>
<td>Maximum</td>
<td>Selects the maximum value in the list. If the list contains Boolean strings, True is greater than False.</td>
</tr>
<tr>
<td>Minimum</td>
<td>Selects the minimum value in the list. If the list contains Boolean strings, False is less than True.</td>
</tr>
<tr>
<td>Sum</td>
<td>Calculates the sum of all of the values in the list.</td>
</tr>
<tr>
<td>Variance</td>
<td>Calculates the variance value of all of the values in the list.</td>
</tr>
</tbody>
</table>

**Aggregation Keys**
Specify aggregation keys to produce multiple aggregation rows for each unique key value.

**H-Pivot step**
Transpose a list into a set of items, based on a key.

- **Scope** Specify the list to transform.
- **Columns** Specify the columns to convert into rows. For each column that you specify, a new record is created.

**HJoin step**
Transform the items from two lists into one nested list.

- **Parent List** Specify the parent list. The parent list cannot already have a containment relationship with the child List.
- **Child List** Specify the child list.
- **Optimization Type** If all of the records have the same key and are consecutive to each other, select In-memory as the optimization type.
- **Parent Keys** Specify the key to use to join the lists.
- **Child Keys** Specify the key to use to join the lists.

For an example of using the HJoin step, see Example 3

**Order Join step**
Join items based on their position in two lists.
Select two lists to join into one list. The items in each list are joined, based on their position in the list. For example, item 1 in list 1 is joined to item 1 in list 2. If one list has fewer items than the other list, items that have a null value are added to the shorter list.

**Regroup step**

Use the Regroup step to create a parent-child list relationship from a single list.

Remove redundancy in a data set by using the Regroup step to transform the items in one list into nested list. Identify items that contain redundant data and move them into a parent list. Identify child items to move in to a child list.

**List to regroup**

Select the list to regroup.

**Scope**

Select the scope.

**Parent and Child Items**

Drag items to identify which belong to the parent list and which belong to the child list.

**Keys**

Specify one or more keys to use for the regroup operation. For each unique instance of a key field, a separate entry is made in the list.

**Input records of regroup lists are clustered by key – optimize execution**

If the input records have already been sorted by the selected key, select this option to improve performance.

For an example that includes the Regroup step, see [Example 2](#).

**Sort step**

Sort the items in a list by one or more keys.

**List to Sort**

Specify the list to sort.

**Scope**

Specify the scope.

**Data set fits into memory; optimize sort**

Select this option to improve performance.

**Keys and Order**

Specify one or more keys by which to sort the values, and specify the sort order for each key.

**Switch step**

Use the Switch step to classify items into one or more new target lists, based on constraints that you specify.

Each target list is associated with a constraint. The target list contains all of the items that passed the constraint. The Default list contains all of the items that failed all of the constraints.

To create a new target list and one or more constraints, click **Add Target**.

The following table describes the functions that you can use to create a constraint:
## Table 20. Functions for the Switch step

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isNull</td>
<td>All types</td>
<td>Returns True if the value is null.</td>
</tr>
<tr>
<td>Greater than</td>
<td>Any number</td>
<td>Returns true if the value is greater than the value of the parameter.</td>
</tr>
<tr>
<td>Greater than or equal</td>
<td>Any number</td>
<td>Returns true if the value is greater than or equal to the value of the parameter.</td>
</tr>
<tr>
<td>Less than</td>
<td>Any number</td>
<td>Returns true if the value is less than the value of the parameter.</td>
</tr>
<tr>
<td>Less than or equal</td>
<td>Any number</td>
<td>Returns true if the value is less than or equal to the value of the parameter.</td>
</tr>
<tr>
<td>Equals</td>
<td>All types</td>
<td>Returns true if the two values are the same.</td>
</tr>
<tr>
<td>Between</td>
<td>Any number</td>
<td>Returns true if the value is within the specified range.</td>
</tr>
<tr>
<td>IsTrue</td>
<td>Boolean</td>
<td>Returns true if the value is true.</td>
</tr>
<tr>
<td>IsFalse</td>
<td>Boolean</td>
<td>Returns true if the value is false.</td>
</tr>
<tr>
<td>Compare</td>
<td>String</td>
<td>Returns true if the string value is the same as the string value of the parameter.</td>
</tr>
<tr>
<td>CompareNoCase</td>
<td>String</td>
<td>Returns true if the string value is the same as the string value of the parameter. Ignores the case.</td>
</tr>
<tr>
<td>Contains</td>
<td>String</td>
<td>Returns true if the string value contains the string value of the parameter.</td>
</tr>
<tr>
<td>ContainsCaseInsensitive</td>
<td>String</td>
<td>Returns true if the string value contains the string value of the parameter. Ignores the case.</td>
</tr>
<tr>
<td>IsBlank</td>
<td>String</td>
<td>Returns true if the string is empty or null.</td>
</tr>
<tr>
<td>IsNotBlank</td>
<td>String</td>
<td>Returns true if the string is not empty and not null.</td>
</tr>
<tr>
<td>Like</td>
<td>String</td>
<td>Returns true if the string value matches the pattern defined by the parameter value. Use a percent sign (%) to define missing letters before and after the pattern.</td>
</tr>
</tbody>
</table>
Union step
Use the Union step to combine two lists that have different structures into a single list that has a predefined structure.

On the Union Type tab, you select the target list schema. On the Mappings tab, you map the two lists.

V-Pivot step
Use the V-Pivot step to transform records into fields of another record.

You define the record source and scope. For each column name, the step output contains a branch with a single item that has the same structure as the step input. Based on the value of the Source of Column Names field, the record is classified into the correct branch.

Examples of transforming XML data
Build these sample jobs that parse and compose XML data.

The examples all use one data file, departments.xml, and two schemas (Employee.xsd and Organization.xsd). You can create each example yourself, by following the step-by-step instructions, or you can import the finished job and explore it on your own. The data for the examples is an XML file that contains information about the employees in one department of a company. The schemas define the structure for the employee information and for the department information.

If you are just getting started with the XML stage, begin by building Example 1, which includes tasks that illustrate the use of the Assembly Editor and the Schema Library Manager.

To get the files for the examples, go to http://www.ibm.com/support/docview.wss?uid=swg27019894.

Example 1: Parsing XML data
Create a simple job that uses the XML stage and the XML Parser step to parse employee data, which is stored in one XML data file, into two flat files.

About this task
This basic parsing example uses the sample XML data file, departments.xml, and the sample schemas, Employee.xsd and Organization.xsd, to illustrate parsing source data from one XML file into two files. The departments.xml file contains information about the employees of one department in a company. In this example, you parse the XML data into two files. One file contains employee business information: employee name, date of hire, date of birth, gender, title, employee ID and department ID. The second file contains employee address information: employee ID, street, city, state, postal code, country, and address type.

To create the example, complete these tasks:

Example 1: Creating the job
Create the example job that includes one XML stage and two Sequential File stages.
About this task

The following figure shows the job that you create for the parsing example. The job includes one XML stage, named Departments, and two Sequential File stages, named Employee_File and Address_File. The Departments stage is linked to the Employee_File stage by a link named Employee, and it is also linked to the Address_File stage by a link named Address.

![Diagram of job setup]

Procedure

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and select New > Parallel job.
3. Open the Real Time section of the palette, and drag one XML stage to the canvas.
4. Perform the following steps to create a job property for the location of the example files:
   b. On the Parameters tab, in the Parameter Name field, enter xml_example_root_folder and then specify the following values:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Root of example folder tree</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Point to the root of the file tree folder</td>
</tr>
</tbody>
</table>

   Each example uses this job parameter.
5. Open the File section of the palette, and drag two Sequential File stages to the canvas. Position these stages to the right of the XML stage.
6. Create a link from the XML stage to each sequential file stage.
7. Rename the stages and links as shown in the following table:

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML stage</td>
<td>Departments</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Employee_File</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Address_File</td>
</tr>
<tr>
<td>Link from Departments to Employee_File</td>
<td>Employee</td>
</tr>
<tr>
<td>Link from Departments to Address_File</td>
<td>Address</td>
</tr>
</tbody>
</table>
8. By looking at the departments.xml file, which contains the XML data, you determine which columns to create to hold the employee data that the XML stage will pass to the Employee_File stage and the Address_File stage. Double-click the Employee_File stage to open the stage properties. Then click the Columns tab, and configure the following columns:

Table 23. Columns for the Employee_File stage

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>MiddleName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>FirstName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>Gender</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>BirthDate</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>HireDate</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>EmployeeID</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>DepartmentID</td>
<td>VarChar</td>
<td></td>
</tr>
</tbody>
</table>

The column names that you create closely resemble the names of the XML schema elements. In the Output step of the assembly, where you configure the mapping between target items and source items, you will see how name similarity affects the mapping suggestions that are returned.

9. Click the Properties tab, and configure the following properties that define the output file:

Table 24. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Enter the path where you want the output file to be created, followed by the file name employee.txt.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose Overwrite to create the file.</td>
</tr>
<tr>
<td>First line is column name</td>
<td>Set to True.</td>
</tr>
</tbody>
</table>

10. Click OK to close the Employee_File stage.

11. Double-click the Address_File stage. On the Properties tab, configure the following properties that define the output file:

Table 25. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Enter the path where you want the output file to be created, followed by the file name address.txt.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose Overwrite to create the file.</td>
</tr>
<tr>
<td>First line is column name</td>
<td>Set to True.</td>
</tr>
</tbody>
</table>

12. For the Address_File stage, do not create any columns now. You will later use the Assembly Editor to propagate those columns automatically.

13. Click OK to close the stage properties.

14. Choose File > Save, and name the job as xml_parser_example.
Example 1: Opening the Assembly Editor and viewing the issues

For each XML stage, you open the Assembly Editor, which you use to define an XML transformation within the context of a job.

About this task

Using the Assembly Editor, you create an assembly. An assembly contains a series of steps that parse, compose, and transform hierarchical data. By default, an assembly contains an Overview, an Input step, and an Output step. You add additional steps to the assembly, based on the type of transformations that you want to perform. In this example, you add one XML_Parser step to parse XML data into two output Sequential File stages, Employee_File and Address_File.

Procedure

1. To open the Assembly Editor, double-click the Departments stage on the canvas. The XML stage editor opens. From the XML stage editor, click Edit assembly to open the Assembly Editor.

2. The Assembly Outline displays the steps in the assembly. This assembly requires an XML Parser step. Click Palette to open the palette and then double-click the XML Parser step to add it to the outline. Because this is the first step that you are adding to the assembly, the step is automatically added between the Input step and the Output step. When you add another step, first select the step above which you want the new step to be located, and then double-click the step in the Palette. The new step is added below the selected step.

3. Now you have created the basic assembly structure, but you still need to configure the steps in the assembly. Before you perform the configuration, notice that in the outline, an error icon (a red exclamation mark) displays beside the XML Parser step and the Output step. The error icon indicates that you must address one or more issues in that step. The total number of issues appears in the upper-right corner of the Assembly Editor. To view the issues list, click View All. The following figure shows the issues for this assembly:
The XML Parser step has two errors, indicating that you must complete two mandatory fields. The Output step has 12 errors. Notice that the errors that require mandatory input are for the output links Employee and Address and the columns that you defined on the Employee link. These links and columns are listed as errors because you have not yet mapped the hierarchical structure of the assembly back to a relational structure. The first error in the Output list indicates no columns were defined on the Address link. For now, you can ignore these errors; you will correct them when you create mappings in the Output step.

4. Close the issues list.

**Example 1: Importing the schemas for the examples**

Use the Schema Library Manager to import the example schemas into a library.

**About this task**

You must import the schemas that the example job uses. Imported schemas are stored in the metadata repository, where they are available for use in any assembly that you create.

**Procedure**

1. From the Assembly Editor, click the Libraries tab to open the Schema Library Manager. The Schema Library Manager is available from the Assembly Editor and from the IBM InfoSphere DataStage and QualityStage Designer client. From the InfoSphere DataStage and QualityStage Designer client, you choose Import > Schema Library Manager.

2. To create a library for the example schemas, click New Library. For the library name, enter Schemas_for_XML_examples. For the category, enter Examples. The library is now categorized under Examples. If you have additional example schemas, you can add them to this library or create a new library for them; all of your example schemas can be categorized under the Examples category.

3. Expand the Examples category, select Schemas_for_XML_examples, and then click Import New Resource. Find the Employee.xsd and Organization.xsd, import both into the library.
Note: Multiple .xsd files can be imported at a time into the library. This can be done by selecting multiple files in the browse window on clicking Import New Resource.

4. Click the Assembly Editor tab to return to the assembly.

Example 1: Configuring the Overview
The Overview provides creation and modification information about the assembly and includes a Description field that you can modify.

About this task
On the Overview, you can enter an optional description for the assembly. This field documents the purpose of the assembly and is helpful when you or others later need to modify the assembly.

Procedure
In the Description field, enter the following description: This assembly uses the Organization.xsd and Employee.xsd schemas. The assembly parses the departments.xml data file and sends the output to two files named address.txt and employee.txt.

Example 1: Configuring the XML Parser step
Configure the location of the XML source data and the schema that you want to use to parse it.

Procedure
1. Click the XML Parser step in the outline. By default, the Configuration tab for the step opens. The following figure shows the Configuration tab for the step:

![XML Parser Step Configuration Tab](image)

2. On the XML Source tab, you specify the location of the XML source data. In this example, the XML source data is in the single file departments.xml. Select Single file, and then click Insert Parameter and select the xml_example_root_folder parameter. You will specify the exact location of the file when you run the job. You can also specify the absolute path to departments.xml.

3. On the Document Root tab, you select the schema element that describes the documents that the step parses. Click Browse. Open the Schemas_for_XML_examples library, which is the library into which you imported the Organization.xsd and Employee.xsd schemas.
4. Click to open the Organization.xsd schema, click the root element departments, and then click OK. The Document Root tab displays the structure of the schema. The step Output also displays this same structure.

5. On the Validation tab, select Strict Validation. By selecting Strict Validation, you automatically ensure that the data types conversion is performed. If you use the default Minimal Validation, all data types are automatically converted to the String type.

Example 1: Configuring the Output step
In the Output step, create mappings that define how to map source items in one data structure to target items in another data structure.

About this task
In this assembly, you use the Output step to map a hierarchical data structure to a relational data structure.

Procedure
1. Click the Output step in the Assembly Outline to open the step. The step displays the Output window of the Configuration tab. In the Output window, the output table describes the data structure that is the output of the assembly. The following figure shows the relevant columns of the output table for this assembly:

The default view in the Output window is the Links view. The Links view looks similar to the table that displays on the Columns tab in the XML stage editor. The output structure for Employee link is shown. Notice that the columns that you defined in the XML stage editor display in the output table. In the Links view, you can modify the columns that you already defined. Any changes that you make to the columns are propagated to the column definitions in the XML stage properties.

2. From the Output Links drop-down list, select Address. The following figure shows that the output table does not display any columns because when you created the job, you did not define any columns for the Address link. The lack of columns is not an error. However, if no columns are defined on the link, you cannot map any source items in the hierarchical data structure to this link. Because this job is designed to produce a file that contains address data,
you need to create the address columns. But instead of returning to the job and manually creating columns, you can automatically create them from the Mappings tab of the Output step.

3. Click the Mappings tab. The following figure shows the mapping table. In this table, each item in the output structure is represented by a row in the table. You map target items to source items. For this job, the target structure is two links, Address and Employee, and the columns that are defined on those links. In the Target column, the Employee link is represented as a list item. Under the Employee list item, each column displays as a content item.

4. To create mappings, first map target list items to source list items. Then map target content items to source content items. There are four ways to specify a mapping: use automatic mapping, select a mapping candidate from a list of valid mapping candidates, select a mapping candidate from the entire source structure using More option in the drop-down list, or specify a constant value as the mapping using Constant option in the drop-down list.

5. Select the Employee list item from the Target column. In the Source column, click and choose employee from the drop-down list. Click Auto Map.

Automatic mapping is context-sensitive, it creates a mapping for the selected item and all of its descendent items. Each source item that is automatically mapped is determined based on similar name and data type.

The following figure shows the result of Auto Map for the Employee list item. The target Employee list item and all of its child items are automatically mapped to source items. If the Employee list item had a descendent list that contained content items, the descendent list and all of its content items would also be automatically mapped.
6. In this example, the target item, BirthDate is wrongly mapped to the source item, hireDate. You need to manually create the correct mapping. Click hireDate in the source list to choose the correct item from the drop-down list. The items in the list appear in order, from highest to lowest, based on their mapping similarity. Select dateOfBirth as the right candidate for the target item, BirthDate from the drop-down list.

7. Next you need to map the Address list item from the Target column. In the Source column, click and choose Address from the drop-down list and click **Propagate**. Propagate automatically creates one column for each item that is a descendent of the Address item in the source structure and automatically maps those items to the respective items in the source column. When you use the **Propagate** button to create items, the items are automatically mapped. The following figure shows the result:

8. Look at Address item in the target structure. The child items describe the address of an employee; however, the structure does not currently contain any way of identifying that employee. To make the resulting data useful, you need to add a column that identifies the employee. You can define a new column directly in the Output step. Then when you save the assembly, the column is
automatically propagated back to the Address link in the job. To relate the Employee to each Address, create an EmployeeID field in the Address list. Follow these steps:

a. Open the Output window on the Configuration tab.
b. In the Output Links field, choose Address. The columns that are mapped to the Address link display in the table.
c. Click the first empty row at the bottom of the table where Click to add is displayed. Enter the item, EmployeeID in the Name field.
d. Choose VarChar in the Type field.

9. Next you must map a source item to the new target item, EmployeeID. Click and open the Mappings window. You will see that the EmployeeID is added to the Address list in the Target column. Manually map the EmployeeID in the source column to the EmployeeID in the Address list of target column. In the Source column, click and choose employeeID from the drop-down list. If you do not see the items in the drop-down list, click More to find the required item. The following figure shows the result of mapping:

<table>
<thead>
<tr>
<th>Source</th>
<th>Result</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td></td>
<td>top</td>
</tr>
<tr>
<td>...employees/employee</td>
<td></td>
<td>Employee</td>
</tr>
<tr>
<td>...ns0:name/lastName</td>
<td></td>
<td>LastName</td>
</tr>
<tr>
<td>...name/middleName</td>
<td></td>
<td>MiddleName</td>
</tr>
<tr>
<td>...s0:name/firstName</td>
<td></td>
<td>FirstName</td>
</tr>
<tr>
<td>..._fields/ns0:gender</td>
<td></td>
<td>Gender</td>
</tr>
<tr>
<td>.../ns0:dateOfBirth</td>
<td></td>
<td>BirthDate</td>
</tr>
<tr>
<td>...pe_fields/ns0:title</td>
<td></td>
<td>Title</td>
</tr>
<tr>
<td>..._fields/ns0:hireDate</td>
<td></td>
<td>HireDate</td>
</tr>
<tr>
<td>...yee/@employeeID</td>
<td></td>
<td>EmployeeID</td>
</tr>
<tr>
<td>...ee/@departmentID</td>
<td></td>
<td>DepartmentID</td>
</tr>
<tr>
<td>..._fields/ns0:Address</td>
<td></td>
<td>Address</td>
</tr>
<tr>
<td>...ns0:Address/street</td>
<td></td>
<td>street</td>
</tr>
<tr>
<td>...ds/ns0:Address/city</td>
<td></td>
<td>city</td>
</tr>
<tr>
<td>.../ns0:Address/state</td>
<td></td>
<td>state</td>
</tr>
<tr>
<td>...s0:Address/country</td>
<td></td>
<td>country</td>
</tr>
<tr>
<td>...Address/postalCode</td>
<td></td>
<td>postalCode</td>
</tr>
<tr>
<td>...d/s/address_type</td>
<td></td>
<td>address_type</td>
</tr>
<tr>
<td>...yee/@employeeID</td>
<td></td>
<td>EmployeeID</td>
</tr>
</tbody>
</table>

10. Click OK to save your work and return to the XML stage editor. Click the Address link in the preview window, and then click the Columns tab to see that the EmployeeID column that you just created has been propagated back to the job. The following figure displays the preview window.
11. Click **OK** to close the stage editor.

**Example 1: Viewing the output of the job**
After you run the parsing job, open the text files, and look at the output.

**Procedure**
1. From the Assembly Editor, click **OK** to return to the XML stage editor. Then click **OK** to close the XML stage editor.
2. From the IBM InfoSphere DataStage and QualityStage Designer client, choose **File > Compile** to compile the job.
3. Choose **File > Run** to run the job.
4. In the parameter window, for the first parameter, **Root of example folder tree**, enter the path of the directory where you have unzipped your examples zip file. For example, if you have downloaded and saved the examples zip file in the directory, C:\Examples, enter this directory as the **Value** for the first parameter, **Root of example folder tree**.

The following is an example input file that contains the employee and address information:

```
- <department xmlns="" departmentID="A100" departmentKind="Division">
  <manager>A7100</manager>
  - <ns1:employees>
    - <ns1:employee employeeID="A8990" departmentID="A100">
      - <name>
        <firstName>Zen</firstName>
        <middleName>P</middleName>
        <lastName>Wright</lastName>
      </name>
      <gender>male</gender>
      <dateOfBirth>1980-04-04</dateOfBirth>
      <title>Mr</title>
      - <Address>
        <street>2301 East Lamar Blvd</street>
        <city>Arlington</city>
        <state>Texas</state>
        <country>USA</country>
        <postalCode>78363</postalCode>
        <address_type>O</address_type>
      </Address>
    </employee>
  </employees>
</department>
```
5. After the job runs, open the address.txt file and the employee.txt file to look at the results.

The address.txt file contains this data:

```
"employeeID","street","city","state","country","postalCode","address_type"
"A8990","2301 East Lamar Blvd","Arlington","Texas","USA","78363","O"
"A8990","2001 West Street","Arlington","Texas","USA","78300","H"
"B6540","San Felipe, Suite 2400","Houston","Texas","USA","77057","O"
"B6540","53rd West Street","Houston","Texas","USA","77000","H"
"C1230","5th South Street","Miami","Florida","USA","32599","O"
"C1230","54th South Street","Miami","Florida","USA","32501","H"
```

The employee.txt file contains this data:

```
"LastName","MiddleName","FirstName","Gender","BirthDate","Title","HireDate","EmployeeID","DepartmentID"
"Wright","P","Zen","male","1980-04-04","Mr","2008-07-11","A8990","A100"
"Donald","P","Cynthia","female","1987-01-17","Miss","2000-07-25","B6540","A100"
"William","G","Tania","female","1980-01-17","Miss","2002-07-25","C1230","A100"
```

You can now expand or modify this example. If you look closely at the sample XML data file, you see that you can add two additional Sequential File stages to the job. One might store job information about each employee, such as start date and salary; and second might store information about each department, such as the department name, department type, and manager. After you learn the basics of parsing by going through this basic example, you might want to modify the job and the assembly to include these additional stages.

**Example 2: Using the XML Composer and Regroup steps**

Build this job that uses the XML stage and the Regroup and XML Composer steps to create an XML file from two relational files.

**About this task**

This basic composing example uses two relational data files, employee.txt and address.txt, and the sample schemas, Employee.xsd and Organization.xsd, to illustrate composing source data from two relational files into one XML file. The employee.txt file contains information about the employees of one department in a company: employee name, date of hire, date of birth, gender, title, department ID, and employee ID. The second file contains employee address information: street, city, state, postal code, country, address type (home or office), and employee ID. Each employee has two addresses, a home address and an office address. In this example, you use a Join stage in the job to join the two tables. Then you use the XML Stage and the Regroup and XML Composer steps to compose a XML file that contains each employee's information, followed by the employee's two addresses. In Example 3: Using the XML Composer and HJoin steps, you compose the same output file, except instead of using a Join stage in the job, you use an HJoin step in the assembly. Using an HJoin step is an alternative approach to using a Join stage in the job itself.
You can follow these steps to build the job, or you can look at the completed job, which is named xml_composer_example.dsx. To look at the completed job or use it as the starting point for expanding the example, import the job into IBM InfoSphere DataStage.

To create the example, complete these steps:

**Example 2: Creating the job**
Create the example job that includes two Sequential File stages, a Join stage, and an XML stage.

**About this task**

The following figure shows the job that you create for the composing example. The job includes two Sequential File stages that are named Employee_File and Address_File. These Sequential File stages are linked to a Join stage by two links that are named Employee and Address. The Join stage is linked to an XML stage by a link that is named Joined_Data.

![Diagram showing job structure](image)

**Procedure**

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and select New > Parallel job.
3. Open the File section of the palette, and drag two Sequential File stages to the canvas.
4. Open the Processing section of the palette, and drag a Join stage to the canvas. Position the Join stage to the right of the two Sequential File stages.
5. Open the Real Time section of the palette, and drag one XML stage to the canvas. Position the XML stage to the right of the Join stage.
6. Perform the following steps to create a job property for the location of the example files:
   b. On the Parameters tab, in the Parameter Name field, enter xml_example_root_folder and then specify the following values:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Root of example folder tree</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Point to the root of the file tree folder</td>
</tr>
</tbody>
</table>

Table 26. Specifying the values for field names

Each example uses this job parameter.
7. Create a link from each Sequential File stage to the Join stage, and create a link from the Join stage to the XML stage.

8. Rename the stages and links as shown in the following table:

Table 27. Names for job elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential File stage</td>
<td>Address_File</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Employee_File</td>
</tr>
<tr>
<td>Join stage</td>
<td>Join</td>
</tr>
<tr>
<td>Link from Address_File to Join</td>
<td>Address</td>
</tr>
<tr>
<td>Link from Employee_File to Join</td>
<td>Employee</td>
</tr>
<tr>
<td>Link from Join to XML Stage</td>
<td>Joined_Data</td>
</tr>
</tbody>
</table>

9. Configure the Employee_File stage. Double-click the Employee_File stage to open the stage properties. On the Properties tab of the Output page, configure the following properties:

Table 28. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Enter the path and file name of the file. For the Employee_File stage, the file name is employee.txt.</td>
</tr>
<tr>
<td>Read Method</td>
<td>Select Specific File(s).</td>
</tr>
<tr>
<td>First Line is Column Names</td>
<td>Select True.</td>
</tr>
</tbody>
</table>

10. Click the Columns tab, and create these columns:

Table 29. Columns for the Employee_File stage

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>MiddleName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>FirstName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>Gender</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>BirthDate</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>HireDate</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>EmployeeID</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>DepartmentID</td>
<td>VarChar</td>
<td></td>
</tr>
</tbody>
</table>

11. Configure the Address_File stage. Double-click the Address_File stage to open the stage properties. On the Properties tab of the Output page, configure the following properties:

Table 30. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Enter the path and file name of the file. For the Address_File stage, the file name is address.txt.</td>
</tr>
<tr>
<td>Read Method</td>
<td>Select Specific File(s).</td>
</tr>
</tbody>
</table>
Table 30. Configuring the properties (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Line is Column Names</td>
<td>Select True</td>
</tr>
</tbody>
</table>

12. Click the **Columns** tab, and create these columns:

Table 31. Columns for the Address_File stage

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Street</td>
<td>VarChar</td>
</tr>
<tr>
<td>City</td>
<td>VarChar</td>
</tr>
<tr>
<td>State</td>
<td>VarChar</td>
</tr>
<tr>
<td>Country</td>
<td>VarChar</td>
</tr>
<tr>
<td>PostalCode</td>
<td>VarChar</td>
</tr>
<tr>
<td>Address_Type</td>
<td>VarChar</td>
</tr>
<tr>
<td>EmployeeID</td>
<td>VarChar</td>
</tr>
</tbody>
</table>

13. Configure the Join stage. Double-click the stage to open the stage properties. On the **Properties** tab of the Stage page, in the **Key** property, enter **EmployeeID**. This column is in both input data sets and is used to identify which data belongs to which employee. On the **Mapping** tab of the **Output** page, drag the header bar of the Columns table to the Joined_Data table. All of the input columns are automatically mapped to output columns. Click **OK** to close the Join stage.

14. Choose **File > Save**, and name the job **xml_composer_example**.

**Example 2: Creating the assembly**

Use the Regroup step to create a parent-child hierarchy from a single list; then use the XML Composer step to compose the XML file.

**Before you begin**

Import the example schemas **employee.xsd** and **organization.xsd**. If you have already built **Example 1: Parsing XML data**, you have imported the schemas as part of that example. For more information about importing the schemas, see [Importing the schemas for the examples](#).

**Procedure**

1. Double-click the XML stage to open the stage properties, and then click **Edit assembly** to open the Assembly Editor.
2. Open the Palette, and then double-click the Regroup step to add it to the assembly.
3. Select the Regroup step, and then from the palette, double-click the XML Composer step to add it below the Regroup step.
4. On the Overview, in the **Description** field, enter the following description: This assembly uses the Organization.xsd and Employee.xsd schemas. The Regroup step creates a hierarchical structure, where the employee information is the parent and the employee address information is the child. The XML Composer step composes the data and saves the output in a file named **employee_output.xml**.

   This assembly uses the default Input step, which shows the input columns that the Join stage passed to the XML stage. You cannot modify the input columns,
but you can display them as either links or a tree. The following figure shows the relational structure that the Input step provides to the Regroup step. The structure is a flat list named Joined_Data:

![Output](image)

Example 2: Configuring the Regroup step

Use the Regroup step to create a parent-child list from items that share the same key.

About this task

To understand how the Regroup step works, first look at the output that the Join stage produces. The Join stage uses the key field, EmployeeID, to join the tables in the two Sequential File stages into one table. Then for every unique combination of fields, the Join stage produces one record, as shown below:

```
"LastName","FirstName","Gender","BirthDate","Title","HireDate",
"EmployeeID","DepartmentID","street","city","state","country","postalCode",
"address_type"
"Wright","P","Zen","male","1980-04-04","Mr","2008-07-11","A8990","A100",
 2301 East Lamar Blvd","Arlington","Texas","USA","78363","O"
"Wright","P","Zen","male","1980-04-04","Mr","2008-07-11","A8990","A100",
 2001 West Street","Arlington","Texas","USA","78300","H"
"William","G","Tania","female","1980-01-17","Miss","2002-07-25","C1230","A100",
 5th South Street","Miami","Florida","USA","32599","O"
"William","G","Tania","female","1980-01-17","Miss","2002-07-25","C1230","A100",
 54th South Street","Miami","Florida","USA","32501","H"
"Donald","P","Cynthia","female","1987-01-17","Miss","2000-07-25","B6540","A100",
 53rd West Street","Houston","Texas","USA","77000","H"
"Donald","P","Cynthia","female","1987-01-17","Miss","2000-07-25","B6540","A100",
 "San Felipe Suite 2400","Houston","Texas","USA","77057","O"
```

There are two records for each employee because each employee has two addresses. To change this structure into a hierarchical structure, where there is one record for each employee, and that record includes the two related addresses, you use a Regroup step.
Procedure

1. On the Configuration tab of the Regroup step, configure these fields:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>List to Regroup</td>
<td>Select top/InputLinks/Joined_Data. This is the list that contains the items that you want to regroup.</td>
</tr>
<tr>
<td>Scope</td>
<td>Choose top.</td>
</tr>
</tbody>
</table>

2. In the Parent and Child Items window, you create the hierarchy for the list. The items that you drag to the Parent Items list are those that are repeated once for each record. The items that you leave in the Child Items list are those that are subordinate to the parent fields and are repeated more than once. To set up the correct structure for this assembly, drag LastName, MiddleName, FirstName, Gender, BirthDate, Title, HireDate, EmployeeID, and DepartmentID to the Parent Items list. Leave the Street, City, State, Country, PostalCode, and Address_Type fields in the Child Items list.

3. In the Keys window, you specify the key to use for the regroup operation. For each unique instance of the key field, a separate entry is made in the list. You want a list of employees. Each record in the Join stage contains the EmployeeID field, so that is the field to use as the key. Click the pencil icon in the Keys window, and then select top/InputLinks/Joined_Data/EmployeeID.

4. The following figure shows the step Output of the Regroup step. The result of the regroup is a list, called Regroup:result. This list contains the employee fields as well as a child list, named Joined_Data, which contains the address fields.

![Diagram of Regroup:result and Joined_Data]

To make the output tree a little easier to understand, rename the lists. Highlight the Regroup:result item in the list, then right-click to display a menu of actions that you can perform on items on the step output. Select Rename, and then rename the list Employee. Then rename the Joined_Data item to be Address. At any time, you can hover over the item name to display the original name. And using the right-click menu, you can choose Reset to restore the original item name.

5. Click OK to save the assembly. As you work on an assembly, it is important to save your work regularly.
Example 2: Configuring the XML Composer step

Configure the XML Composer step to create a XML file that lists each employee once, followed by the employee’s two addresses.

Procedure

1. In the XML Target window of the XMLComposer step, choose Write to File. In the Output Directory field, click Insert Parameter and choose xml_example_root_folder. In the Filename Prefix field, enter the name of the output file, which in this case is employee_output. Depending on how you create the mappings, you can produce one file or create multiple files.

2. In the Document Root window, select the schema element that describes the documents that the step composes:
   a. Click Browse, and then open the Schemas_for_XML_examples library, which is the library into which you imported the Organization.xsd and Employee.xsd schemas.
   b. Click to open the Employee.xsd schema, click the root element employees, and then click OK. The Document Root tab displays the structure of the schema.

3. Click the Validation tab. By default, validation for the XML Composer is set to Strict. Set the validation to Minimal.

4. In the Regroup step, you established a hierarchical structure where the address items are in a list that is a child of the employee list. Now on the Mappings tab, map the items from the target schema to the items in the Regroup:Employees list. The target schema contains some items that you will not use. For example, the target schema contains a list named Job, which includes items such as jobTitle and startDate. You do not need mappings to items in the Job list because the Regroup:Employees list does not contain any job items. To create the mappings, perform the following steps:
   a. The mapping for the document_collection item determines when a new XML document is created. If you map the top element of the step Input to document_collection, one document is created. If you map a list item to document_collection, one document is created for each item in the list. The name of each resulting file has the file name prefix that you specified on the XML Target tab and an index that identifies the item from which the output was created. In this case, top is automatically mapped to document_collection; therefore, one document will be created.
   b. Map top/Regroup:Employees in the Source list to the employee item in the Target list, and then click Auto Map.
   c. Using automatic mapping, you create the majority of the mappings that you need, but you also create some mappings that you do not need. The Job list item contains items that you do not need and that were automatically mapped by choosing the best candidate for each item from the Regroup:Employees list. You need to clear these mappings because you do not need them. If you did not remove this mapping, a new file would be created for each employee item in the input data. Select the Job item in the Target list, and then click Clear Mappings to clear all of the mapped child items.
   d. Evaluate the mappings to confirm that they are what you want. Notice that ns0:dateOfBirth in the Target list is mapped to top/Regroup:Employees/Input:HireDate in the Source list. This mapping is incorrect. Click the arrow on top/Regroup:Employees/Input:HireDate, and then select top/Regroup:Employees/Input:BirthDate.
e. Make sure to map all the fields correctly and they are available in the output. If you see the incorrect mapping in the Source column, click the pencil icon in the Source list for that item, and then click More to display a list of choices. Click and choose the appropriate field from the drop-down list to map the item.

Results

The following figure shows the results of the mappings:

<table>
<thead>
<tr>
<th>Source</th>
<th>Result</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../Employees</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/Fname</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/Lname</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/Gnder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/BirthDate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/Title</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../openAddress</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/Hrs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/Oth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/Country</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/Stk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/County</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/PostalCode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Click to select)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../Address_Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../input/Email</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Click to select)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../EmployeeID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../DependentID</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The step Output contains a new node, named XML_Composer:result. This node carries the path and file name for the output file during runtime.

Example 2: Viewing the output of the job

After you run the job, look at the output.

Procedure

1. From the Assembly Editor, click OK to return to the XML stage editor. Then click OK to close the XML stage editor.
2. From the IBM InfoSphere DataStage and QualityStage Designer client, choose File > Compile to compile the job.
3. Choose File > Run to run the job.
4. In the parameter window, for the first parameter, Root of example folder tree, enter the path of the directory where you have unzipped your examples zip file. For example, if you have downloaded and saved the examples zip file in C:\Examples folder, enter this directory as the Value for the first parameter, Root of example folder tree.
5. Open the employee_output.xml file to look at the results, which are shown below. The resulting file contains a list of employees. For each employee, the home and office addresses are shown. The type of address, home or office, is identified by the address_type tag, which displays O for office address and H for home address.
<?xml version="1.0" encoding="UTF-8"?>
<tns:employees xmlns:tns="Http://ibm.com/infosphere/xml/Employee">
  <tns:employee employeeID="B6540" departmentID="A100">
    <name>
      <firstName>Cynthia</firstName>
      <middleName>P</middleName>
      <lastName>Donald</lastName>
    </name>
    <gender>female</gender>
    <dateOfBirth>1987-01-17</dateOfBirth>
    <title>Miss</title>
    <Address>
      <street>San Felipe, Suite 2400</street>
      <city>Houston</city>
      <state>Texas</state>
      <country>USA</country>
      <postalCode>77057</postalCode>
      <address_type>O</address_type>
    </Address>
    <Address>
      <street>53rd West Street</street>
      <city>Houston</city>
      <state>Texas</state>
      <country>USA</country>
      <postalCode>77000</postalCode>
      <address_type>H</address_type>
    </Address>
    <hireDate>2007-07-25</hireDate>
  </tns:employee>
  <tns:employee employeeID="B6540" departmentID="A100">
    <name>
      <firstName>Tania</firstName>
      <middleName>G</middleName>
      <lastName>William</lastName>
    </name>
    <gender>female</gender>
    <dateOfBirth>1980-01-17</dateOfBirth>
    <title>Miss</title>
    <Address>
      <street>5th South Street</street>
      <city>Miami</city>
      <state>Florida</state>
      <country>USA</country>
      <postalCode>32599</postalCode>
      <address_type>O</address_type>
    </Address>
    <Address>
      <street>54th South Street</street>
      <city>Miami</city>
      <state>Florida</state>
      <country>USA</country>
      <postalCode>32501</postalCode>
      <address_type>H</address_type>
    </Address>
    <hireDate>2007-07-25</hireDate>
  </tns:employee>
  <tns:employee employeeID="A8990" departmentID="A100">
    <name>
      <firstName>Zen</firstName>
      <middleName>P</middleName>
      <lastName>Wright</lastName>
    </name>
    <gender>male</gender>
    <dateOfBirth>1980-04-4</dateOfBirth>
    <title>Mr</title>
    <Address>
      <street>2301 East Lamar Blvd</street>
    </Address>
  </tns:employee>
</tns:employees>
Example 3: Using the XML Composer and HJoin steps

Build this job that uses the XML stage and the XML Composer and HJoin steps to create a hierarchical structure.

About this task

This basic composing example uses two relational data files, employee.txt and address.txt, and the sample schemas, Employee.xsd and Organization.xsd, to illustrate composing source data from two relational files into a XML file. The employee.txt file contains information about the employees of one department in a company: employee name, date of hire, date of birth, gender, title, department ID, and employee ID. The second file contains employee address information: street, city, state, postal code, country, address type (home or office), and employee ID. Each employee has two addresses, a home address and an office address. In this example, you use an HJoin step in the assembly to join the two tables. Then you use the XML Composer step to compose a XML file that contains each employee's information, followed by the employee's two addresses.

The completed example job, xml_hjoin_example.dsx, is also available. To look at the completed example or use it as the starting point for expanding the example, import it into IBM InfoSphere DataStage.

To create the example, complete these steps:

Example 3: Creating the job

Build this job that uses the XML stage and the XML Composer and HJoin steps to compose data into a hierarchical structure.

About this task

The following figure shows the job that you create for the composing example. The job includes two Sequential File stages that are named Employee_File and Address_File. These Sequential File stages are linked to an XML stage by two links that are named Employee and Address. In this example, instead of using a Join stage in the job to join the files into a list and a Regroup step in the assembly to join the files into a nested list, you use the HJoin step in the assembly to perform both actions.
**Procedure**

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the **Repository** pane, right-click the **Jobs** folder, and select **New > Parallel job**.
3. Open the **File** section of the palette, and drag two Sequential File stages to the canvas.
4. Open the **Real Time** section of the palette, and drag one XML stage to the canvas. Position the XML stage to the right of Sequential File stages.
5. Perform the following steps to create a job property for the location of the example files:
   a. Choose **Edit > Job Properties**.
   b. On the **Parameters** tab, in the **Parameter Name** field, enter `xml_example_root_folder` and then specify the following values:

   **Table 33. Specifying the values for field names**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Root of example folder tree</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Point to the root of the file tree folder</td>
</tr>
</tbody>
</table>

   Each example uses this job parameter.
6. Create a link from each Sequential File stage to the XML stage.
7. Rename the stages and links as shown in the following table:

   **Table 34. Names for job elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential File stage</td>
<td>Address_File</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Employee_File</td>
</tr>
<tr>
<td>Link from Address_File to XML</td>
<td>Address</td>
</tr>
<tr>
<td>Link from Employee_File to XML</td>
<td>Employee</td>
</tr>
</tbody>
</table>

8. Configure the Employee_File stage. Double-click the Employee_File stage to open the stage properties. On the **Properties** tab of the Output page, configure the following properties:

   **Table 35. Configuring the properties**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Enter the path and file name of the file. For the Employee_File stage, the file name is employee.txt.</td>
</tr>
<tr>
<td>Read Method</td>
<td>Select <strong>Specific File(s)</strong>.</td>
</tr>
<tr>
<td>Field name</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>First Line is Column Names</td>
<td>Select True.</td>
</tr>
</tbody>
</table>

9. Click the **Columns** tab, and create these columns:

### Table 36. Columns for the Employee_File stage

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>MiddleName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>FirstName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>Gender</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>BirthDate</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>HireDate</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>EmployeeID</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>DepartmentID</td>
<td>VarChar</td>
<td></td>
</tr>
</tbody>
</table>

10. Configure the Address_File stage. Double-click the Address_File stage to open the stage properties. On the **Properties** tab of the Output page, configure the following properties:

### Table 37. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Enter the path and file name of the file. For the Address_File stage, the file name is address.txt.</td>
</tr>
<tr>
<td>Read Method</td>
<td>Select Specific File(s).</td>
</tr>
<tr>
<td>First Line is Column Names</td>
<td>Select True.</td>
</tr>
</tbody>
</table>

11. Click the **Columns** tab, and create these columns:

### Table 38. Columns for the Address_File stage

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Street</td>
<td>VarChar</td>
</tr>
<tr>
<td>City</td>
<td>VarChar</td>
</tr>
<tr>
<td>State</td>
<td>VarChar</td>
</tr>
<tr>
<td>Country</td>
<td>VarChar</td>
</tr>
<tr>
<td>PostalCode</td>
<td>VarChar</td>
</tr>
<tr>
<td>Address_Type</td>
<td>VarChar</td>
</tr>
<tr>
<td>EmployeeID</td>
<td>VarChar</td>
</tr>
</tbody>
</table>

12. Choose **File > Save**, and name the job `xml_hjoin_example`.

**Example 3: Creating the assembly**

Use the HJoin step to create a parent-child hierarchy from a two lists; then use the XML Composer step to compose the XML file.
Before you begin

Make sure that you already imported the example schemas Employee.xsd and Organization.xsd. If you have already built Example 1 or Example 2, you have imported the schemas as part of that example. For more information about importing the schemas, see Importing the schemas for the examples.

Procedure

1. Double-click the XML stage to open the stage properties, and then click Edit assembly to open the Assembly Editor.
2. Open the Palette, and then double-click the HJoin step to add it to the assembly.
3. Select the HJoin step, and then from the palette, double-click the XML_Composer step to add it below the HJoin step.
4. On the Overview, in the Description field, enter the following description: This assembly uses the Organization.xsd and Employee.xsd schemas. The HJoin step creates a hierarchical structure, where the employee information is the parent and the employee address information is the child. The XML_Composer step composes the data and saves the output in a file named employee_output.
5. This assembly uses the default Input step, which shows the columns that the two input links, Employee and Address, pass to the XML stage. You cannot modify the input columns, but you can display them as either links or a tree. The following figure shows the structure that the Input step provides as step Input to the Hjoin step. The two lists are at the same level in the hierarchy.

Example 3: Configuring the HJoin step

Use the HJoin step to create a parent-child hierarchy from a single list.

About this task

The step Input to the HJoin step contains two lists, Employee and Address. When you configure the HJoin step, you identify which list is the parent list and which
list is the child list, and you specify the keys to use to join the two lists together. Wherever the value of the key item is the same in both lists, the HJoin step joins the lists.

**Procedure**

1. On the **Configuration** tab of the HJoin step, configure these fields:

   **Table 39. Configuring the fields**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent List</td>
<td>Select top/InputLinks/Employee.</td>
</tr>
<tr>
<td>Child List</td>
<td>Select top/InputLinks/Address.</td>
</tr>
</tbody>
</table>

2. In the **Optimization Type** field, choose **In-memory**.
3. In the **Parent Keys** column, select `top/InputLinks/Employee/EmployeeID`.
4. In the **Child Keys** column, select `top/InputLinks/Address/EmployeeID`.

**Results**

The step Output shows the addition of the child list Address to the Employee list. The step Output also contains a new node named HJoin:orphans, which contains items that do not match the keys. The following figure shows this portion of the step Output.

---

**Example 3: Configuring the XML Composer step**

Configure the XML Composer step to create a XML file that lists each employee once, followed by the employee’s two addresses.

**Procedure**

1. In the **XML Target** window of the XML Composer step, choose **Write to File**. For the **Output Directory**, click **Insert Parameter**, and then select `xml_example_root_folder`. For the **Filename Prefix**, enter `employee_output`.
2. In the **Document Root** window, select the schema element that describes the documents that the step composes:
a. Click Browse, and then open the Schemas_for_XML_examples library, which is the library into which you imported the Organization.xsd and Employee.xsd schemas.
b. Click to open the Employee.xsd schema, click the root element employees, and then click OK. The Document Root tab displays the structure of the schema.

3. On the Validation tab, select Minimal validation.

4. In the HJoin step, you combined two lists and established a hierarchical structure where the address items are in a list that is a child of the employee list. Now on the Mappings tab, map the items from the target schema to the items in the Employee list. The target schema contains some items that you will not use. For example, the target schema contains a list named Job, which includes items such as jobTitle and startDate. You do not need mappings to items in the Job list because the Regroup:Employees list does not contain any job items. To create the mappings, perform the following steps:

a. The mapping for the document_collection item determines when a new XML document is created. If you map the top element of the step Input to document_collection, one document is created. If you map a list item to document_collection, one document is created for each item in the list. The name of each resulting file has the file name prefix that you specified on the XML Target tab and an index that identifies the item that the output was created from.
b. Map top/InputLinks/Employee in the Source list to the employee item in the Target list, and then click Automap.
c. Using automatic mapping, you create the majority of the mappings that you need, but you also create some mappings that you do not need. The Job list item contains items that you do not need and that were automatically mapped by choosing the best candidate for each item from the top/InputLinks/Employees list. You need to clear these mappings because you do not need them. Select the Job item in the Target list, and then click Clear Mappings to clear all of the mapped child items.
d. Evaluate the rest of the mappings to confirm that they are what you want. Notice that ns0:dateOfBirth in the Target list is mapped to top/InputLinks/Employee/Input:HireDate in the Source list. This mapping is incorrect. Click the arrow on top/InputLinks/Employee/Input:HireDate, and then select top/InputLinks/Employee/Input:BirthDate/
e. Make sure to map all the fields correctly and they are available in the output. If you see the incorrect mapping in the Source column, click the pencil icon in the Source list for that item, and then click More to display a list of choices. Click and choose the appropriate field from the drop-down list to map the item.
Results

The following figure shows the results of the mappings:

The step Output contains a new node, named XML_Composer:result. This node carries the path and file name for the output file during runtime.

Example 3: Viewing the output of the job
After you finish building the assembly, run the job and look at the output.

Procedure
1. From the Assembly Editor, click OK to return to the XML stage editor. Then click OK to close the XML stage editor.
2. From the IBM InfoSphere DataStage and QualityStage Designer client, choose File > Compile to compile the job.
3. Choose File > Run to run the job.
4. In the parameter window, for the first parameter, Root of example folder tree, enter the path of the directory where you have unzipped your examples zip file. For example, if you have downloaded and saved the examples zip file in C:\Examples folder, enter this directory as the Value for the first parameter, Root of example folder tree.
5. Open the employee_output.xml file to look at the results, which are shown below. The resulting file contains a list of employees. For each employee, the home and office addresses are shown. The type of address, home or office, is identified by the address_type tag, which displays O for office address and H for home address.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:employees xmlns:tns="Http://ibm.com/infosphere/xml/Employee">
  <tns:employee employeeID="B6540" departmentID="A100">
    <name>
      <firstName>Cynthia</firstName>
      <middleName>P</middleName>
      <lastName>Donald</lastName>
    </name>
    <gender>female</gender>
    <dateOfBirth>1987-01-17</dateOfBirth>
  </tns:employee>
</tns:employees>
```
<title>Miss</title>
<Address>
  <street>San Felipe, Suite 2400</street>
  <city>Houston</city>
  <state>Texas</state>
  <country>USA</country>
  <postalCode>77057</postalCode>
</Address>
<Address>
  <street>53rd West Street</street>
  <city>Houston</city>
  <state>Texas</state>
  <country>USA</country>
  <postalCode>77000</postalCode>
</Address>
<hireDate>2007-07-25</hireDate>
</tns:employee>
<tns:employee employeeID="B6540" departmentID="A100">
  <name>
    <firstName>Tania</firstName>
    <middleName>G</middleName>
    <lastName>William</lastName>
  </name>
  <gender>female</gender>
  <dateOfBirth>1980-01-17</dateOfBirth>
  <title>Miss</title>
  <Address>
    <street>5th South Street</street>
    <city>Miami</city>
    <state>Florida</state>
    <country>USA</country>
    <postalCode>32599</postalCode>
  </Address>
  <Address>
    <street>54th South Street</street>
    <city>Miami</city>
    <state>Florida</state>
    <country>USA</country>
    <postalCode>32501</postalCode>
  </Address>
  <hireDate>2007-07-25</hireDate>
</tns:employee>
<tns:employee employeeID="A8990" departmentID="A100">
  <name>
    <firstName>Zen</firstName>
    <middleName>P</middleName>
    <lastName>Wright</lastName>
  </name>
  <gender>male</gender>
  <dateOfBirth>1980-04-4</dateOfBirth>
  <title>Mr</title>
  <Address>
    <street>2301 East Lamar Blvd</street>
    <city>Arlington</city>
    <state>Texas</state>
    <country>USA</country>
    <postalCode>76013</postalCode>
  </Address>
  <Address>
    <street>2001 West Street</street>
    <city>Arlington</city>
    <state>Texas</state>
  </Address>
</tns:employee>
Example 4: Using the XML Parser and Switch steps

Create a job that uses the External Source Stage, XML stage, and two sequential files. The Switch step is used to filter the data.

About this task

This example uses the sample XML data files, employee1.xml and employee2.xml, and the sample schemas, Employee.xsd and Organization.xsd, to illustrate how to filter the data based on the predefined conditions. The employee1.xml and employee2.xml files contain information about the employees in a department. In this example, you use External Source stage to read the files. Then you use the XML Parser step and the Switch step within the XML stage, where the Parser step validates the XML files and the Switch step filters the files based on the predefined conditions. The output files contain the employee information: last name, middle name, first name, gender, birth date, title, hire date, employee ID and department ID.

Note: In this example, the employee2.xml file is invalid. The employee2.xml file contains the data for the element dateOfBirth as 17-01-1987. The default format for the date is YYYY-MM-DD. As the data does not follow this format, it is invalid. In the XML Parser step, in Validation tab, there is a rule called, Data type has an illegal value which checks if the value specified for an element is valid. This rule is used in this example to validate the data.

After you complete all the steps in the assembly, the Assembly Editor will look as shown in the figure below:
The completed example job, *xml_switch_example.dsx*, is also available. To look at the completed example or use it as the starting point for expanding the example, import it into IBM InfoSphere DataStage.

To create the example, complete these tasks:

**Example 4: Creating the job**
Create the Switch example job that includes one External Source stage, one XML stage, and two Sequential File stages.

**About this task**

The following figure shows the job that you create for this Switch example. The job includes one External Source stage named Employee_Files, one XML stage named Employee_Filter, and two Sequential File stages, named Employee_Validfile and Employee_Invalidfile. The External Source stage is linked to the XML stage by a link named Employee, and the XML stage is linked to two sequential files, Employee_Validfile and Employee_Invalidfile by a link named Valid_file and Invalid_file respectively.
Procedures

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and select New > Parallel job.
3. Open the File section of the palette, and drag one External Source stage to the canvas.
4. Open the Real Time section of the palette, and drag one XML stage to the canvas. Position the XML stage to the right of the External Source stage.
5. Open the File section of the palette, and drag two sequential file stages to the canvas. Position the two sequential file stages to the right of the XML stage.
6. Create a link from the External Source stage to the XML stage, and create links from the XML stage to the two sequential file stages.
7. Rename the stages and links as shown in the following table:

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Source stage</td>
<td>Employee_Files</td>
</tr>
<tr>
<td>XML stage</td>
<td>Employee_Filter</td>
</tr>
<tr>
<td>Link from External Source stage to XML stage</td>
<td>Employee</td>
</tr>
<tr>
<td>First Sequential File Stage</td>
<td>Employee_Validfile</td>
</tr>
<tr>
<td>Second Sequential File Stage</td>
<td>Employee_Invalidfile</td>
</tr>
<tr>
<td>Link from XML stage to Employee_Validfile</td>
<td>Valid_file</td>
</tr>
<tr>
<td>Link from XML stage to Employee_Invalidfile</td>
<td>InValid_file</td>
</tr>
</tbody>
</table>

8. Create job properties for the location of the example files:
   a. Click the Job Properties icon on the toolbar.
b. On the Parameters tab, in the Parameter Name field, enter xml_example_root_folder, and then specify the following values for the first parameter.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Root of example folder tree</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Point to the root of the file tree folder</td>
</tr>
</tbody>
</table>

Table 41. Specifying the values for field names

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Location of XML files</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>ls &lt;Enter_the_location_of_the_zip&gt;/xml_examples/switch_step/example1/input_files/*.xml</td>
</tr>
<tr>
<td>Help Text</td>
<td>The location of the input files.</td>
</tr>
</tbody>
</table>

Table 42. Specifying the values for field names

d. Click OK.

9. Double-click the External Source stage to configure the stage properties.

10. Click the Properties tab, and select Source > Source Program. Enter the path for the input files, in the Source Program field. You can use the parameter, file_location to specify the location of the input files.

   **Note:** The External Source stage returns the absolute location of the xml files. For example, if your data files (employee1.xml and employee2.xml) are in the C:\Test\ directory, the output of the External Source stage will be C:\Test\employee1.xml and C:\Test\employee2.xml.

11. Click the Columns tab, and create these columns:

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>employee_file_list</td>
<td>VarChar</td>
</tr>
</tbody>
</table>

Table 43. Columns for the Employee_Files stage

12. Click OK.

13. By looking at the input file, employee1.xml, which contains the XML data, you determine which columns to create to hold the employee data that the XML stage will pass to the Employee_Validfile stage and the Employee_Invalidfile stage.

14. Double-click the Employee_Validfile stage to configure the stage properties.

15. Click the Properties tab, and configure the following properties that define the output file:

   **Table 44. Configuring the properties**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Enter the path to where you want the output file to be created, followed by the file name for example, valid_emp.txt.</td>
</tr>
</tbody>
</table>
Table 44. Configuring the properties (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose <strong>Overwrite</strong> to create the file.</td>
</tr>
<tr>
<td>First line is column name</td>
<td>Set to <strong>True</strong>.</td>
</tr>
</tbody>
</table>

16. Click the **Columns** tab, and create these columns:

Table 45. Columns for the Employee_Validfile stage

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>MiddleName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>FirstName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>Gender</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>BirthDate</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>HireDate</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>EmployeeID</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>DepartmentID</td>
<td>VarChar</td>
<td></td>
</tr>
</tbody>
</table>

17. Click **OK** to close the Employee_Validfile stage.

18. Double-click the Employee_Invalidfile stage to configure the stage properties.

19. Click the **Properties** tab, and configure the following properties that define the output file:

Table 46. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Enter the path to where you want the output file to be created, followed by the file name for example, InValid_emp.txt.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose <strong>Overwrite</strong> to create the file.</td>
</tr>
<tr>
<td>First line is column name</td>
<td>Set to <strong>True</strong>.</td>
</tr>
</tbody>
</table>

20. Click the **Columns** tab, and create these columns:

Table 47. Columns for the Employee_Invalidfile stage

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>MiddleName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>FirstName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>Gender</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>BirthDate</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>HireDate</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>EmployeeID</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>DepartmentID</td>
<td>VarChar</td>
<td></td>
</tr>
</tbody>
</table>

21. Click **OK** to close the Employee_Invalidfile stage.
22. Choose File > Save, and name the job as xml_switch_example.

Example 4: Creating the assembly
Use the XML Parser stage to parse the XML files, the Switch step to separate input files into valid and invalid files, and the Output step to map the data to the sequential files.

Before you begin
Import the example schemas employee.xsd and organization.xsd. If you have already built any of the previous examples, you have imported the schemas as part of that example. For more information about importing the schemas, see Importing the schemas for the examples.

Procedure
1. Double-click the XML stage to open the stage properties.
2. Click the Edit assembly to open the Assembly Editor.
3. Open the Palette, and then double-click the XML_Parser Step to add it to the Assembly Outline.
4. Select the XML_Parser Step, and then from the palette, double-click the Switch Step to add it below the XML_Parser Step in the Assembly Outline.
5. On the Overview, in the Description field, enter the following description: This assembly uses the Organization.xsd and Employee.xsd schemas. The XML_Parser Step reads xml files, the Switch Step filters the files, and the Output Step maps the files into two sequential files.

Example 4: Configuring the XML Parser step
Configure the location of the XML source data and the schema that you want to use.

Procedure
1. Click the XML Parser step in the Assembly Outline. By default, the Configuration tab for the step opens. The following figure shows the Configuration tab for the step:

2. On the XML Source tab, specify the location of the XML source data. In this example, the XML source data is in multiple files, employee1.xml and employee2.xml. Select File set, and choose the value from the list. The value is
top/InputLinks/Employee/employee_file_list where employee_file_list is the column specified in the External Source stage.

3. On the **Document Root** tab, select the schema element that describes the documents that the step parses.
   
   a. Click **Browse**, and then open the Schemas_for_XML_examples library, which is the library into which you imported the Organization.xsd and Employee.xsd schemas.
   
   b. Click to open the Employee.xsd schema, click the root element employees, and then click **OK**. The **Document Root** tab displays the structure of the schema. The step Output also displays this same structure.

4. On the **Validation** tab, select **Strict Validation**. By selecting **Strict Validation**, you automatically ensure that the data types conversion is performed. If you use the default **Minimal Validation**, all data types are automatically set to the String type.

5. In **Value Validation**, set the *Data type has an illegal value* rule to 'Reject'. When you select the action as 'Reject', the Output schema displays two elements named success and message in XML_Parser: ParserStatus. If the XML file is valid, the 'success' will hold the parsing status as 'True' and if the XML file is invalid, it holds the value 'False'. The 'message' will hold the reason for the parsing failure if the 'success' is 'False'. The **Data type has an illegal value** rule checks if the values specified for each of the elements in the XML file is valid and within the specified range for that data type.

**Note:** In this example, the employee2.xml file is invalid. The employee2.xml file contains the data for the element dateOfBirth as 17-01-1987. The default format for the date is YYYY-MM-DD. As the data does not follow this format, it is invalid and the 'success' will hold the value 'False'.

The following figure shows the Output Schema of the Parser step:
Example 4: Configuring the Switch step

Use the Switch step to separate the input files into valid and invalid files.
About this task

With the Switch step you classify elements from the input files and place them into new target lists. Each target list is associated with the criteria that you specify. You can use multiple criteria in the target list. The data that passes the criteria is placed in the corresponding target list and the default list will contain the data that fails all the other criteria.

Procedure

1. On the **Configuration** tab of the Switch step, configure these fields:

   **Table 48. Configuring the fields**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>List to Categorize</td>
<td>Select top/InputLinks/Employee. This is the list that contains the items on which you specify the criteria.</td>
</tr>
<tr>
<td>Scope</td>
<td>Choose top.</td>
</tr>
</tbody>
</table>

2. Select **Add Target**. In the New Target Constraint window, configure these fields:
   - Specify the **Target Name**. In this example you specify the name as valid_file.
   - Select the element in the **Filter Field** as 'success'.
   - Select the function as 'IsTrue'.

Results

The Output step contains a new node named Switch:filtered, which contains two lists. The valid data is placed in the valid_file list and the invalid data is placed in the default list. The following figure shows this portion of the step Output.
Example 4: Configuring the Output step

In the Output step, create mappings that define how to map source items in one data structure to target items in another data structure.

About this task

In this assembly, you use the Output step to map a hierarchical data structure to a relational data structure.

Procedure

1. Click the Output Step in the Assembly Outline to open the step. The step displays the Configuration tab in the Output Step window. In the Output window, the output table describes the data structure that is the output of the assembly. The following figure shows the relevant columns of the output table for this assembly:

![Output Table Screenshot]

The default view in the Output window is the Links view. The Links view looks similar to the table that displays on the Columns tab in the XML stage editor. The output structure for Valid_file link is shown above. In the Links view, you can modify the columns that you already defined. Any changes that you make to the columns are propagated to the column definitions in the Sequential file.

2. From the Output Links drop-down list, select Invalid_file. The output structure for Invalid_file link is shown below.
3. Click the **Mappings** tab. The following figure shows the mapping table. In this table, each item in the output structure is represented by a row in the table. You map target items to source items. For this job, the target structure has two links, `Valid_file` and `Invalid_file`, and the columns that are defined on those links. In the Target column, the `Valid_file` and `Invalid_file` links are represented as a list item and each column displays as a content item.
4. To create mappings, first map target list items to source list items. Then map target content items to source content items. Select the `Valid_file` list item in the Target list and then in the Source select from the drop-down list,

```
top/Switch:filtered/valid_file/XML_Parser:result/tns:employees/employee
```

and click **Auto Map**.

The following figure shows the results of the automatic mapping for the `Valid_file`.
5. Next you need to map the Invalid_file list item. Select the Invalid_file list item in the Target list and then in the Source select from the drop-down list, `top/ Switch:filtered/default/ XMLParser:result/ tns:employees/ employee` and click Auto Map. The following figure shows the results of the automatic mapping for the Invalid_file.

![Output Step Diagram]

When you map the Source items from the Suggestion list, you see only the relative path. To view the entire path you can hover over each element.

6. Click OK to close the assembly editor.
7. Click OK to close the Stage Editor.
8. Save the job.

**Example 4: Viewing the output of the job**
After you run the switch job, open the text files, and look at the output.
**Procedure**

1. From the IBM InfoSphere DataStage and QualityStage Designer client, choose **File > Compile** to compile the job.
2. Choose **File > Run** to run the job.
3. In the parameter window, for the first parameter, **Root of example folder tree**, enter the path of the directory where you have unzipped your examples zip file. For example, if you have downloaded and saved the examples zip file in C:\Test folder, enter this directory as the **Value** for the first parameter, **Root of example folder tree**.
4. For the second parameter, after ls, change only the text within braces to specify the path to the directory where you saved the example files. For example, if you have placed the example files in C:\Test folder, specify ls C:\Test\xml_examples\switch_step\example1\input_files\*.xml.
5. Click **Run**.
6. After the job runs, open the Valid_file.txt file and the Invalid_file.txt file to look at the results.

The Valid_file.txt file contains this data:

"firstName","middleName","lastName","gender","dateOfBirth","title","hireDate", "employeeID","departmentID"
"Zen","P","Wright","male","1980-04-04","Mr","2008-07-11","A8990","A100"

The Invalid_file.txt file contains this data:

"firstName","middleName","lastName","gender","dateOfBirth","title","hireDate", "employeeID","departmentID"
"Cynthia","P","Donald","female","17-01-1987","Miss","2000-07-25","B6540","A100"

**Example 5: Using the XML Parser and Union steps**

Create a job that uses the XML stage and a Sequential file stage. The Union step is used to combine two lists into a single list.

**About this task**

This example uses the sample XML data files, employee1.xml and employee2.xml, and multiple sample schemas, to illustrate how to combine the two lists into a single list that has a predefined structure. The employee1.xml and employee2.xml files contain information about the employees in a department. In this example, use the XML Parser steps within the XML stage to parse the files. Then you use the Union step within the XML stage, to combine two lists into a single file. The XML Composer step is used to compose a XML file that contains the employee information. The output step maps the employee information to a single column in the sequential file.

After you complete all the steps in the assembly, the Assembly Editor will look as shown in the figure below:
The completed example job, xml_union_example.dsx, is also available. To look at the completed example or use it as the starting point for expanding the example, import it into IBM InfoSphere DataStage.

To create the example, complete these tasks:

**Example 5: Creating the job**
Create the job that includes a XML stage, and a Sequential File stage.

**About this task**

The following figure shows the job that you create for the Union example. The job includes a XML stage named Employees, and a Sequential File stage named Employees_File. The XML stage is linked to a Sequential file stage by a link named Employee_Union.
Procedure
1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and select New > Parallel job.
3. Open the Real Time section of the palette, and drag XML stage to the canvas.
4. Create job properties for the location of the example files:
   a. Choose the Edit > Job Properties.
   b. On the Parameters tab, in the Parameter Name field, enter xml_example_root_folder and then specify the following values:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Root of example folder tree</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Point to the root of the file tree folder</td>
</tr>
</tbody>
</table>

5. Open the File section of the palette, and drag a Sequential File stage to the canvas. Position this stage to the right of the XML stage.
6. Create a link from the XML stage to the Sequential file stage.
7. Rename the stages and link as shown in the following table:

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML stage</td>
<td>Employees</td>
</tr>
<tr>
<td>Link from XML stage to Sequential File stage</td>
<td>Employee_Union</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Employees_File</td>
</tr>
</tbody>
</table>

8. Double-click the Employees_File stage to configure the stage properties.
9. Click the Properties tab, and configure the following properties that define the output file:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Enter the path where you want the output file to be created, followed by the file name employees_output.xml.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose Overwrite to create the file.</td>
</tr>
</tbody>
</table>

10. Click the Format tab. Right-click the field Quote = double, and select Remove.
11. For the Employees_File stage, do not create any columns now. You will later use the Assembly Editor to propagate the columns automatically.
12. Click OK to close the Employees_File stage.
13. Choose File > Save, and name the job as xml_union_example.

Example 5: Importing the schemas for the Union example
Use the Schema Library Manager to import the example schemas into the library.
About this task

You must import the schemas that the example job uses. Imported schemas are stored in the metadata repository, where they are available for use in any assembly that you create.

Procedure

1. Double-click the XML stage to open the stage properties, and then click Edit assembly to open the Assembly Editor.
2. From the Assembly Editor, click Libraries tab to open the Schema Library Manager. You can either use Schema Library Manager from the Assembly Editor or choose Import > Schema Library Manager from the IBM InfoSphere DataStage and QualityStage Designer client.
3. To create a library for the example schemas, click New Library. In the New Contract Library window, specify Schemas_for_XML_example for the library name and Example for the category. The library is now categorized under Example.

Note: If you have already built the previous examples, the library Schemas_for_XML_example has been created and you need not perform the step 3.
4. Similarly, create two more libraries, Schemas_for_XML_example1 and Schemas_for_XML_example2 with the categories Example1, and Example2 respectively. The library is now categorized under Example1, and Example2.
5. Expand the category Example, select the library, Schemas_for_XML_example, and then click Import New Resource. Browse to find the Employee.xsd schema. Select the schema and then click OK. Repeat this process to import the Organization.xsd schema. Similarly, import Employee1.xsd and Organization1.xsd for the library, Schemas_for_XML_example1, and Employee2.xsd and Organization2.xsd for the library, Schemas_for_XML_example2.

The following schemas are located in the respective folders as shown:

Table 52. Schema location

<table>
<thead>
<tr>
<th>Schema</th>
<th>Folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee.xsd and Organization.xsd</td>
<td>xml_examples\schemas</td>
</tr>
<tr>
<td>Employee1.xsd, Organization1.xsd, and Employee2.xsd, Organization2.xsd</td>
<td>xml_examples\union_step\example1\union_schemas</td>
</tr>
</tbody>
</table>

6. If you have imported only one schema for example, Employee.xsd and have not imported the other schema, Organization.xsd, then an error message is displayed indicating that the library has failed validation. The error indicates that the Employee.xsd schema cannot resolve the element JobType that has been defined in the Organization.xsd schema, which you have not yet imported. Repeat the import process to import the Organization.xsd schema. The library passes the validation.

Note: Similar issues might appear for the other schemas, Employee1.xsd, Organization1.xsd, and Employee2.xsd, Organization2.xsd, if you have imported only one schema among these.

Note: Multiple .xsd files can be imported at a time into the library. This can be done by selecting multiple files in the browse window on clicking Import New Resource.
7. Click **OK** to close the Assembly Editor.
8. Click **OK** to close the Stage Editor.
9. Choose **File > Save**.

**Example 5: Creating the assembly**

In the assembly, you add the Parser, Union, and Composer steps where the Parser
parses the xml files, Union combines the files into a single file and the Composer
composes the XML file.

**Before you begin**

You have already imported the schemas in the previous step, Example 5: importing
the schemas for the Union example.

**Procedure**

1. Double-click the XML stage to open the stage properties, and then click **Edit
   assembly** to open the Assembly Editor.
2. Open the Palette, and then double-click the XML_Parser Step to add it to the
   Assembly Outline.
3. Select the XML_Parser Step, and then from the palette, double-click to add
   another parser step below XML_Parser Step. This time it will be named as
   XML_Parser_1 Step.
4. Select the XML_Parser_1 Step, and then from the palette, double-click the
   Union Step to add it below the XML_Parser_1 Step.
5. Select the Union Step, and then from the palette, double-click the
   XML_Composer Step to add it below the Union Step.
6. On the Overview, in the **Description** field, enter the following description:
   In this assembly, the XML_Parser Step reads xml files, the Union Step
   combines the two files, the Composer Step composes the xml file, and the
   Output Step writes the xml file into a sequential file.

**Example 5: Configuring the first XML Parser step**

Configure the location and the schema of the first XML source data that you want
to parse.

**Procedure**

1. Click the XML_Parser step in the Assembly Outline. By default, the
   **Configuration** tab for the step opens. The following figure shows the
   **Configuration** tab for the step:
2. On the **XML Source** tab, you specify the location of the XML source data. In this example, the XML source data is in a single file, `employee1.xml`. Select **Single file** and then click **Insert Parameter**, and select the `xml_example_root_folder` parameter. You need to specify the exact location of the file when you run the job. Enter the complete location of the XML input file as, `#xml_example_root_folder#/xml_examples/union_step/example1/input_files/employee1.xml`.

3. On the **Document Root** tab, you select the schema element that describes the documents that the step parses. Click **Browse**. Open the `Schemas_for_XML_example1` library, which is the library into which you have imported the `Organization1.xsd` and `Employee1.xsd` schemas.

4. Click to open the `Employee1.xsd` schema, click the root element `Employee_List`, and then click **OK**. The **Document Root** tab displays the structure of the schema. The step **Output** also displays this same structure.

5. On the **Validation** tab, select **Strict Validation**. By selecting **Strict Validation**, you automatically ensure that the data types conversion is performed. If you use the default **Minimal Validation**, all data types are automatically converted to the **String** type.
Example 5: Configuring the second XML Parser step

Configure the location and the schema of the second XML source data that you want to parse.

Procedure

1. Click the XML_PARSER_1 Step in the Assembly Outline. By default, the Configuration tab for the step opens. The following figure shows the Configuration tab for the step:

![XML_PARSER_1 Step](image)

2. On the XML Source tab, you specify the location of the XML source data. In this example, the XML source data is in a single file, employee2.xml. Select Single file and then click Insert Parameter, and select the xml_example_root_folder parameter. You need to specify the exact location of the file when you run the job. Enter the complete location of the XML input file as,#xml_example_root_folder#/xml_examples/union_step/example1/input_files/employee2.xml.

3. On the Document Root tab, you select the schema element that describes the documents that the step parses. Click Browse. Open the
Schemas_for_XML_example2 library, which is the library into which you have imported the Organization2.xsd and Employee2.xsd schemas.

4. Click to open the Employee2.xsd schema, click the root element Dep_Employees_List, and then click OK. The Document Root tab displays the structure of the schema. The step Output also displays this same structure.

5. On the Validation tab, select Strict Validation. By selecting Strict Validation, you automatically ensure that the data types conversion is performed. If you use the default Minimal Validation, all data types are automatically set the String type.

Example 5: Configuring the Union step

Use the Union step to combine two lists to create a single list.

About this task

In the Union step you select the target schema definition from the Contract Libraries and then map the target schema to the source schema in the mapping table.

Procedure

1. On the Configuration tab of the Union step, configure these fields:
   - **Union Type** – Click Browse to open the library, Schema_for_XML_example.
   - Expand Employee, and select the root element, employee.
   - Click OK.

   The Union Type tab displays the structure of the employee schema.

2. Click the Mappings tab.

3. To create mappings, first map target list items to source list items.

4. On the Source, corresponding to the left list, click to choose EmployeeInfo from the Suggestions list or drop-down list.

5. Click Auto Map. The Employee and Address information is mapped for the left list. The following figure shows the result of Auto Map for the left list. The target left list item and all of its child items are automatically mapped to source items.
6. You will see that the target item, dateOfBirth is wrongly mapped to the source item, hireDate. You need to update this entry to create the correct mapping. Click hireDate in the Source list to choose the correct item from the drop-down list. The items in the list appear in order, from highest to lowest, based on their mapping scores. Select DOB as the right candidate for the target item, dateOfBirth from the drop-down list.

7. You will also see that the address fields are not mapped correctly in the Source column. Make sure to map all the fields correctly and they are available in the
output. If you see the incorrect mapping in the Source column, click the pencil icon in the Source list for that item, and then click More to display a list of choices. Click and choose the appropriate field from the drop-down list to map the item. Following table shows the target items and the corresponding source items that you need to manually map to:

Table 53. Corresponding source items to the target items

<table>
<thead>
<tr>
<th>Source items</th>
<th>Target items</th>
</tr>
</thead>
<tbody>
<tr>
<td>street_name</td>
<td>street</td>
</tr>
<tr>
<td>city_name</td>
<td>city</td>
</tr>
<tr>
<td>state_name</td>
<td>state</td>
</tr>
<tr>
<td>country_name</td>
<td>country</td>
</tr>
</tbody>
</table>

8. On the Source, corresponding to the right list, click to choose Dept_Employee from the Suggestions list or drop-down list.

9. Click Auto Map. The Employee and Address information is mapped for the right list.

As seen above, you can map two different schema structures to a common schema structure in the Union step. The common schema structure is the one that you define in the Union Type tab. In the Mappings tab, the left and right lists have this common structure and you can map each of these lists to different schema structures in the Source view. In this example, two different schema structures are used in the parser. In the first parser, the EmployeeInfo list contains the employee information and in the second parser, Dept_employee list also contains the employee information. You can map the EmployeeInfo list to the left list and Dept_employee list to the right list in the Mappings tab. The following figure shows this view.
<table>
<thead>
<tr>
<th>Source</th>
<th>Result</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...type_list:EmplyeeInfos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...0:EMP_Name:firstName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...0:EMP_Name:middleName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...0:EMP_Name:lastName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:gender</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:dateOfBirth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:title</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...fields:isnull:EMP_Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...P_Address:street</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...P_Address:city</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...P_Address:state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...Address:country</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...P_Address:postcode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...Address:phone</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...idNumber:exttext:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>..._Address:address_type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:isnull:hireDate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...fields:isnull:job</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:isnull:employeeID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:isnull:departmentID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...list:Dept_employee</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...0:Emp_firstName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...0:Emp_middleName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...0:Emp_lastName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:gender</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:dateOfBirth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:title</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...fields:isnull:Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...nullAddress:add_street</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...nullAddress:add_city</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...nullAddress:add_state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...nullAddress:country</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...nullAddress:postcode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...nullAddress:phone</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...nullAddress:address_type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:isnull:hireDate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:isnull:job</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:employeeID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...info:departmentID</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results

The step output contains a new node named Union:result. The structure of the Union:result is same as the structure that is defined in the Union Type. The following figure shows this portion of the step output.

Example 5: Configuring the XML Composer step
Configure the XML Composer step to create a XML file of the employees in a department.
Procedure

1. In the XML Target window of the XML Composer step, choose Pass as String. This passes the composed XML string to a downstream step or stage for further processing.

2. In the Document Root window, select the schema element that describes the documents that the step composes:
   a. Click Browse, and then open the Schemas_for_XML_examples library, which is the library into which you imported the Organization.xsd and Employee.xsd schemas.
   b. Click to open the Employee.xsd schema, click the root element employees, and then click OK. The Document Root tab displays the structure of the schema.

3. On the Validation tab, by default Strict validation is selected. You need not make any changes to the fields in this tab.

4. In the Union step, you combined two lists into a single hierarchical list. Now in the Composer step, on the Mappings tab, map the items from the target schema to the items in the Union:result list. To create the mappings, perform the following steps:
   a. Map top/Union:result in the Source list to the employee item in the Target list, and then click Automap.
   b. Using automatic mapping, you map each item from the Union:result list to the employee list.
   c. Evaluate all the items to see if the correct mapping is done. It is seen that the mapping is done correctly.

5. On the Format tab, default encoding is UTF-8. You need not change this field.

6. Select the Format Style and configure the following fields.
   a. In the Indentation length field, select the number of characters to use for each indentation level in the XML output.
   b. In the New Line Style field, select the type of new line. Choices are UNIX (LF), DOS(CRLF) or MAC(CR).

Results

The step Output contains a new node, named XML_Composer:result. This node carries the entire hierarchical data during runtime. The following figure shows the output of the XML Composer step.

Example 5: Configuring the Output step

In the Output step, you create the mappings that define how to map target items in one data structure to source items in another data structure.

About this task

In this assembly, you use the Output step to map the output of the composer to a sequential file.
**Procedure**

1. Click the Output Step in the Assembly Outline to open the step. The step displays the **Configuration** tab in the Output Step window.

![Output Step window](image)

The default view in the Output window is the Links view. The output structure for Employee_Union link is shown. The above figure shows that the output table does not display any columns because when you created the job, you did not define any columns for the Employee_Union link. The lack of columns is not an error. However, if no columns are defined on the link, you cannot map any source items in the hierarchical data structure to this link.

The composer output, result-string is a single column that contains the employee data and you need to create that single column in this step. But instead of returning to the job and manually creating the column, you can automatically create it from the **Mappings** tab of the Output step.

2. Click the **Mappings** tab. The following figure shows the mapping table. In this table, each item in the output structure is represented by a row in the table. You map target items to source items. For this job, the target structure has a single link, Employee_Union, and no columns are defined for this link. In the Target column, the Employee_Union link is represented as a list item.

![Mapping table](image)

3. Next you need to map the Employee_Union list item. Select the Employee_Union item. For this item, in the Source, select top from the Suggestion list, and click Propagate. The Propagate automatically creates one column for each item that is a descendent of the top item in the source structure. When you use the Propagate button to create items, the items are automatically mapped. The following figure shows the result.
4. Click OK to close the assembly editor.
5. Click OK to close the Stage Editor.
6. Save the job.

**Example 5: Viewing the output of the job**
After you run the Union job, open the text file, and look at the output.

**Procedure**
1. From the IBM InfoSphere DataStage and QualityStage Designer client, choose File > Compile to compile the job.
2. Choose File > Run to run the job.
3. In the parameter window, for the first parameter, **Root of example folder tree**, enter the path of the directory where you have unzipped your examples zip file. For example, if you have downloaded and saved the examples zip file in directory, C:\Examples, enter this as the Value for the first parameter, **Root of example folder tree**.
4. Click Run.
5. After the job runs, open the employees_output.xml file to look at the results.

The employees_output.xml file contains this data:
```xml
<?xml version="1.0" encoding="UTF-8"?>
  <prn:employee employeeID="B6540" departmentID="A100">
    <name>
      <firstName>Cynthia</firstName>
      <middleName>P</middleName>
      <lastName>Donald</lastName>
    </name>
    <gender>female</gender>
    <dateOfBirth>1987-01-17</dateOfBirth>
    <title>Miss</title>
    <Address>
      <street>San Felipe Suite 2400</street>
      <city>Houston</city>
      <state>Texas</state>
      <country>USA</country>
      <postalCode>77057</postalCode>
      <address_type>O</address_type>
    </Address>
    <Address>
      <street>53rd West Street</street>
    </Address>
  </prn:employee>
</prn:employees>
```
Example 6: Using the XML Composer and H-Pivot steps

Create a job that uses a Sequential file stage and the XML stage. The H-Pivot step is used to combine multiple fields into a single list.

About this task

This example is used to convert the relational file, address.txt into a XML file. The address.txt contains address information of the employees in an organization. Using the H-Pivot step you can map multiple fields into a single list. Use the XML Composer step to create a hierarchical structure that contains employee address information. The output step maps the address information to a single XML file.

The job includes one Sequential File stage, named Address_file, and a XML stage, named Address_compose. The Sequential file stage is linked to a XML stage by a link named Address.

After you complete all the steps in the assembly, the Assembly Editor will look as shown in the figure below:
The completed example job, xml_hpivot_example.dsx, is also available. To look at the completed example or use it as the starting point for expanding the example, import it into IBM InfoSphere DataStage.

To create the example, complete these tasks:

**Example 6: Creating the job**

Create the example job that includes one Sequential File stage and a XML stage.

**About this task**

The following figure shows the job that you create for the H-Pivot example. The job includes one Sequential File stage named Address_file, and a XML stage named Address_compose. The Sequential file stage is linked to a XML stage by a link named Address.
Procedure

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the **Repository** pane, right-click the **Jobs** folder, and select **New > Parallel job**.
3. Open the **File** section of the palette, and drag a Sequential File stage to the canvas.
4. Open the **Real Time** section of the palette, and drag XML stage to the canvas. Position this stage to the right of the Sequential File stage.
5. Perform the following steps to create a job property for the location of the example files:
   a. Choose **Edit > Job Properties**.
   b. On the **Parameters** tab, in the **Parameter Name** field, enter `xml_example_root_folder` and then specify the following values:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Root of example folder tree</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Point to the root of the file tree folder</td>
</tr>
</tbody>
</table>

   Each example uses this job parameter.
6. Create a link from the Sequential File stage to the XML stage.
7. Rename the stages and links as shown in the following table:

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential File stage</td>
<td>Address_file</td>
</tr>
<tr>
<td>XML stage</td>
<td>Address_compose</td>
</tr>
<tr>
<td>Link from Sequential File stage to XML stage</td>
<td>Address</td>
</tr>
</tbody>
</table>

8. Click **OK**.
9. Configure the Address_file stage. Double-click the Address_file stage to open the stage properties.
10. Click the **Properties** tab, and configure the following properties that define the output file:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source File File</td>
<td>Enter the path and file name of the file. For the Address_file stage, the file name is address.txt</td>
</tr>
<tr>
<td>Source Read Method</td>
<td>Select Specific File</td>
</tr>
<tr>
<td>Options First Line is Column Names</td>
<td>Select True</td>
</tr>
</tbody>
</table>

11. Click the **Columns** tab, and create these columns:

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>street</td>
<td>VarChar</td>
</tr>
</tbody>
</table>
### Table 57. Columns for the Address_file stage (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>city</td>
<td>VarChar</td>
</tr>
<tr>
<td>state</td>
<td>VarChar</td>
</tr>
<tr>
<td>country</td>
<td>VarChar</td>
</tr>
<tr>
<td>postalCode</td>
<td>VarChar</td>
</tr>
<tr>
<td>phonenumber1</td>
<td>VarChar</td>
</tr>
<tr>
<td>phonenumber2</td>
<td>VarChar</td>
</tr>
<tr>
<td>address_type</td>
<td>VarChar</td>
</tr>
</tbody>
</table>

12. Click **OK** to close the stage properties.

13. Choose **File > Save**, and name the job as `xml_hpivot_example`.

**Example 6: Creating the assembly**

Use the H-Pivot step to combine multiple fields into a single list.

**Before you begin**

Make sure that you have already imported the example schemas `Employee.xsd` and `Organization.xsd`. If you have built the XML Parser example, you have imported the schemas as part of that example. For more information about importing the schemas, see [Importing the schemas for the examples](#).

**Procedure**

1. Double-click the XML stage to open the stage properties, and then click **Edit assembly** to open the Assembly Editor.
2. Open the Palette, and then double-click the H-Pivot Step to add it to the Assembly Outline.
3. Select the H-Pivot Step, and then from the palette, double-click the XML_Composer step to add it below the H-Pivot step.
4. On the Overview, in the **Description** field, enter the following description: This assembly uses the `Organization.xsd` and `Employee.xsd` schemas. In the H-Pivot step you combine multiple fields into a list. The XML_Composer step composes the data and saves the output in a file named `address_output`.

**Example 6: Configuring the H-Pivot step**

Use the H-Pivot step to transform fields or items into a list.

**About this task**

With the H-Pivot step you combine multiple fields into a list. This helps in mapping multiple fields into a single list in the composer.

**Procedure**

1. On the **Configuration** tab of the H-Pivot step, configure the field:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Choose Address.</td>
</tr>
</tbody>
</table>
Note: The list that you select as the Scope contains the fields that need to be transformed into a list.

2. Click and select the field, phonenumber1 from the drop-down list.
3. Click and select the field, phonenumber2 again from the drop-down list.

Results

The following figure shows the output of the H-pivot on the Output tab.

![H-Pivot Result](image)

The output of the H-Pivot contains the list named rows. The rows include two fields, name and value. The name represents the name of the field that you have defined in the H-Pivot and the value holds the data for the corresponding names. In this example, the field name contains phonenumber1 and phonenumber2, and the field value contains the data related to the name.

Example 6: Configuring the XML Composer step

Configure the XML Composer step to create a XML file containing address information.

Procedure

1. In the XML Target window of the XML Composer step, choose Write to File. For the Output Directory, click Insert Parameter, and then select xml_example_root_folder. For the Filename Prefix, enter address_output.
2. In the Document Root window, select the schema element that describes the documents that the step composes:
   a. Click Browse, and then open the Schemas_for_XML_examples library, which is the library into which you imported the Organization.xsd and Employee.xsd schemas.
   b. Click to open the Employee.xsd schema, click the root element addresses, and then click OK. The Document Root tab displays the structure of the schema.
3. On the Validation tab, by default Strict validation is selected. You need not make any changes to the fields in this tab.
4. On the Mappings tab, map the items from the target schema to the items in the address list. To create the mappings, perform the following steps:
   a. Map top/InputLinks/Address in the Source list to the address item in the Target list, and then click Automap.
   b. Using automatic mapping, you create the majority of the mappings that you need, but you also create some mappings that you do not need. For example, the name is mapped to e2res:text(), which is not a correct mapping and you need to change this to value.
   c. The following figure shows the mapping.
5. Notice that the H-Pivot:result/rows is mapped to the list, phoneNumber. In H-Pivot step, you have combined the fields, phoneNumber1 and phoneNumber2 into a single list, rows and the data is stored in the field, value. Hence the H-pivot step allows you to map two relational fields in the input to a single list in the composer.
Results

The following figure shows the results of the mapping:

Example 6: Viewing the output of the job
After you finish building the assembly, run the job and look at the output.

Procedure
1. From the IBM InfoSphere DataStage and QualityStage Designer client, choose File > Compile to compile the job.
2. Choose File > Run to run the job.
3. In the parameter window, for the first parameter, Root of example folder tree, enter the path of the directory where you have unzipped your examples zip file. For example, if you have downloaded and saved the examples zip file in the directory, C:\Examples, enter this as the Value for the first parameter, Root of example folder tree.
4. Click Run.

5. After the job runs, open the address_output.xml file to look at the results. The address_output.xml file contains this data:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:addresses xmlns:tns="http://ibm.com/infosphere/xml/Employee">
  <tns:address>
    <street>2301 East Lamar Blvd</street>
    <city>Arlington</city>
    <state>Texas</state>
    <country>USA</country>
    <postalCode>78363</postalCode>
    <phoneNumber>1-234-234-4567</phoneNumber>
    <phoneNumber>1-234-234-4566</phoneNumber>
    <address_type>O</address_type>
  </tns:address>
  <tns:address>
    <street>2001 West Street</street>
    <city>Arlington</city>
    <state>Texas</state>
    <country>USA</country>
    <postalCode>78300</postalCode>
    <phoneNumber>1-876-456-4321</phoneNumber>
    <phoneNumber>1-876-456-4320</phoneNumber>
    <address_type>H</address_type>
  </tns:address>
  <tns:address>
    <street>San Felipe Suite 2400</street>
    <city>Houston</city>
    <state>Texas</state>
    <country>USA</country>
    <postalCode>77057</postalCode>
    <phoneNumber>1-965-234-1234</phoneNumber>
    <phoneNumber>1-965-234-1235</phoneNumber>
    <address_type>O</address_type>
  </tns:address>
  <tns:address>
    <street>53rd West Street</street>
    <city>Houston</city>
    <state>Texas</state>
    <country>USA</country>
    <postalCode>77000</postalCode>
    <phoneNumber>1-000-000-8765</phoneNumber>
    <phoneNumber>1-000-000-8764</phoneNumber>
    <address_type>H</address_type>
  </tns:address>
  <tns:address>
    <street>5th South Street</street>
    <city>Miami</city>
    <state>Florida</state>
    <country>USA</country>
    <postalCode>32500</postalCode>
    <phoneNumber>1-999-999-0989</phoneNumber>
    <phoneNumber>1-999-999-0979</phoneNumber>
    <address_type>O</address_type>
  </tns:address>
  <tns:address>
    <street>54th South Street</street>
    <city>Miami</city>
    <state>Florida</state>
    <country>USA</country>
    <postalCode>32501</postalCode>
    <phoneNumber>1-567-654-7890</phoneNumber>
    <phoneNumber>1-567-654-7891</phoneNumber>
    <address_type>H</address_type>
  </tns:address>
</tns:addresses>
```

Example 7: Using the XML Parser and Aggregate steps

In this example, create a job that uses the XML stage and a Sequential file stage. The Aggregate step is used to perform hierarchical aggregations on the items in a list. For example, use it to concatenate the data, count the number of items in the list, and look for the first and last element in the list of items.

About this task

This example uses the sample XML data file, departments.xml, and the sample schemas, Employee.xsd and Organization.xsd, to illustrate parsing source data from one XML file into a sequential file. The departments.xml file contains information about the employees of various departments in a company. In this example, you parse the XML data into a sequential file. The file contains employee business and address information. The employee business information contains the details such as employee name, date of hire, date of birth, gender, title, employee ID, and department ID. The employee address information contains employee ID, street, city, state, postal code, country, and address type.

After you complete all the steps in the assembly, the Assembly Editor will look as shown in the following figure.

The completed example job, xml_aggregate_example.dsx, is also available. To look at the completed example or use it as the starting point for expanding the example, import it into IBM InfoSphere DataStage.

To create the example, complete these tasks:

**Step 7.1: Creating the job**
Create the job that includes an XML stage and a Sequential File stage.
About this task

The following figure shows the job that you create for the Aggregate example. The job includes an XML stage named Employees and a Sequential File stage named Employee_File. The XML stage is linked to a Sequential file stage by a link named Employees_Aggregate.

Procedure

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and select New > Parallel job.
3. Open the Real Time section of the palette, and drag XML stage to the canvas.
4. Create job properties for the location of the example files:
   b. On the Parameters tab, in the Parameter Name field, enter xml_example_root_folder and then specify the following values:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Points to the location of the example folder</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Points to the location of the example folder</td>
</tr>
</tbody>
</table>

5. Open the File section of the palette, and drag a Sequential File stage to the canvas. Position this stage to the right of the XML stage.
6. Create a link from the XML stage to the Sequential file stage.
7. Rename the stages and link as shown in the following table:

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML stage</td>
<td>Employees</td>
</tr>
<tr>
<td>Link from XML stage to Sequential File stage</td>
<td>Employees_Aggregate</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Employee_File</td>
</tr>
</tbody>
</table>

8. Double-click the Employee_File stage to configure the stage properties.
9. Click the Properties tab, and configure the following properties that define the output file:
Table 61. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Enter the path where you want the output file to be created, followed by the file name employees_output.xml.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose Overwrite to create the file.</td>
</tr>
<tr>
<td>First Line is Column Names</td>
<td>Choose True.</td>
</tr>
</tbody>
</table>

For the Employee_File stage, do not create any columns now. You let the Assembly Editor to propagate the columns automatically.

10. Click OK to close the Employees_File stage.

11. Choose File > Save, and name the job as xml_aggregate_example.

**Step 7.2: Creating the assembly**

Use the XML Parser stage to parse the XML files, the Aggregate step to perform hierarchical aggregations on the items in a list, and the Output step to map the data to the sequential file.

**Before you begin**

Import the example schemas employee.xsd and organization.xsd. If you have already built any of the previous examples, you have imported the schemas as part of that example. For more information about importing the schemas, see [Importing the schemas for the examples](#).

**Procedure**

1. Double-click the XML stage to open the stage properties.
2. Click the Edit assembly button to open the Assembly Editor.
3. Open the Palette, and then double-click the XML_Parser Step to add it to the Assembly Outline.
4. Select the XML_Parser Step, and then from the palette, double-click the Aggregate Step to add it below the XML_Parser Step in the Assembly Outline.
5. On the Overview, in the Description field, enter the following description: This assembly uses the Organization.xsd and Employee.xsd schemas to read the input XML file. In this example, the Aggregate Step counts the number of employees in various departments.

**Step 7.3: Configuring the XML Parser step**

Configure the location of the XML source data and the schema that you want to use.

**Procedure**

1. In the Assembly Outline, click the XML Parser step. By default, the Configuration tab for the step opens. The following figure shows the Configuration tab for the step:
2. On the **XML Source** tab, specify the location of the XML source data. In this example, the XML source data is in the single file `departments.xml`. Select **Single file**, and then click **Insert Parameter** and select the `xml_example_root_folder` parameter. You will specify the exact location of the file when you run the job. You can also specify the absolute path to `departments.xml`.

3. On the **Document Root** tab, select the schema element that describes the documents that the step parses.
   a. Click **Browse**, and then open the Schemas_for_XML_examples library, which is the library into which you imported the `Organization.xsd` and `Employee.xsd` schemas.
   b. Click to open the namespace, `http://ibm.com/infosphere/xml/` Organization, click the root element `department`, and then click **OK**. The **Document Root** tab displays the structure of the schema. The step Output also displays this same structure.

4. On the **Validation** tab, select **Strict Validation**. By selecting **Strict Validation**, you automatically ensure that the data types conversion is performed. If you use the default **Minimal Validation**, all data types are treated as String type. The following figure shows the output schema of the Parser step:
Step 7.4: Configuring the Aggregate step

Use the Aggregate step to perform hierarchical aggregations on the items in a list.
About this task

In the Aggregate step you specify the input list on which the aggregation is performed. You can also specify the Aggregation key for the input list to be grouped and to perform the aggregation on all the groups separately rather than on the whole list.

Procedure

1. On the Configuration tab of the Aggregate step, configure these fields:

   Table 62. Configuring the fields
   
<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>List to Aggregate</td>
<td>Choose employee.</td>
</tr>
<tr>
<td>Scope</td>
<td>Choose top.</td>
</tr>
</tbody>
</table>

   The Scope in the Aggregate step defines the scope of the Aggregate function. The elements are aggregated within the list that is selected as the scope. The Scope also determines the location of the output of the Aggregate step.

   The Scope should always be the parent node of the list selected in the List to Aggregate field. In this example, because the List to Aggregate is selected as employee, you can only select the scope as either top or department. This is because there are two lists existing above the employee list which is department and top.

   If you choose the scope as top, the Aggregation step looks into the elements within the top list and performs the aggregation. For example, if the scope is set as top and the item for aggregation is employeeID, the Aggregation step checks for all the employeeIDs within the top list and then performs the aggregation. Therefore the employeeID is checked for within the entire input file and aggregated. The following figure shows the step result output under the top list.

   If you choose the scope as department, the Aggregation step looks for items within each department one at a time and does the aggregation. For example, if the scope is department and the item for aggregation is ‘employeeID’, the Aggregation step checks for the employeeID’s within the first department and move on to the next department until all the departments are checked for or
covered and performs the aggregation. Therefore the employeeID is aggregated within each department. The following figure shows step result output when the scope is selected as department.

2. There are two tabs located on the Configuration tab. They are Aggregation Items and Aggregation keys.
   - In Aggregation Items tab, you specify the list on which the aggregation is performed. For example, the Aggregation Item may be employeeID.
   - In Aggregation Keys tab, you specify the key on which the input list is grouped. For example, the key can be departmentID.

3. Select the Aggregation Items tab.
   - From the drop-down list, select employeeID. The employeeID is the unique key that is specified for each employee.
   - From the drop-down list, select Count. The count function counts the number of employees in a department. There are multiple functions for example, first, last, and concatenate that you can make use of based on your requirement.

4. Select the Aggregation keys tab.
   - From the drop-down list, select departmentID. The departmentID is the key on which the employee details are aggregated.

If you choose the List to Aggregate field as employee and set the scope as top, and selected the Aggregation item as employeeID, and Aggregation function as count, on Aggregation, the Aggregation step checks for all the employees within top list and counts the number of employees. Similarly if you select the Aggregation key as departmentID, the Aggregation step checks for the employees within top list, group them based on the departmentID and counts the number of employees in each group.

If you choose the List to Aggregate field as employee and set the scope as department, and have selected the Aggregation item as employeeID and Aggregation function as count, on Aggregation, the Aggregation step checks for all the employees within each department and counts the number of
employees. Note that the checks are not performed from the top node as the criterion set in the scope is department. Similarly if you select the Aggregation key as departmentID, the Aggregation step checks for the employees within the department list, group them based on the departmentID and counts the number of employees in each group.

**Results**

The following figure shows the output of the Aggregate on the Output tab.

The output of the Aggregate step contains a new list named Aggregate:result, which contains two nodes; aggregate and Aggregate_Input:keys. The employeeID_CNT holds the count of the number of employees in a department. The Aggregate_Input:keys holds the aggregation keys which contains the departmentID.

**Step 7.5: Configuring the Output step**

In the Output step, create mappings that define how to map source items in one data structure to target items in another data structure.

**About this task**

In this example, you use the Output step to map the output of the composer to a sequential file.

**Procedure**

1. In the Assembly Outline, click the Output Step to open the step. The step displays the Configuration tab in the Output Step window. In the Output window, the output table describes the data structure that is the output of the assembly. The following figure shows the relevant columns of the output table for this assembly:
The default view in the Output window is the Links view. The output structure for Employee_Aggregate link is shown. The previous figure shows that the output table does not display any columns because when you created the job, you did not define any columns for the Employee_Aggregate link. The lack of columns is not an error. However, if no columns are defined on the link, you cannot map any source items in the hierarchical data structure to this link. Instead of returning to the job and manually creating the column, you can automatically create it from the Mappings tab of the Output step.

2. Click the Mappings tab. The following figure shows the mapping table. In this table, each item in the output structure is represented by a row in the table. You map target items to source items. For this job, the target structure has a single link, Employee_Aggregate, and no columns are defined for this link. In the Target column, the Employee_Aggregate link is represented as a list item.

3. Map the Employee_Aggregate list item. Select the Employee_Aggregate item. For this item, in the Source, from the Suggestion list, select Aggregate:result, and click Propagate. One column for each item that is a descendent of the Aggregate:result item is created in the source structure. When you use the Propagate button to create items, the items are automatically mapped. The following figure shows the result:
4. Click **OK** to close the assembly editor.
5. Click **OK** to close the Stage Editor.
6. Save the job.

**Step 7.6: Viewing the output of the job**

After you run the Aggregate job, open the text file, and look at the output.

**Procedure**

1. From the IBM InfoSphere DataStage and QualityStage Designer client, choose **File > Compile** to compile the job.
2. Choose **File > Run** to run the job.
3. In the parameter window, for the first parameter, **Root of example folder tree**, enter the path of the directory where you have extracted the files from your examples compressed file, examples.zip. For example, if you have downloaded and saved the examples.zip file in the directory, C:\Examples, enter this directory as the **Value** for the first parameter, **Root of example folder tree**.

The following is a part of the input file that contains the employee information of a single department:

```
<department xmlns="" departmentID="A100" departmentKind="Division">
  <manager>A7100</manager>
  <ns1:employees>
    <ns1:employee employeeID="A8990" departmentID="A100">
      <name>
        <firstName>Zen</firstName>
        <middleName>P</middleName>
        <lastName>Wright</lastName>
      </name>
      <gender>male</gender>
      <dateOfBirth>1980-04-04</dateOfBirth>
      <title>Mr</title>
      <Address>
        <street>2301 East Lamar Blvd</street>
        <city>Arlington</city>
        <state>Texas</state>
        <country>USA</country>
        <postalCode>78363</postalCode>
      </Address>
    </ns1:employee>
  </ns1:employees>
</department>
```
4. After the job runs, open the employees_output.txt file to look at the results. The employees_output.txt file contains this data:

   "employeeID_CNT","departmentID"
   "3","A100"
   "3","A101"

As you see in the output file, the Aggregate step has counted the number of employees in each department in the input file under the top element.

**Example 8: Using the XML Composer and Sort steps**

In this example, you create a job that uses the XML stage and a Sequential file stage. The Sort step is used to sort the items in a list either in ascending or descending order.
About this task

This example uses the sample XML data file, departments.xml, and the sample schemas, Employee.xsd and Organization.xsd. The departments.xml file contains information about the employees of various departments in a company. This file contains employee business and address information. The employee business information contains the details such as employee name, date of hire, date of birth, gender, title, employee ID, and department ID. The employee address information contains employee ID, street, city, state, postal code, country, and address type.

In this example, you parse the input XML file and sort the employee list in ascending order based on the hire date. After this, you compose the sorted output into a XML file.

After you complete all the steps in the assembly, the Assembly Editor will look as shown in the following figure.

The completed example job, xml_sort_example.dsx, is also available. To look at the completed example or use it as the starting point for expanding the example, import it into IBM InfoSphere DataStage.

To create the example, complete these tasks:

**Step 8.1: Creating the job**
Create the job that includes an XML stage and a Sequential File stage.

About this task

The following figure shows the job that you create for the Sort example. The job includes an XML stage named Employees and a Sequential File stage named Employee_File. The XML stage is linked to a Sequential file stage by a link named Employees_Sort.
Procedure

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.

2. In the Repository pane, right-click the Jobs folder, and select New > Parallel job.

3. Open the Real Time section of the palette, and drag XML stage to the canvas.

4. Create job properties for the location of the example files:
   b. On the Parameters tab, in the Parameter Name field, enter xml_example_root_folder and then specify the following values:

   Table 63. Specifying the values for field names

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Points to the location of the example folder</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Points to the location of the example folder</td>
</tr>
</tbody>
</table>

5. Open the File section of the palette, and drag a Sequential File stage to the canvas. Position this stage to the right of the XML stage.

6. Create a link from the XML stage to the Sequential file stage.

7. Rename the stages and link as shown in the following table:

   Table 64. Names for job elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML stage</td>
<td>Employees</td>
</tr>
<tr>
<td>Link from XML stage to Sequential File stage</td>
<td>Employees_Sort</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Employee_File</td>
</tr>
</tbody>
</table>

8. Double-click the Employee_File stage to configure the stage properties.

9. Click the Properties tab, and configure the following properties that define the output file:
Table 65. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Select the xml_example_root_folder parameter and enter the path as xml_examples/sort_step/example1/output/employees_output.xml.</td>
</tr>
<tr>
<td></td>
<td>The output file, employees_output.xml is created in the location #xml_example_root#/xml_examples/sort_step/example1/output.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose Overwrite to create the file.</td>
</tr>
</tbody>
</table>

10. Click the Format tab. Right-click the fields, Quote=double and Delimiter = comma and select Remove.
    For the Employee_File stage, do not create any columns now. You let the Assembly Editor to propagate the columns automatically.
11. Click OK to close the Employees_File stage.
12. Choose File > Save, and name the job as xml_sort_example.

Step 8.2: Creating the assembly
Use the XML Parser stage to parse the XML files, the Sort step to perform sort on the items in a list either in ascending or descending order, and the Composer to compose the XML file.

Before you begin
Import the example schemas employee.xsd and organization.xsd. If you have already built any of the previous examples, you have imported the schemas as part of that example. For more information about importing the schemas, see Importing the schemas for the examples.

Procedure
1. Double-click the XML stage to open the stage editor.
2. Click the Edit assembly button to open the Assembly Editor.
3. Open the Palette, and then double-click the XML_Parser Step to add it to the Assembly Outline.
4. Select the XML_Parser Step, and then from the palette, double-click the Sort Step to add it below the XML_Parser Step in the Assembly Outline.
5. Select the Sort Step, and then from the palette, double-click the XML_Composer Step to add it below the Sort Step in the Assembly Outline.
6. On the Overview, in the Description field, enter the following description: This assembly uses the Organization.xsd and Employee.xsd schemas. The sort step sorts the employee list in ascending order based on hiredate.

Step 8.3: Configuring the XML Parser step
Configure the location of the XML source data and the schema that you want to use.

Procedure
1. In the Assembly Outline, click the XML Parser step. By default, the Configuration tab for the step opens. The following figure shows the Configuration tab for the step:
2. On the **XML Source** tab, specify the location of the XML source data. In this example, the XML source data is in the single file named `departments.xml`. Select **Single file**, and then click **Insert Parameter** and select the `xml_example_root_folder` parameter. Along with this, you need to enter the path as `#xml_example_root_folder#/sort_step/example1/input_files/departments.xml`. When you run the job, you have to specify the exact location of the example folder where you unzipped your examples zip file.

3. On the **Document Root** tab, select the schema element that describes the documents that the step parses.
   a. Click **Browse**, and then open the `Schemas_for_XML_examples` library, which is the library into which you imported the `Organization.xsd` and `Employee.xsd` schemas.
   c. Click **OK**. The **Document Root** tab displays the structure of the schema. The step Output also displays this same structure.

4. On the **Validation** tab, select **Strict Validation**. By selecting **Strict Validation**, you automatically ensure that the data types conversion is performed. If you use the default **Minimal Validation**, all data types are treated as String type.

The following figure shows the output schema of the Parser step:
Step 8.4: Configuring the Sort step
Use the Sort step to sort the items in the list in ascending or descending order.
About this task

In the Sort step you specify the input list on which the sort is performed. You can also specify the Sort key for the list to be sorted.

Procedure

1. On the Configuration tab of the Sort step, configure these fields:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>List to Sort</td>
<td>Choose employee.</td>
</tr>
<tr>
<td>Scope</td>
<td>Choose department.</td>
</tr>
</tbody>
</table>

The Scope in the Sort step defines the scope of the Sort function. The elements are sorted within the list that is selected as the scope. The Scope also determines the location of the output of the Sort step.

The Scope should always be the parent node of the list selected in the List to Sort field. In this example because the List to Sort is selected as employee, you can only select the scope as either top or department. This is because there are two lists existing above the employee list which is department and top.

If you choose the scope as top, the Sort step looks into the elements within the top list and performs the sort. For example, if the scope is set as top and the key selected is hireDate, the Sort step checks for all hireDates within the top list and then performs the sort. Therefore the hireDate is checked for within the entire input file and sorted. The following figure shows the step result output under the top list.
If you choose the scope as department, the Sort step looks for items within each department one at a time and does the sort within this scope. For example, if the scope is department and the key selected for sort is hireDate, the Sort step checks for the hireDate’s within the first department, sorts the data and move...
on to the next department until all the departments are checked for and sorted. Therefore the hireDate is sorted within each department. The following figure shows step result output when the scope is selected as department.

2. Select the **Keys** tab.
3. From the drop-down list, select hireDate or use the More option to find it.
4. Select the **Order** field. The default option set is Ascending.
Results

The following figure shows the output of the Sort on the Output tab.

![Diagram of Sort:output]

The step output contains a new node named Sort:result. The Sort:result contains the sorted employee list.

**Step 8.5: Configuring the XML Composer step**

Configure the XML Composer step to compose the sort step output into a XML file.

**Procedure**

1. In the **XML Target** window of the XML Composer step, choose **Pass as String**. This passes the composed XML string to a downstream step or stage for further processing.
2. In the **Document Root** window, select the schema element that describes the documents that the step composes:
   a. Click **Browse**, and then open the Schemas_for_XML_examples library, which is the library into which you imported the **Organization.xsd** and **Employee.xsd** schemas.
   c. Click **OK**. The **Document Root** tab displays the structure of the schema.
3. On the **Validation** tab, by default **Strict validation** is selected. You do not need to make any changes to the fields in this tab.
4. In the Sort step, you sorted the input file. Now in the Composer step, on the Mappings tab, map the items from the target schema to the items in the output schema coming from the previous steps. To create the mappings, perform the following steps:
   a. Map top/XML_Parser:result/org:departments/ns0:department in the Source list to the department in the Target list. Also manually map the manager.
   b. Next manually map the Sort:result/tns:employee in the source list to the employee in the target list, and then click Automap. The Automap automatically maps each item under Sort:result/tns:employee list with the items under the target employee list.
   c. You also need to manually map the items, annualBudget, departmentID, and departmentKind in the target list to the corresponding items from the XML Parser result in the source list.
   d. Review all the items to see if the correct mapping is done.
5. On the Format tab, default encoding is UTF-8. You do not need to change this field.
6. Select the Format Style and configure the following fields.
   a. In the Indentation length field, select the number of characters to use for each indentation level in the XML output. By default, number of characters is set to 4.
   b. In the New Line Style field, select the type of new line. Choices are UNIX (LF), DOS(CRLF) or MAC(CR). By default, the choice selected is DOS.

**Results**

The following figure shows the output of the XML Composer step:

![XML Composer:result](image)

The step Output contains a new node, named XML_Composer:result. This node carries the entire hierarchical data during runtime.

**Step 8.6: Configuring the Output step**

In the Output step, you create the mappings that define how to map target items in one data structure to source items in another data structure.

**About this task**

In this example, you use the Output step to map the output of the composer to a sequential file.

**Procedure**

1. In the Assembly Outline, click the Output Step to open the step. The step displays the Configuration tab in the Output Step window.
The default view in the Output window is the Links view. The output structure for Employees_Sort link is shown. The previous figure shows that the output table does not display any column because when you created the job, you did not define any column for the Employees_Sort link. The lack of column is not an error. However, if no column is defined on the link, you cannot map any source items in the hierarchical data structure to this link. Instead of returning to the job and manually creating the column, you can automatically create it from the Mappings tab of the Output step.

2. Click the Mappings tab. The following figure shows the mapping table. In this table, each item in the output structure is represented by a row in the table. You map target items to source items. For this job, the target structure has a single link, Employees_Sort, and no column is defined for this link. In the Target column, the Employees_Sort link is represented as a list item.

3. Map the Employees_Sort list. Select the Employees_Sort list. For this item, in the Source, from the Suggestion list, select top and click Propagate. One column for each item that is a descendent of the top item is created in the source structure. When you use the Propagate button to create items, the items are automatically mapped. The following figure shows the result:
4. Click OK to close the assembly editor.
5. Click OK to close the Stage Editor.
6. Save the job.

Step 8.7: Viewing the output of the job
After you run the Sort job, open the XML file, and look at the output.

Procedure
1. From the IBM InfoSphere DataStage and QualityStage Designer client, choose File > Compile to compile the job.
2. Choose File > Run to run the job.
3. In the parameter window, for the first parameter, Root of example folder tree, enter the path of the directory where you have extracted the files from your examples compressed file, examples.zip. For example, if you have downloaded and saved the examples.zip file in the directory, C:\Examples, enter this directory as the Value for the first parameter, Root of example folder tree.

The following is an input file that contains the employee information of a single department:

```xml
<departments xmlns:ns1="http://ibm.com/infosphere/xml/Employee"
xmlns="http://ibm.com/infosphere/xml/Organization"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <department xmlns="" departmentID="A100" departmentKind="Division">
    <manager>A7100</manager>
    <ns1:employees>
      <ns1:employee employeeID="A8990" departmentID="A100">
        <name>
          <firstName>Zen</firstName>
          <middleName>P</middleName>
          <lastName>Wright</lastName>
        </name>
        <gender>male</gender>
        <dateOfBirth>1980-04-04</dateOfBirth>
        <title>Mr</title>
        <Address>
          <street>2301 East Lamar Blvd</street>
          <city>Arlington</city>
          <state>Texas</state>
          <country>USA</country>
          <postalCode>78363</postalCode>
          <address_type>O</address_type>
```
4. After the job runs, open the employees_output.xml file to look at the results.

The employees_output.xml file contains this data:

```xml
  <department departmentID="A100" departmentKind="Division">
    <manager>A7100</manager>
      <tns:employee employeeID="B6540" departmentID="A100">
        <name>
          <firstName>Cynthia</firstName>
          <middleName>P</middleName>
          <lastName>Donald</lastName>
        </name>
        <gender>female</gender>
        <dateOfBirth>1987-01-17</dateOfBirth>
        <title>Miss</title>
        <Address>
          <street>54th South Street</street>
          <city>Miami</city>
          <state>Florida</state>
          <country>USA</country>
          <postalCode>32501</postalCode>
          <address_type>H</address_type>
        </Address>
        <hireDate>2002-07-25</hireDate>
        <Jobs>
          <job department="DV100" id="6000" isManager="false">
            <jobTitle>Associate Developer</jobTitle>
            <startDate>1998-01</startDate>
            <endDate>2002-07</endDate>
            <annualSalary>1000000</annualSalary>
          </job>
          <job department="A100" id="5000" isManager="false">
            <jobTitle>Software Developer</jobTitle>
            <startDate>2002-07</startDate>
            <endDate>2010-12</endDate>
            <annualSalary>2000000</annualSalary>
          </job>
          <job department="DV100" id="6000" isManager="false">
            <jobTitle>Associate Developer</jobTitle>
            <startDate>1998-01</startDate>
            <endDate>2002-07</endDate>
            <annualSalary>1000000</annualSalary>
          </job>
        </Jobs>
      </tns:employee>
    </tns:employees>
  </department>
</org:departments>
```
As you see in the output file, the Sort step has sorted the employee list based on the hire date.

**Example 9: Using the XML Composer and OrderJoin steps**

In this example, you create a job that uses the XML stage and a Sequential file stage. The OrderJoin step is used to merge the items in the employee and address lists into a single list based on their position.
**About this task**

This example uses the sample XML files, `employee.xml` and `address.xml`, and the sample schemas, `Employee.xsd` and `Organization.xsd`. The `employee.xml` file contains information such as name, gender, date of birth, and employee ID. The second file contains employee address information: street, city, state, postal code, country, and employee ID. In this example, you use the OrderJoin step to merge the employee and address list into a single list. Then you use the XML Composer step to compose a XML file that contains employee name and address details.

In this example, you parse the input XML file and sort the employee list in ascending order based on the hire date. After this, you compose the sorted output into a XML file.

After you complete all the steps in the assembly, the Assembly Editor will look as shown in the following figure.

The completed example job, `xml_orderjoin_example.dsx`, is also available. To look at the completed example or use it as the starting point for expanding the example, import it into IBM InfoSphere DataStage.

To create the example, complete these tasks:

**Step 9.1: Creating the job**
Create the job that includes an XML stage and a Sequential File stage.

**About this task**

The following figure shows the job that you create for the OrderJoin example. The job includes an XML stage named `Employees` and a Sequential File stage named `Employees_File`. The XML stage is linked to a Sequential file stage by a link named `Employees_OrderJoin`. 
Procedure

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and select New > Parallel job.
3. Open the Real Time section of the palette, and drag XML stage to the canvas.
4. Specify job properties for the location of the example files:
   b. On the Parameters tab, in the Parameter Name field, enter xml_example_root_folder and then specify the following values:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Points to the location of the example folder</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Points to the location of the example folder</td>
</tr>
</tbody>
</table>

5. Open the File section of the palette, and drag a Sequential File stage to the canvas. Position this stage to the right of the XML stage.
6. Create a link from the XML stage to the Sequential file stage.
7. Rename the stages and link as shown in the following table:

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML stage</td>
<td>Employees</td>
</tr>
<tr>
<td>Link from XML stage to Sequential File stage</td>
<td>Employees_OrderJoin</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Employees_File</td>
</tr>
</tbody>
</table>

8. Double-click the Employees_File stage to configure the stage properties.
9. Click the Properties tab, and configure the following properties that define the output file:
Table 69. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Select the xml_example_root_folder parameter and enter the path as xml_examples/orderjoin_step/example1/output/employees_output.xml. The output file, employees_output.xml is created in the location xml_example_root_folder/xml_examples/orderjoin_step/example1/output/employees_output.xml.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose Overwrite to create the file.</td>
</tr>
</tbody>
</table>

10. Click the Format tab. Right-click the fields, Quote=double and Delimiter = comma and select Remove.
    For the Employees_File stage, do not create any columns now. You let the Assembly Editor to propagate the columns automatically.
11. Click OK to close the Employees_File stage.
12. Choose File > Save, and name the job as xml_orderjoin_example.

**Step 9.2: Creating the assembly**

Use the OrderJoin step to merge the items in the employee and address lists into a single list based on their relative positions.

**Before you begin**

Import the example schemas employee.xsd and organization.xsd. If you have already built any of the previous examples, you have imported the schemas as part of that example. For more information about importing the schemas, see [Importing the schemas for the examples](#).

**Procedure**

1. Double-click the XML stage to open the stage editor.
2. Click the Edit assembly button to open the Assembly Editor.
3. Open the Palette, and then double-click the XML_Parser Step to add it to the Assembly Outline.
4. Select the XML_Parser Step, and then from the palette, double-click to add another parser step below XML_Parser Step. This time it will be named as XML_Parser_1 Step.
5. Select the XML_Parser_1 Step, and then from the palette, double-click the OrderJoin Step to add it below the XML_Parser_1 Step.
6. Select the OrderJoin Step, and then from the palette, double-click the XML_Composer Step to add it below the OrderJoin Step.
7. On the Overview, in the Description field, enter the following description: This assembly uses the Organization.xsd and Employee.xsd schemas. The OrderJoin step is used to merge the items in the employee and address lists into a single list based on their relative positions.

**Step 9.3: Configuring the first XML Parser step**

Configure the location and the schema of the first XML source data that you want to parse.
Procedure

1. In the Assembly Outline, click the XML Parser step. By default, the Configuration tab for the step opens. The following figure shows the Configuration tab for the step:

2. On the XML Source tab, specify the location of the XML source data. In this example, the XML source data is in the single file named employee.xml. Select Single file and click Insert Parameter, and select the xml_example_root_folder parameter. You need to specify the exact location of the xml example folder when you run the job. Enter the complete location of the XML input file as, #xml_example_root_folder#/xml_examples/orderjoin_step/example1/input_files/employee.xml.

3. On the Document Root tab, select the schema element that describes the documents that the step parses.
   a. Click Browse. Open the Schemas_for_XML_example1 library, which is the library into which you have imported the Organization1.xsd and Employee1.xsd schemas.
   b. Click to open the namespace, http://ibm.com/infosphere/xml/ Employee1, and click the root element Employee_Names.
   c. Click OK. The Document Root tab displays the structure of the schema. The step Output also displays this same structure.

4. On the Validation tab, select Strict Validation. By selecting Strict Validation, you automatically ensure that the data types conversion is performed. If you use the default Minimal Validation, all data types are treated as String type. The following figure shows the output schema of the first Parser step:
Step 9.4: Configuring the second XML Parser step

Configure the location and the schema of the second XML source data that you want to parse.

Procedure

1. In the Assembly Outline, click the XML_Parser_1 Step. By default, the Configuration tab for the step opens. The following figure shows the Configuration tab for the step:

2. On the XML Source tab, specify the location of the XML source data. In this example, the XML source data is in the single file named, address.xml. Select Single file and then click Insert Parameter, and select the xml_example_root_folder parameter. You need to specify the exact location of
the xml example folder when you run the job. Enter the complete location of
the XML input file as, #xml_example_root_folder#/xml_examples/
orderjoin_step/example1/input_files/address.xml.

3. On the Document Root tab, select the schema element that describes the
documents that the step parses.

   a. Click Browse. Open the Schemas_for_XML_example1 library, which is the
      library into which you have imported the Organization1.xsd and
      Employee1.xsd schemas.
   
   b. Click to open the namespace, http://ibm.com/infosphere/xml/ Employee1,
      and click the root element Employee_Addresses.
   
   c. Click OK. The Document Root tab displays the structure of the schema.
      The step Output also displays this same structure.

4. On the Validation tab, select Strict Validation. By selecting Strict Validation,
you automatically ensure that the data types conversion is performed. If you
use the default Minimal Validation, all data types are treated as String type.

The following figure shows the output schema of the second Parser step:

---

**Step 9.5: Configuring the OrderJoin step**

Use the OrderJoin step to merge the items in the employee and address lists into a
single list based on their position.

**About this task**

The Input to the OrderJoin step contains two lists, employee and address. When
you configure the OrderJoin step, make a selection for the left and right list.

**Procedure**

On the Configuration tab of the OrderJoin step, configure these fields:

**Table 70. Configuring the fields**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
</table>
| Left List  | Select top/XML_Parser:result/
             | prn:Empoyee_Names/employee. |

---

162 XML Transformation Guide
Table 70. Configuring the fields (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right List</td>
<td>Select top/XML_Parser_1:result/ prn:Employee_Addresses/address.</td>
</tr>
</tbody>
</table>

Results

The step Output contains a new node named OrderJoin:result, which contains the merged list of employee and address lists. The following figure shows this portion of the step Output.

Step 9.6: Configuring the XML Composer step

Configure the XML Composer step to compose the OrderJoin step output into a XML file.

Procedure

1. In the XML Target window of the XML Composer step, choose Pass as String.
   This passes the composed XML string to a downstream step or stage for further processing.
2. In the **Document Root** window, select the schema element that describes the documents that the step composes:
   a. Click **Browse**, and then open the Schemas_for_XML_examples library, which is the library into which you imported the Organization1.xsd and Employee1.xsd schemas.
   b. Click to open the namespace, `http://ibm.com/infosphere/xml/Organization1`, and select the root element Employees.
   c. Click **OK**. The **Document Root** tab displays the structure of the schema.

3. On the **Validation** tab, by default **Strict validation** is selected. You do not need to make any changes to the fields in this tab.

4. In the OrderJoin step, you merged two lists into a single list based on their position. Now on the **Mappings** tab, map the items from the target schema to the items in the output schema coming from the previous steps. To create the mappings, perform the following steps:
   a. Map `top/OrderJoin:result` in the Source list to the Employee in the Target list, and then click **Automap**.
   b. The **Automap** automatically maps each item under `top/OrderJoin:result` list with the items under the target Employee list.
   c. Review all the items to see if the correct mapping is done.

5. On the **Format** tab, default encoding is UTF-8. You need not change this field.

6. Select the Format Style and configure the following fields.
   a. In the **Indentation length** field, select the number of characters to use for each indentation level in the XML output. By default, number of characters is set to 4.
   b. In the **New Line Style** field, select the type of new line. Choices are UNIX (LF), DOS(CRLF) or MAC(CR). By default, the choice selected is DOS.

**Results**

The following figure shows the results of the mappings:

![XML_Composer:result](image)

The step Output contains a new node, named XML_Composer:result. This node carries string representation of the result of the XML composer.

**Step 9.7: Configuring the Output step**

In the Output step, you create the mappings that define how to map target items in one data structure to source items for downstream stage.

**About this task**

In this example, you use the Output step to map the output of the composer to a sequential file.

**Procedure**

1. In the Assembly Outline, click the Output Step to open the step. The step displays the **Configuration** tab in the **Output Step** window.
The default view in the Output window is the Links view. The output structure for Employees_OrderJoin link is shown. The previous figure shows that the output table does not display any columns because when you created the job, you did not define any column for the Employees_OrderJoin link. The lack of columns is not an error. However, if no column is defined on the link, you cannot map any source items in the hierarchical data structure to this link.

The composer output, result-string is a single column that contains the employee data and you need to create that single column in this step. But instead of returning to the job and manually creating the column, you can automatically create it from the Mappings tab of the Output step.

2. Click the Mappings tab. The following figure shows the mapping table. In this table, each item in the output structure is represented by a row in the table. You map target items to source items. For this job, the target structure has a single link, Employees_OrderJoin, and no column is defined for this link. In the Target column, the Employees_OrderJoin link is represented as a list item.

3. Map the Employees_OrderJoin list item. Select the Employees_OrderJoin item. For this item, in the Source, from the Suggestion list, select top and click Propagate. One column for each item that is a descendent of the top item is created in the source structure. When you use the Propagate button to create items, the items are automatically mapped. The following figure shows the result:
4. Click OK to close the assembly editor.
5. Click OK to close the Stage Editor.
6. Save the job.

Step 9.8: Viewing the output of the job
After you run the OrderJoin job, open the XML file, and look at the output.

Procedure
1. From the IBM InfoSphere DataStage and QualityStage Designer client, choose File > Compile to compile the job.
2. Choose File > Run to run the job.
3. In the parameter window, for the first parameter, Root of example folder tree, enter the path of the directory where you have extracted the files from your examples compressed file, examples.zip. For example, if you have downloaded and saved the examples.zip file in the directory, C:\Examples, enter this directory as the Value for the first parameter, Root of example folder tree.

The following are the two input files that contain the employee and address information:

employee.xml file:

```xml
<tns:Employee_Names xmlns:tns="http://ibm.com/infosphere/xml/Employee1">
  <tns:employee EmpId="A8990">
    <name>
      <firstName>Zen</firstName>
      <middleName>P</middleName>
      <lastName>Wright</lastName>
    </name>
    <gender>male</gender>
    <dateOfBirth>1980-04-04</dateOfBirth>
    <title>Mr</title>
  </tns:employee>
  <tns:employee EmpId="B6540">
    <name>
      <firstName>Cynthia</firstName>
      <middleName>P</middleName>
      <lastName>Donald</lastName>
    </name>
    <gender>female</gender>
    <dateOfBirth>1987-01-17</dateOfBirth>
  </tns:employee>
</tns:Employee_Names>
```
<tns:employee EmpId="C1230">
  <name>
    <firstName>Tania</firstName>
    <middleName>G</middleName>
    <lastName>William</lastName>
  </name>
  <gender>female</gender>
  <dateOfBirth>1980-01-17</dateOfBirth>
  <title>Miss</title>
</tns:employee>
<tns:employee EmpId="A8991">
  <name>
    <firstName>John</firstName>
    <middleName>K</middleName>
    <lastName>Milsted</lastName>
  </name>
  <gender>male</gender>
  <dateOfBirth>1982-08-07</dateOfBirth>
  <title>Mr</title>
</tns:employee>
<tns:employee EmpId="B6541">
  <name>
    <firstName>Emily</firstName>
    <middleName>P</middleName>
    <lastName>Cheeseman</lastName>
  </name>
  <gender>female</gender>
  <dateOfBirth>1987-07-07</dateOfBirth>
  <title>Miss</title>
</tns:employee>
</tns:Employee_Names>

address.xml file:

<tns:Empolyee_Addresses xmlns:tn=
  "http://ibm.com/infosphere/xml/Employee1">
  <tns:address EmpId="A8990">
    <street>2301 East Lamar Blvd</street>
    <city>Arlington</city>
    <state>Texas</state>
    <country>USA</country>
    <postalCode>78363</postalCode>
  </tns:address>
  <tns:address EmpId="B6540">
    <street>San Felipe Suite 2400</street>
    <city>Houston</city>
    <state>Texas</state>
    <country>USA</country>
    <postalCode>77057</postalCode>
  </tns:address>
  <tns:address EmpId="C1230">
    <street>54th South Street</street>
    <city>Miami</city>
    <state>Florida</state>
    <country>USA</country>
    <postalCode>32501</postalCode>
  </tns:address>
  <tns:address EmpId="A8991">
    <street>609 Eighth Street</street>
    <city>Portland</city>
    <state>Texas</state>
    <country>USA</country>
    <postalCode>78374</postalCode>
  </tns:address>
  <tns:address EmpId="B6541">
    <street>750 Agronomy Road</street>
  </tns:address>
</tns:Empolyee_Addresses>
4. After the job runs, open the `employees_output.xml` file to look at the results. The following is a portion of the `employees_output.xml` file.

```xml
<Employee>
    <name>
      <firstName>Zen</firstName>
      <middleName>P</middleName>
      <lastName>Wright</lastName>
    </name>
    <gender>male</gender>
    <dateOfBirth>1980-04-04</dateOfBirth>
    <title>Mr</title>
  </prn:employee>
  <prn:address xmlns:prn="http://ibm.com/infosphere/xml/Employee1" EmpId="A8990">
    <street>2301 East Lamar Blvd</street>
    <city>Arlington</city>
    <state>Texas</state>
    <country>USA</country>
    <postalCode>78363</postalCode>
  </prn:address>
  <Employee>
    <prn:employee xmlns:prn="http://ibm.com/infosphere/xml/Employee1" EmpId="B6540">
      <name>
        <firstName>Cynthia</firstName>
        <middleName>P</middleName>
        <lastName>Donald</lastName>
      </name>
      <gender>female</gender>
      <dateOfBirth>1987-01-17</dateOfBirth>
      <title>Miss</title>
    </prn:employee>
    <prn:address xmlns:prn="http://ibm.com/infosphere/xml/Employee1" EmpId="B6540">
      <street>San Felipe Suite 2400</street>
      <city>Houston</city>
      <state>Texas</state>
      <country>USA</country>
      <postalCode>77057</postalCode>
    </prn:address>
  </Employee>
</Employee>
```

As you see in the output file, the Order join step has merged the employee and address lists.

**Example 10: Using the XML Parser and V-Pivot steps**

In this example, you create a job that uses the XML stage and a Sequential file stage. The V-Pivot step is used to transform the values in a column into a single row.

**About this task**

This example uses the sample XML data file, `departments.xml`, and the sample schemas, `Employee.xsd` and `Organization.xsd`, to illustrate parsing source data.
from one XML file into a sequential file. The departments.xml file contains information about the employees of various departments in a company. In this example, you parse the XML data into a sequential file. The file contains employee business and address information. The employee business information contains the details such as employee name, date of hire, date of birth, gender, title, employee ID, and department ID. The employee address information contains employee ID, street, city, state, postal code, country, and address type.

After you complete all the steps in the assembly, the Assembly Editor will look as shown in the following figure.

The completed example job, xml_vpivot_example.dsx, is also available. To look at the completed example or use it as the starting point for expanding the example, import it into IBM InfoSphere DataStage.

To create the example, complete these tasks:

**Step 10.1: Creating the job**
Create the job that includes an XML stage and a Sequential File stage.

**About this task**

The following figure shows the job that you create for the V-Pivot example. The job includes an XML stage named Department and a Sequential File stage named Employee_File. The XML stage is linked to a Sequential file stage by a link named Employees_Vpivot.
Procedure

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and select New > Parallel job.
3. Open the Real Time section of the palette, and drag XML stage to the canvas.
4. Specify job properties for the location of the example files:
   b. On the Parameters tab, in the Parameter Name field, enter xml_example_root_folder and then specify the following values:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Points to the location of the example folder</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Points to the location of the example folder</td>
</tr>
</tbody>
</table>

5. Open the File section of the palette, and drag a Sequential File stage to the canvas. Position this stage to the right of the XML stage.
6. Create a link from the XML stage to the Sequential file stage.
7. Rename the stages and link as shown in the following table:

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML stage</td>
<td>Department</td>
</tr>
<tr>
<td>Link from XML stage to Sequential File stage</td>
<td>Employees_Vpivot</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Employee_File</td>
</tr>
</tbody>
</table>

8. Double-click the Employee_File stage to configure the stage properties.
9. Click the Properties tab, and configure the following properties that define the output file:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Enter the path where you want the output file to be created, followed by the file name employees_output.txt.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose Overwrite to create the file.</td>
</tr>
</tbody>
</table>
**Table 73. Configuring the properties (continued)**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Line is Column Names</td>
<td>Choose True</td>
</tr>
</tbody>
</table>

For the Employee_File stage, do not create any columns now. You let the Assembly Editor to propagate the columns automatically.

10. Click **OK** to close the Employee_File stage.

11. Choose **File > Save**, and name the job as `xml_vpivot_example`.

**Step 10.2: Creating the assembly**

Use the XML Parser stage to parse the XML files, the V-Pivot transforms the field values of columns into a single row, and the Output step to map the data to the sequential file.

**Before you begin**

Import the example schemas `employee.xsd` and `organization.xsd`. If you have already built any of the previous examples, you have imported the schemas as part of that example. For more information about importing the schemas, see [Importing the schemas for the examples](#).

**Procedure**

1. Double-click the XML stage to open the stage editor.
2. Click the **Edit assembly** button to open the Assembly Editor.
3. Open the Palette, and then double-click the XML_Parser Step to add it to the Assembly Outline.
4. Select the XML_Parser Step, and then from the palette, double-click the V-Pivot Step to add it below the XML_Parser Step in the Assembly Outline.
5. On the Overview, in the **Description** field, enter the following description: This assembly uses the `Organization.xsd` and `Employee.xsd` schemas to read the input XML file. In this example, the V-Pivot Step transforms the field values of columns into a single row.

**Step 10.3: Configuring the XML Parser step**

Configure the location of the XML source data and the schema that you want to use.

**Procedure**

1. In the Assembly Outline, click the XML parser step. By default, the **Configuration** tab for the step opens. The following figure shows the **Configuration** tab for the step:
2. On the **XML Source** tab, specify the location of the XML source data. In this example, the XML source data is in the single file `departments.xml`. Select **Single file**, and then click **Insert Parameter** and select the `xml_example_root_folder` parameter. You will specify the exact location of the file when you run the job. You can also specify the absolute path to `departments.xml`.

3. On the **Document Root** tab, select the schema element that describes the documents that the step parses.
   a. Click **Browse**, and then open the Schemas_for_XML_examples library, which is the library into which you imported the `Organization.xsd` and `Employee.xsd` schemas.
   b. Click to open the namespace, `http://ibm.com/infosphere/xml/`. Organization, click the root element `department`, and then click **OK**. The **Document Root** tab displays the structure of the schema. The step Output also displays this same structure.

4. On the **Validation** tab, select **Strict Validation**. By selecting **Strict Validation**, you automatically ensure that the data types conversion is performed. If you use the default **Minimal Validation**, all data types are treated as String type. The following figure shows the output schema of the Parser step:
Step 10.4: Configuring the V-Pivot step

Use the V-Pivot step to transform the field values of columns into a single row.
About this task

In the V-Pivot step you transform the fields that are represented in a list of columns into a single row. In this step you specify the `address_type` values as column names based on which the V-Pivot function is performed.

In V-Pivot, the column values can contain alphabets, digits and the special characters, . (dot), _ (underscore), and - (hyphen). The column values can only start with either alphabets or underscores. The column values cannot contain special characters such as, : (colon), @, etc. This is because the given column values are treated as column names, and the column names cannot contain special characters.

Procedure

1. On the **Configuration** tab of the V-Pivot step, configure these fields:

   **Table 74. Configuring the fields**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source of Rows</td>
<td>Choose Address.</td>
</tr>
<tr>
<td>Scope</td>
<td>Choose employee.</td>
</tr>
</tbody>
</table>

   The Scope in the V-Pivot step defines the scope of the V-Pivot function. The fields in the columns within this scope are transformed into a single row. The Scope also determines the location of the output of the V-Pivot step.

   The Scope should always be the parent node of the list selected in the **Source of Rows** field. In this example, because the **Source of Rows** is selected as Address, you can only select the scope as employee.

   You cannot select **top** as the scope. By default, the scope is set to **top**. You have to specify proper scope based on your specific V-Pivot use case.

2. Select the **Source of Column Names**.

3. Choose value of **Source of Column Names** as `address_type`. Specify the address type as H and O under the **Column Name** field where H and O are the address type values in the input file. The column names should match the values that are present in the input file.

4. The value that you selected under **Column Name** is the result output that appears in a single row rather than in different columns.
Results

The following figure shows the output of the V-Pivot on the Output tab.

The output of the V-Pivot step contains a new list named V-Pivot:result, which contains two nodes; H and O. The node, 'H' holds the house address of the employees in a department. The node, 'O' holds the office address of the employees in a department.

**Step 10.5: Configuring the Output step**

In the Output step, you create the mappings that define how to map target items in one data structure to source items in another data structure.

**About this task**

In this example, you use the Output step to map the output of the composer to a sequential file.
Procedure

1. In the Assembly Outline, click the Output Step to open the step. The step displays the Configuration tab in the Output Step window.

```
<table>
<thead>
<tr>
<th>Output Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
</tr>
</tbody>
</table>

Switch to Tree View

Output Links: Employee_Vpivot

Enable Runtime Column Propagation

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Type</th>
<th>Extended</th>
<th>Length</th>
<th>Scale</th>
<th>Nullable</th>
<th>Display Size</th>
<th>Description/Path</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Click to add</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The default view in the Output window is the Links view. The output structure for Employee_Vpivot link is shown. The previous figure shows that the output table does not display any column because when you created the job, you did not define any column for the Employee_Vpivot link. The lack of column is not an error. However, if no column is defined on the link, you cannot map any source items in the hierarchical data structure to this link. Instead of returning to the job and manually creating the column, you can automatically create it from the Mappings tab of the Output step.

2. Click the Mappings tab. The following figure shows the mapping table. In this table, each item in the output structure is represented by a row in the table. You map target items to source items. For this job, the target structure has a single link, Employee_Vpivot, and no column is defined for this link. In the Target column, the Employee_Vpivot link is represented as a list item.

```
<table>
<thead>
<tr>
<th>Source</th>
<th>Result</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td></td>
<td>top</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Click to select] (mapping)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Employees_Vpivot</td>
</tr>
</tbody>
</table>
```

3. Map the Employee_Vpivot list item. Select the Employee_Vpivot item. For this item, in the Source, from the Suggestion list, select ‘employee’ and click Propagate. One column for each item that is a descendent of the ‘employee’ item is created in the source structure. When you use the Propagate button to create items, the items are automatically mapped. The following figure shows the result:
4. Click OK to close the assembly editor.
5. Click OK to close the Stage Editor.
6. Save the job.

**Step 10.6: Viewing the output of the job**

After you run the V-Pivot job, open the text file, and look at the output.

**Procedure**

1. From the IBM InfoSphere DataStage and QualityStage Designer client, choose File > Compile to compile the job.
2. Choose File > Run to run the job.
3. In the parameter window, for the first parameter, Root of example folder tree, enter the path of the directory where you have extracted the files from your examples compressed file, examples.zip. For example, if you have downloaded and saved the examples.zip file in directory, C:\Examples, enter this directory as the Value for the first parameter, Root of example folder tree.

The following is an input file that contains the employee information in a department:

```xml
<department xmlns="" departmentID="A100" departmentKind="Division">
  <manager>A7100</manager>
  <ns1:employees>
    <ns1:employee employeeID="A8990" departmentID="A100">
      <name>
        <firstName>Zen</firstName>
        <middleName>P</middleName>
        <lastName>Wright</lastName>
      </name>
    </ns1:employee>
  </ns1:employees>
</department>
```

XML transformation 177
4. After the job runs, open the employees_output.txt file to look at the results. The employees_output.txt file contains this data:

firstName", "middleName", "lastName", "gender", "dateOfBirth", "title", "hireDate", "employeeID", "departmentID", "street", "city", "state", "country", "postalCode", "address_type", "street_2", "city_2", "state_2", "country_2",
Examples of transforming JSON data

You can build sample jobs that parse and compose JSON data.

The examples all use one schema file schema.json. You can create each example or you can import the completed job. The data for the examples is a JSON file that contains contact information. The schema defines the contact information structure.


Example 1: Parsing JSON data

You can create a simple job that uses the JSON Parser step to parse contact data, which is stored in one JSON data file, into two flat files.

About this task

This basic parsing example uses the sample JSON data file, schema.json. You parse contact information from the schema.json file into two files. One file contains contact information: first name, last name, age, street address, city, postal code, new subscription, and the company name. The second file contains phone numbers: number and type.

Related reference:

"JSON Parser step" on page 40

Use the JSON Parser step to parse one or more documents that have the same structure.

Creating the job

Create the example job that includes one XML stage and two Sequential File stages.

About this task

The following figure shows the job that you create for the parsing example. The job includes one XML stage, named JSON_Parser, and two Sequential File stages, named Contacts_File and PhoneNumbers_File. The JSON_Parser stage is linked to the Contacts_File stage by a link named Contacts, and it is also linked to the PhoneNumbers_File stage by a link named PhoneNumbers.
Procedure

1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and then click New > Parallel job.
3. Open the Real Time section of the palette, and drag one XML stage to the canvas.
4. Create a job parameter for the location of the example files:
   a. Click Edit > Job Properties.
   b. On the Parameters page, specify the values in the following table.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Name</td>
<td>json_example_root_folder</td>
</tr>
<tr>
<td>Prompt</td>
<td>Root of example folder tree</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Point to the root of the file tree folder</td>
</tr>
</tbody>
</table>

Each example uses this job parameter.
5. Open the File section of the palette, and drag two Sequential File stages to the canvas. Position these stages to the right of the XML stage.
6. Create a link from the XML stage to each Sequential File stage.
7. Rename the stages and links as shown in the following table.

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML stage</td>
<td>JSON_Parser</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>Contacts_File</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>PhoneNumbers_File</td>
</tr>
<tr>
<td>Link from JsonParser to Contacts_File</td>
<td>Contacts</td>
</tr>
<tr>
<td>Link from JsonParser to PhoneNumbers_File</td>
<td>PhoneNumbers</td>
</tr>
</tbody>
</table>

XML transformation 181
8. Double-click the Contacts_File stage, and configure the columns that are listed in the following table in the Columns tab. By looking at the schema.json file, which contains the JSON data, you determine which columns to create to hold the contacts data that the XML stage will pass to the Contacts_File stage and the PhoneNumbers_File stage.

Table 77. Columns for the Contacts_File stage

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>LastName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>Age</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>StreetAddress</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>PostalCode</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>NewSubscription</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>CompanyName</td>
<td>VarChar</td>
<td></td>
</tr>
</tbody>
</table>

The columns can also be auto-generated by using the Propagate button in the Mappings tab in the Output step.

a. On the Properties page, configure the properties that define the output file:

Table 78. Properties for the output file

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Enter the path where you want the output file to be created, followed by the file name Contacts.txt.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose Overwrite to create the file.</td>
</tr>
<tr>
<td>First line is column name</td>
<td>Set to True.</td>
</tr>
</tbody>
</table>

b. Click OK to close the Contacts_File stage.

9. Double-click the PhoneNumbers_File stage, and then, on the Properties page, configure the properties that are listed in the following table: For the PhoneNumbers_File stage, do not create any columns. Later, you will use the Assembly Editor to propagate those columns automatically.

Table 79. Configuring the properties

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target &gt; File</td>
<td>Enter the path where you want the output file to be created, followed by the file name PhoneNumbers.txt.</td>
</tr>
<tr>
<td>Target &gt; File Update Mode</td>
<td>Choose Overwrite to create the file.</td>
</tr>
<tr>
<td>First line is column name</td>
<td>Set to True.</td>
</tr>
</tbody>
</table>

a. Click OK to close the stage properties.

10. Choose File > Save, and name the job json_parser_example.

Opening the Assembly Editor and viewing the issues
For each XML stage, you open the Assembly Editor, which you use to define an XML transformation within the context of a job.
About this task

You use the Assembly Editor to create an assembly. An assembly contains a series of steps that parse, compose, and transform hierarchical data. By default, an assembly contains an overview, an input step, and an output step. You add steps to the assembly based on the type of transformations that you want to perform. In this example, you add one JSON_Parser step to parse JSON data into two output Sequential File stages, Contacts_File, and Employee_File.

Procedure

1. Double-click the JSON_Parser stage and then, click **Edit assembly**.
2. Click **Palette** to open the palette, and then double-click the JSON Parser step to add it to the outline. Because the JSON Parser step is the first step that you added to the assembly, the step is added between the Input step and the Output step.
3. In the outline, an error icon (a red exclamation mark) is shown beside the JSON Parser step and the Output step. The error icon indicates that you must address one or more issues in that step. To view the issues list, click **View All**. The following figure shows the issues for this assembly:

```
- JSON_Parser - 2 Issues
  - CDIUI2805E Input is required for: assembly.step.error.jsoninput.connectoroutputschema.
  - CDIUI2805E Input is required for: assembly.step.error.jsoninput.connectorproperties.

- Output - 9 Issues
  - CDIUI2805E Input is required for: top/outputLinks/Contacts.
  - CDIUI2805E Input is required for: top/outputLinks/Contacts/FirstName.
  - CDIUI2805E Input is required for: top/outputLinks/Contacts/LastName.
  - CDIUI2805E Input is required for: top/outputLinks/Contacts/Age.
  - CDIUI2805E Input is required for: top/outputLinks/Contacts/StreetAddress.
  - CDIUI2805E Input is required for: top/outputLinks/Contacts/City.
  - CDIUI2805E Input is required for: top/outputLinks/Contacts/State.
  - CDIUI2805E Input is required for: top/outputLinks/Contacts/PostalCode.
  - CDIUI2805E Input is required for: top/outputLinks/PhoneNumbers.

Navigate to item
```

The JSON Parser step has two errors, indicating that you must complete two mandatory fields. The Output step has nine errors. Notice that the errors that require mandatory input are for the Contacts and PhoneNumbers output links and the columns that you defined on the Contacts link. These links and columns are listed as errors because the hierarchical structure of the assembly is not mapped to a relational structure. The first error in the Output list indicates
that no columns were defined on the PhoneNumbers link. For now, you can ignore these errors; you correct them when you create mappings in the Output step.

4. Close the issues list.

**Importing the schemas for the examples**

Use the schema library manager to generate a schema to describe a selected JSON data and import the schema into a library.

**About this task**

You must generate and import the schemas that the example job uses. Imported schemas are stored in the metadata repository, where you can access them to use in any assembly that you create.

**Procedure**

1. From the Assembly Editor, click the Libraries tab to open the Schema Library Manager.
2. Click New Library to create a library for the example schemas.
   a. Enter Example_PARSER for the library name.
   b. Enter Examples for the category
   The library is now categorized under Examples.
3. Expand the Examples category, select Example_PARSER, and then click Import New Resource. Set the Files of type to JSON Instance to show the available json files. Find the schema.json, and import it into the library.
   a. Import schema.json file. This will generate a schema to describe the structure and types in the selected schema.json file. The schema is imported into the specified library.
4. To return to the assembly, click the Assembly Editor tab.

**Related tasks:**

“Opening and configuring the Assembly Editor” on page 194

In the Assembly Editor, you add the HJoin step, which is used to create a parent-child hierarchy from two lists. Then, you add the JSON Composer step, which can compose the JSON file.

**Configuring the overview**

The Overview provides creation and modification information about the assembly and includes a Description field that you can modify.

**About this task**

On the Overview, you can enter an optional description for the assembly. This field documents the purpose of the assembly and is helpful when you or others later need to modify the assembly.

**Procedure**

In the Description field, enter the following description: This assembly uses the schema.json schema. The assembly parses the schema.json data file and sends the output to two files named Contacts.txt and PhoneNumbers.txt.
**Configuring the JSON Parser step**

Configure the location of the JSON source data and the schema that you want to use to parse it.

**Procedure**

1. Click the JSON Parser step in the outline.
2. On the JSON Source page specify the location of the JSON source data. In this example, the JSON source data is in the single file `schema.json`.
   a. Select **Single file**.
   b. Click **Insert Parameter** and select the `json_example_root_folder` parameter. Specify the exact location of the file when you run the job. For example if you have downloaded and saved the json schema file in the directory, `C:\json_example_root_folder`, enter this directory as the value for the first parameter: `c:/json_example_root_folder/schema.json`.
3. On the Document Root page, you select the schema element that describes the documents that the step parses. Click **Browse**. Open the Example_Parser library, which is the library into which you imported the `schema.json` file.
4. Select the root element and then click **OK**. The Document Root page displays the structure of the schema. The Output schema page also displays the same structure.
5. On the Validation page, select **Minimal Validation**. If you use Minimal Validation all data types are automatically converted to the String type.

**Configuring the Output step**

In the Output step, create mappings that define how to map source items in one data structure to target items in another data structure.

**About this task**

In this example, you use the Output step to map a hierarchical data structure to a relational data structure.

**Procedure**

1. From the Assembly Editor, click the Output step. On the Output page, the output table shows the data structure that is the output of the assembly.
The output structure for the Contacts link is shown. Notice that the columns that you defined in the Sequential file stage editor are shown in the output table. In the Links view, you can modify the columns that you defined. Any changes that you make to the columns are propagated to the column definitions in the Sequential file stage properties.

2. From the Output Links list, select PhoneNumbers. The output table does not show any columns because you did not define any columns for the PhoneNumbers link when you created the job. If no columns are defined on the link, you cannot map any source items in the hierarchical data structure to this link. Because this job is designed to produce a file that contains phone numbers data, you need to create the phone numbers columns. But instead of returning to the job and manually creating columns, you can automatically create them from the Mappings tab of the Output step.

3. Click the Mappings tab. Each item in the output structure is represented by a row in the mappings table. You map target items to source items. For this job, the target structure has two links, Contacts and PhoneNumbers, and the columns that are defined on those links. In the Target column, the Contacts link is represented as a list item. Under the Contacts list item, each column displays as a content item.
   a. To create mappings, map target list items to source list items. Then map target content items to source content items.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Map</td>
<td>Click Auto Map to map target content items to source content items automatically.</td>
</tr>
</tbody>
</table>
Table 80. Five ways to specify mapping (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>More</td>
<td>Select a mapping candidate from a list of valid mapping candidates or select a mapping candidate from the entire source structure using the <strong>More</strong> option in the list.</td>
</tr>
<tr>
<td>Constant</td>
<td>Specify a constant value as the mapping using the <strong>Constant</strong> option in the list.</td>
</tr>
<tr>
<td>Map to selected target</td>
<td>Right-click on a potential mapping source node in the Input schema tree control, and select <strong>Map to selected target</strong>.</td>
</tr>
<tr>
<td>Add as new column for link or Add children as new column for link</td>
<td>Select <strong>Add as new column for link</strong> menu item to propagate the node as new column. If you right-click on a vector or group node, the <strong>Add children as new column for link</strong> menu item propagates the direct children as new columns for the link.</td>
</tr>
</tbody>
</table>

4. Select the Contacts list item from the Target section.
   a. In the Source section, click and choose the item from the list.
   b. To create a mapping for the selected item and all of its descendent items, click **Auto Map**

Each source item that is automatically mapped is determined based on similar name and data type.

In the following figure the target Contacts list item and all of its child items are automatically mapped to source items.
5. Map the PhoneNumbers list item from the Target column.
   a. In the Source column, select PhoneNumbers from the list, and then, click **Propagate**. Propagate automatically creates one column for each item that is a descendent of the PhoneNumbers item in the source structure and automatically maps those items to the respective items in the source column.

The following figure shows the mapping result after propagating.

6. Click **OK** to close the stage editor.

**Viewing the output of the job**

After you run the parsing job, open the text files, and look at the output.
Procedure

1. From the Assembly Editor, click **OK** to return to the XML stage editor, and then click **OK** to close the XML stage editor.

2. To compile the job, choose **File > Compile** from the IBM InfoSphere DataStage and QualityStage Designer client.

3. Click **File > Run** to run the job.

4. In the parameter window, for the first parameter, **Root of example folder tree**, enter the path of the directory where you extracted your examples .zip file. For example, if you downloaded and saved the examples .zip file in the C:\Examples directory, enter this directory.

   The following is an example input file that contains the contacts and phone number information:

   ```json
   {  
     "contacts": [  
       {  
         "firstName": "John",  
         "lastName": "Smith",  
         "age": 25,  
         "address": {  
           "streetAddress": "21 2nd Street",  
           "city": "New York",  
           "state": "NY",  
           "postalCode": "10021"  
         },  
         "phoneNumbers": [  
           { "type": "home", "number": "212 555-1234" }  
         ],  
         "newSubscription": false,  
         "companyName": "FakeOne"  
       },  
       {  
         "firstName": "John",  
         "lastName": "Smith",  
         "age": 25,  
         "address": {  
           "streetAddress": "21 2nd Street",  
           "city": "New York",  
           "state": "NY",  
           "postalCode": "10021"  
         },  
         "phoneNumbers": [  
           { "type": "home", "number": "212 555-1234" },  
           { "type": "fax", "number": "646 555-4567" }  
         ],  
         "newSubscription": false,  
         "companyName": "FakeOne"  
       }  
     ]  
   }
   ```

5. After the job runs, open the **Contacts.txt** and the **PhoneNumbers.txt** files to look at the results.

   The **Contacts.txt** file contains this data:

<table>
<thead>
<tr>
<th>FirstName</th>
<th>LastName</th>
<th>Age</th>
<th>StreetAddress</th>
<th>City</th>
<th>State</th>
<th>PostalCode</th>
<th>newSubscription</th>
<th>CompanyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>25</td>
<td>21 2nd Street</td>
<td>New York</td>
<td>NY</td>
<td>10021</td>
<td>false</td>
<td>FakeOne</td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
<td>25</td>
<td>21 2nd Street</td>
<td>New York</td>
<td>NY</td>
<td>10021</td>
<td>false</td>
<td>FakeOne</td>
</tr>
</tbody>
</table>

   The **PhoneNumbers.txt** file contains this data:
Example 2: Composing JSON data by using the JSON Composer and HJoin steps

Build this job that uses the JSON Composer and HJoin steps in the XML stage to create a hierarchical structure.

About this task

This basic example uses two relational data files, Contacts.txt and PhoneBooks.txt, and the schema.json sample JSON data file to compose source data from two relational files into a JSON file. The Contacts.txt file contains these contact details: first name, last name, age, street address, city, state, postal code, new subscription, and company name. The PhoneBooks.txt file contains first name, last name, number, and type. Each person has an address and a phone number. In this example, you use an HJoin step in the assembly to join two tables to create hierarchical structure. Then you use the Address_JSON_Composer Step and PhoneNumbers_JSON_Composer Step for each view to build one part of your JSON document and JSON_Composer Step to assemble all the generated parts and to compose a JSON file that contains contact information followed by the phone numbers.

This example illustrates how to use schema views to compose JSON sample data. A schema view is used to define a subset of a large schema. When you design an assembly to compose JSON data based on a small schema, such as the one in this example, you do not need to split the composing process into multiple parts. You can compose the sample JSON data by using one composer step. For more information about large schema, see "Large Schema" on page 46.

Related reference:

"JSON Composer step" on page 43

Use a JSON schema or a view created from the JSON schema to compose JSON data.

Creating the job

Create the example job that includes one XML stage and two Sequential File stages.

About this task

The following figure shows the job that you create for the composing example. The job includes two Sequential File stages that are named Contacts_file and PhoneNumbers_file. These Sequential File stages are linked to an XML stage by two links that are named Contacts and PhoneNumbers. In this example, instead of using a Join stage in the job to join the files into a list and a Regroup step in the assembly to join the files into a nested list, you use the HJoin step in the assembly to perform both actions.
Procedure
1. Start the IBM InfoSphere DataStage and QualityStage Designer client.
2. In the Repository pane, right-click the Jobs folder, and then click New > Parallel job.
3. Open the File section of the palette, and drag two Sequential File stages to the canvas.
4. Open the Real Time section of the palette, and drag one XML stage to the canvas. Position the XML stage to the right of the Sequential File stages.
5. Create a job parameter for the location of the example files:
   a. Click Edit > Job Properties.
   b. On the Parameters page, specify the values in the following table.

   Table 81. Values for the parameter
<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Name</td>
<td>json_example_root_folder</td>
</tr>
<tr>
<td>Prompt</td>
<td>Root of example folder tree</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Help Text</td>
<td>Point to the root of the file tree folder</td>
</tr>
</tbody>
</table>

   c. Click OK to save the values for the parameter.
6. Create a link from each Sequential File stage to the XML stage.
7. Rename the stage and links as shown in the following table.

   Table 82. Names for stages and links
<table>
<thead>
<tr>
<th>Stage or link</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential File stage</td>
<td>Contacts_file</td>
</tr>
<tr>
<td>Sequential File stage</td>
<td>PhoneNumbers_file</td>
</tr>
<tr>
<td>Link from Contacts_file to XML stage</td>
<td>Contacts</td>
</tr>
<tr>
<td>Link from PhoneNumbers_file to XML stage</td>
<td>PhoneNumbers</td>
</tr>
</tbody>
</table>
Table 82. Names for stages and links (continued)

<table>
<thead>
<tr>
<th>Stage or link</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML stage</td>
<td>ComposeJSON</td>
</tr>
</tbody>
</table>

**Configuring the Sequential file stages**

Configure the Sequential file stages to read the required files.

**About this task**

This job uses two sequential file stages. Configure the properties for Contacts_file and PhoneNumbers_file stages.

**Procedure**

1. Configure the Contacts_file stage.
   a. Double-click the Contacts_file stage.
   b. Click the Output tab, and then click the Properties tab.
   c. Specify values for the properties in the following table.

   *Table 83. Values for properties*

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>The path and file name of the file. For the Contacts_file stage, the file name is contacts.txt</td>
</tr>
<tr>
<td>Read Method</td>
<td>Specific File(s)</td>
</tr>
<tr>
<td>First Line is Column Names</td>
<td>True</td>
</tr>
</tbody>
</table>

2. Click the **Columns** tab, and create the columns in the following table.

   *Table 84. Columns for the Contacts_file stage*

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>LastName</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>Age</td>
<td>VarChar</td>
<td>40</td>
</tr>
<tr>
<td>StreetAddress</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>PostalCode</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>NewSubscription</td>
<td>VarChar</td>
<td></td>
</tr>
<tr>
<td>CompanyName</td>
<td>VarChar</td>
<td></td>
</tr>
</tbody>
</table>

3. Configure the PhoneNumbers_file stage. Double-click the PhoneNumbers_file stage to open the stage properties. On the **Properties** tab of the Output page, configure the following properties:
   a. Double-click the PhoneNumbers_file stage.
   b. Click the Output tab, and then click the Properties tab.
   c. Specify values for the properties in the following table.
**Table 85. Values for properties**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>The path and file name of the file. For the PhoneNumbers_file stage, the file name is phonenumbers.txt</td>
</tr>
<tr>
<td>Read Method</td>
<td>Specific File(s)</td>
</tr>
<tr>
<td>First Line is Column Names</td>
<td>True</td>
</tr>
</tbody>
</table>

4. Click the **Columns** tab and create the columns in the following table.

**Table 86. Columns for the PhoneNumbers_file stage**

<table>
<thead>
<tr>
<th>Column name</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstName</td>
<td>VarChar</td>
</tr>
<tr>
<td>LastName</td>
<td>VarChar</td>
</tr>
<tr>
<td>Number</td>
<td>VarChar</td>
</tr>
<tr>
<td>Type</td>
<td>VarChar</td>
</tr>
</tbody>
</table>

5. Click **File > Save**, and then name the job **json_hjoin_example**

**Creating schema views**

When a schema contains a large number of nodes, you create schema views to reduce the size of schema tree and improve the efficiency of the parsing and composing processes. Schema views are saved in the Schema Library Manager so that they can be reused for different steps and different job designs.

**Before you begin**

The `schema.json` is a simple schema hence does not require views to compose JSON data. To show case how views can be used, we have used simple schema to compose JSON data by using views.

**About this task**

Create three views based on the original root schema.

**Procedure**

1. Start the IBM® InfoSphere® DataStage® and QualityStage™ Designer.
2. Open Assembly Editor from the XML stage.
3. On **Libraries** tab, click **New Library** or select an existing schema library.
4. Select the node in the JSON schema that you want to create a schema view for.
5. Click **Create View**.
6. Enter the view name and the description.
7. Use the **Find** option to locate the nodes by a name. You can also search the nodes by specifying the node level. Click **Next** after entering the node name and specifying the node level.
8. Specify the properties for the schema view.
Table 87. Properties for the schema view

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include in View</td>
<td>Select this check box for the optional nodes that you want to include in the view. Mandatory nodes are selected automatically. You need to manually select the nodes that are optional.</td>
</tr>
<tr>
<td>Include All Descendants</td>
<td>Select this check box to include all XML elements that are under the selected node.</td>
</tr>
<tr>
<td>Chunk</td>
<td>Select this check box to chunk schema tree of a selected XML element as one single XML node with the XML type.</td>
</tr>
</tbody>
</table>

9. Click OK to save the view. Create address_view, phonenumbers_view, and root_view. The address_view contains the address information. The phoneNumbers_view contains the phone numbers. The root_view is created to combine the information in the address_view and phonenumbers_view. As shown in the below figure, the address and the phone numbers in the root_view are chunked and represented by a single node.

Opening and configuring the Assembly Editor

In the Assembly Editor, you add the HJoin step, which is used to create a parent-child hierarchy from two lists. Then, you add the JSON Composer step, which can compose the JSON file.

Before you begin

To create a schema to describe the structures and types in the example JSON data file, import the schema.json file. If you built the example job that parsed JSON data, you already have a schema to describe the JSON data that you want to compose.
About this task

You can use the Assembly Editor to create an assembly. An assembly contains a series of steps that parse, compose, and transform hierarchical data. By default, an assembly contains an Overview, an Input step, and an Output step. You can add steps to the assembly based on the type of transformations that you want to perform. In this example you need add multiple steps to compose an JSON document.

Procedure

1. Double-click the XML stage and then click Edit assembly.
2. On the Overview page, in the Description field, enter the following description:
   This assembly uses the OneContact.json schema. The HJoin step creates a hierarchical structure, where the contact information is the parent and the phone number information is the child. The JSON_Composer step composes the data and saves the output in a file named contacts_output.
   This assembly uses the default Input step, which shows the columns that the two input links, Contacts and Phone Numbers, pass to the XML stage. You can display the input columns either as links or a tree.
3. Open the Palette, and then double-click the HJoin step to add it to the assembly.
4. Select the HJoin step, and then double-click the JSON Composer step in the palette to add the JSON Composer step below the HJoin step. Create one HJoin step and three JSON Composer steps.
5. Name the JSON Composer steps as Address_JSON_Composer step, PhoneNumbers_JSON_Composer step, and JSON_Composer step.

Related tasks:
“Importing the schemas for the examples” on page 184
Use the schema library manager to generate a schema to describe a selected JSON data and import the schema into a library.

Configuring the HJoin step

You can configure the HJoin step to create a parent-child list from items that share the same key.

About this task

The input to the HJoin step contains two lists, Contact and PhoneNumbers. When you configure the HJoin step, you identify the parent list and the child list, and you specify the keys to use to join the two lists together. Where the value of the key item is the same in both lists, the HJoin step joins the lists.

Procedure

1. On the Configuration page of the HJoin step, specify the parent and child lists:

   "Table 88. Specify the values for Parent and Child list"

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent List</td>
<td>top/InputLinks/Contacts.</td>
</tr>
<tr>
<td>Child List</td>
<td>top/InputLinks/PhoneNumbers</td>
</tr>
</tbody>
</table>

2. In the Optimization Type field, select In-memory.
3. In the Parent Keys column, select top/InputLinks/Contacts/firstName.
4. In the **Child Keys** column, select top/InputLinks/PhoneNumbers/firstName. The Output page shows the child list PhoneNumbers is added to the Contacts list. The Output page also contains a new node named HJoin:orphans, which contains items that do not match the keys. The following figure shows this portion of the Output page.
Configuring the JSON Composer step

Configure the JSON Composer step to create a JSON file that lists contact addresses, followed by phone numbers.

Procedure

1. In the JSON Target page of the Address_JSON_Composer Step, choose Pass as string.
2. In the Document Root page of the Address_JSON_Composer Step, select the address_view to describe the JSON data produced by the Address_JSON_Composer Step.
   
   a. Click Browse, and then open Schemas_for_JSON_examples library, which is the library where you imported the schema.json sample data and generated a schema that describes the sample data.
   
   b. Select address_view element.
   
   c. Click OK. The Document Root displays the structure of the address_view element.
3. In the JSON Target page of the PhoneNumbers_JSON_Composer Step, choose Pass as string.
4. In the Document Root page of the PhoneNumbers_JSON_Composer Step, select the phonenumbers_view to describe the JSON data produced by the PhoneNumbers_JSON_Composer Step.
   
   a. Click Browse, and then open Schemas_for_JSON_examples library, which is the library where you imported the schema.json sample data and generated a schema that describes the sample data.
   
   b. Select phonenumbers_view element.
   
   c. Click OK. The Document Root displays the structure of the phonenumbers_view element.
5. In the JSON Target page of the JSON_Composer Step, choose Write to File.
   
   a. For the Output Directory, click Insert Parameter, and then select json_example_root_folder.
   
   b. For the Filename Prefix, enter contacts_output.
6. In the Document Root page of the JSON_Composer Step, select the schema element that describes the documents that the step composes:
   
   a. Click Browse, and then open Schemas_for_JSON_examples library, which is the library where you imported the schema.json sample data and generated a schema that describes the sample data.
   
   b. Select the root_view element as the document root to describe the JSON data produced by the JSON_Composer Step.
   
   c. Click OK. The Document Root displays the structure of the root_view element.
7. On the Validation page of the JSON_Composer Step, select Strict validation.
8. On the Mappings page of the JSON_Composer Step, map the items from the Target list to the items in the Source list: The mapping for the document_collection item determines how many JSON files are created. If you map the top element of the input to the document_collection item, one document is created. If you map a list item to the document_collection item, one document is created for each item in the list. The name of each document that is generated has the file name prefix that you specified on the JSON Target page and an index that is an increasing number, for example contacts_output_1.json.
a. Map top/InputLinks/Contacts/first name in the Source list to the first name item in the Target list, and then click Automap. Using automatic mapping, you create the majority of the mappings that you need, but you also create some mappings that you do not want.

b. Select the Job item in the Target list that has incorrect mapping. From the drop down list select More, and then select the correct item for mapping. The result-string generated from the Address_JSON_Composer Step and PhoneNumbers_JSON_Composer Step are mapped to the value for the chunked node "address" and "phoneNumbers". The following figure shows the result of the mapping:

![Mapping Result](image)

The Output page contains a new node named JSON_Composer:result. This node carries the path and file name for the output file during runtime.

**Viewing the output of the job**

After you run the job, look at the output.

**Procedure**

1. In the IBM InfoSphere DataStage and QualityStage Designer client, compile and run the job.

2. In the Job Run Options window, click Parameters page, for the Root of example folder tree parameter, enter the path of the directory where you extracted your example .zip file. For example, if you downloaded and saved the examples .zip file in the Examples folder, enter C:\Examples.

3. Open the contacts_output.json file to view the results, which are shown below. The resulting file contains a list of address, and phone numbers for each person.

```json
{
    "contacts": [
        {
            "firstName": "John",
            "lastName": "Smith",
            "age": "25",
            "address": {
                "streetAddress": "21 2nd Street",
            }
        }
    ]
}
```
Reference

These topics describe supported XML types and XML-type-to-DataStage-type mappings, describe how to get the example files, and provide user responses to error messages and information about working with IBM Software Support.

For more information on XML stage, refer to the Developer article links below:

Supported XML types and type mappings

When you create an assembly, refer to these tables that describe supported XML types, DataStage-to-XML type mappings, and XML-to-DataStage type mappings.

**XML types**

This table describes the types that the XML stage supports and provides examples of each type.

<table>
<thead>
<tr>
<th>XML type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>P0Y0M0DT0H0M0S -P120D, P1Y2MT2S</td>
</tr>
<tr>
<td>dateTime</td>
<td>1970-01-01T00:00:00 1970-01-01T00:00:00002001-04-29T14:29:03.5 2001-04-29T14:29:03+06:00 2001-04-29T14:29:03-06:00 20001-04-29T14:29:00Z</td>
</tr>
<tr>
<td>Time</td>
<td>00:00:00 14:29:03.5 14:29:03+06:00 14:39:03-06:00 14:29:00Z</td>
</tr>
</tbody>
</table>
Table 89. Supported XML types (continued)

<table>
<thead>
<tr>
<th>XML type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>1970-01-01&lt;br&gt;2001-04-29&lt;br&gt;-0099-04-29+06:00&lt;br&gt;2001-04-29-06:00&lt;br&gt;2001-04-29-06Z</td>
</tr>
<tr>
<td>gYear</td>
<td>1970&lt;br&gt;2065-05:00&lt;br&gt;-0099</td>
</tr>
<tr>
<td>gYearMonth</td>
<td>1989-01&lt;br&gt;2065-10-05:00&lt;br&gt;2065-07-05:00&lt;br&gt;2065-09Z</td>
</tr>
<tr>
<td>gMonth</td>
<td>--01&lt;br&gt;--10-05:00&lt;br&gt;--07+05:00&lt;br&gt;--09Z</td>
</tr>
<tr>
<td>gMonthDay</td>
<td>--01-29&lt;br&gt;--10-05:00&lt;br&gt;--07+07:00&lt;br&gt;--09Z</td>
</tr>
<tr>
<td>gDay--</td>
<td>--01</td>
</tr>
<tr>
<td>anyURI</td>
<td><a href="http://sample.gov">http://sample.gov</a>&lt;br&gt;../list.html#item&lt;br&gt;urn:sample.gov</td>
</tr>
<tr>
<td>ENTITY</td>
<td>namesNameWithNoColons</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>namesNameWithNoColons1&lt;br&gt;namesNameWithNoColons2</td>
</tr>
<tr>
<td>ID</td>
<td>Unique_ID: aNameWithNoColons</td>
</tr>
<tr>
<td>IDREF</td>
<td>Reference_to_n_ID: aNameWithNoColons</td>
</tr>
<tr>
<td>IDREFS</td>
<td>namesNameWithNoColons1&lt;br&gt;namesNameWithNoColons2</td>
</tr>
<tr>
<td>QName</td>
<td>item&lt;br&gt;bag:item</td>
</tr>
<tr>
<td>token</td>
<td>A token has only single space character in it.</td>
</tr>
<tr>
<td>language</td>
<td>en-US&lt;br&gt;fr</td>
</tr>
<tr>
<td>Name</td>
<td>foo3&lt;br&gt;foo:bar&lt;br&gt;non-leading,hyphen</td>
</tr>
<tr>
<td>NCName</td>
<td>aNameWithNoColons</td>
</tr>
<tr>
<td>NMTOKEN</td>
<td>comb -no_whitespace</td>
</tr>
<tr>
<td>NMTOKENS</td>
<td>comb_brush -no_whitespace</td>
</tr>
<tr>
<td>NOTATION</td>
<td>item&lt;br&gt;bag:item</td>
</tr>
<tr>
<td>normalizedString</td>
<td>A normalizedString has only space characters in it.</td>
</tr>
<tr>
<td>string</td>
<td>A string, which can include tab characters.</td>
</tr>
</tbody>
</table>
Table 89. Supported XML types (continued)

<table>
<thead>
<tr>
<th>XML type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>32-bit floating-point number. -34E5, 123.65E12, 98.72-2, 12, INF, -INF, NaN</td>
</tr>
<tr>
<td>double</td>
<td>64-bit floating-point number. -34E5, 123.65E12, 98.72-2, 12, INF, -INF, NaN</td>
</tr>
<tr>
<td>decimal</td>
<td>0, 0.0, +1134.56, -32.1</td>
</tr>
<tr>
<td>integer</td>
<td>-4, 0, 76768878788</td>
</tr>
<tr>
<td>long</td>
<td>Range is -9223372036854775808 to 9223372036854775807</td>
</tr>
<tr>
<td>int</td>
<td>Range is -2147483648 to 2147483647</td>
</tr>
<tr>
<td>short</td>
<td>Range is -32768 to 32767.</td>
</tr>
<tr>
<td>byte</td>
<td>Range is -128 to 127.</td>
</tr>
<tr>
<td>positiveInteger</td>
<td>Range is 1, 2, and so on.</td>
</tr>
<tr>
<td>nonPositiveInteger</td>
<td>Range is ..., -2, -1, 0.</td>
</tr>
<tr>
<td>negativeInteger</td>
<td>Range is ...-3, -2, -1.</td>
</tr>
<tr>
<td>nonNegativeInteger</td>
<td>Range is 0, 1, 2, and so on.</td>
</tr>
<tr>
<td>unsignedLong</td>
<td>Range is 0 to 18446744073709551615.</td>
</tr>
<tr>
<td>unsignedInt</td>
<td>Range is 0 to 4294967295.</td>
</tr>
<tr>
<td>unsignedShort</td>
<td>Range is 0 to 65535.</td>
</tr>
<tr>
<td>unsignedByte</td>
<td>Range is 0 to 225.</td>
</tr>
<tr>
<td>hexBinary</td>
<td>None.</td>
</tr>
<tr>
<td>base64Binary</td>
<td>None.</td>
</tr>
<tr>
<td>boolean</td>
<td>Legal values are true and false; 1 and 0.</td>
</tr>
</tbody>
</table>

DataStage-to-XML type mappings

Table 90. DataStage-to-XML type mappings

<table>
<thead>
<tr>
<th>DataStage type</th>
<th>XML type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>long</td>
</tr>
<tr>
<td>Unsigned BigInt</td>
<td>unsignedLong</td>
</tr>
<tr>
<td>Binary</td>
<td>binary</td>
</tr>
<tr>
<td>Bit</td>
<td>boolean</td>
</tr>
<tr>
<td>Char</td>
<td>string</td>
</tr>
<tr>
<td>Date</td>
<td>date</td>
</tr>
<tr>
<td>Decimal</td>
<td>decimal</td>
</tr>
<tr>
<td>Double</td>
<td>double</td>
</tr>
<tr>
<td>Float</td>
<td>double</td>
</tr>
<tr>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>Unsigned Integer</td>
<td>unsignedInt</td>
</tr>
<tr>
<td>LongNVarChar</td>
<td>string</td>
</tr>
<tr>
<td>LongVarBinary</td>
<td>binary</td>
</tr>
<tr>
<td>LongVarChar</td>
<td>string</td>
</tr>
<tr>
<td>NChar</td>
<td>string</td>
</tr>
</tbody>
</table>
Table 90. DataStage-to-XML type mappings (continued)

<table>
<thead>
<tr>
<th>DataStage type</th>
<th>XML type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>decimal</td>
</tr>
<tr>
<td>NVarchar</td>
<td>string</td>
</tr>
<tr>
<td>Real</td>
<td>float</td>
</tr>
<tr>
<td>SmallInt</td>
<td>short</td>
</tr>
<tr>
<td>Unsigned SmallInt</td>
<td>unsignedShort</td>
</tr>
<tr>
<td>Time</td>
<td>time</td>
</tr>
<tr>
<td>Timestamp</td>
<td>dateTime</td>
</tr>
<tr>
<td>TinyInt</td>
<td>byte</td>
</tr>
<tr>
<td>Unsigned TinyInt</td>
<td>unsignedByte</td>
</tr>
<tr>
<td>VarBinary</td>
<td>binary</td>
</tr>
<tr>
<td>VarChar</td>
<td>string</td>
</tr>
</tbody>
</table>

XML-to-DataStage data type mappings
This table shows the data type mappings.

Table 91. XML-type-to-DataStage data type mappings

<table>
<thead>
<tr>
<th>XML type</th>
<th>DataStage data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>VarChar</td>
</tr>
<tr>
<td>dateTime</td>
<td>TimeStamp</td>
</tr>
<tr>
<td>time</td>
<td>Time</td>
</tr>
<tr>
<td>date</td>
<td>Date</td>
</tr>
<tr>
<td>gYear</td>
<td>Date</td>
</tr>
<tr>
<td>gYearMonth</td>
<td>Date</td>
</tr>
<tr>
<td>gMonth</td>
<td>Date</td>
</tr>
<tr>
<td>gMonthDay</td>
<td>Date</td>
</tr>
<tr>
<td>gDay</td>
<td>Date</td>
</tr>
<tr>
<td>anyURI</td>
<td>VarChar</td>
</tr>
<tr>
<td>ENTITY</td>
<td>VarChar</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>VarChar</td>
</tr>
<tr>
<td>ID</td>
<td>VarChar</td>
</tr>
<tr>
<td>IDREF</td>
<td>VarChar</td>
</tr>
<tr>
<td>IDREFS</td>
<td>VarChar</td>
</tr>
<tr>
<td>QName</td>
<td>VarChar</td>
</tr>
<tr>
<td>token</td>
<td>VarChar</td>
</tr>
<tr>
<td>language</td>
<td>VarChar</td>
</tr>
<tr>
<td>Name</td>
<td>VarChar</td>
</tr>
<tr>
<td>NCName</td>
<td>VarChar</td>
</tr>
<tr>
<td>NMTOKEN</td>
<td>VarChar</td>
</tr>
<tr>
<td>NMTOKENS</td>
<td>VarChar</td>
</tr>
<tr>
<td>NOTATION</td>
<td>VarChar</td>
</tr>
<tr>
<td>normalizedString</td>
<td>VarChar</td>
</tr>
</tbody>
</table>
### Files for XML stage examples

The files includes schemas and data files that are used in the example jobs that illustrate how to use steps to create assemblies for the XML stage.

You use these files when you create the example jobs that use the XML stage. For each example, the finished job is available. You can either follow the step-by-step instructions to build each example, or import the complete job and explore it on your own.


<table>
<thead>
<tr>
<th>Example</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1: Parsing XML data</td>
<td>departments.xml (input file), Employee.xsd (schema), Organization.xsd (schema), xml_parser_example.dsx (job), employee.txt (output file), address.txt (output file)</td>
</tr>
<tr>
<td>Example 2: Using the XML Composer and Regroup steps</td>
<td>departments.xml (input file), Employee.xsd (schema), Organization.xsd (schema), employee.txt (input file), address.txt (input file), xml_composer_example.dsx (job), employee_output.xml, (output file)</td>
</tr>
<tr>
<td>Example</td>
<td>Files</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Example 3: Using the XML Composer and HJoin steps</td>
<td>departments.xml (input file), Employee.xsd (schema), Organization.xsd (schema), employee.txt (input file), address.txt (input file), xml_hjoin_example.dsx (job), employee_output_xml (output file)</td>
</tr>
<tr>
<td>Example 4: using the XML Parser and Switch steps</td>
<td>employee1.xml (input file), employee2.xml (input file), Employee.xsd (schema), Organization.xsd (schema), xml_switch_example.dsx (job), valid_emp.txt (output file), InValid_emp.txt (output file)</td>
</tr>
<tr>
<td>Example 5: using the XML Parser and Union steps</td>
<td>employee1.xml (input file), employee2.xml (input file), Employee.xsd (schema), Organization.xsd (schema), Organization1.xsd (union_schemas\schema1), Employee1.xsd (union_schemas\schema1), Employee2.xsd (union_schemas\schema2), Organization2.xsd (union_schemas\schema2), xml_union_example.dsx (job), employees_output.xml (output file)</td>
</tr>
<tr>
<td>Example 6: Using the XML Composer and H-Pivot steps</td>
<td>address.txt (input file), Employee.xsd (schema), Organization.xsd (schema), xml_hpivot_example.dsx (job), address_output.xml (output file)</td>
</tr>
<tr>
<td>Example 7: Using the XML Parser and Aggregate steps</td>
<td>departments.xml (input file), employee.xsd (schema), organization.xsd (schema), xml_aggregate_example.dsx (job), employees_output.txt (output file)</td>
</tr>
<tr>
<td>Example 8: Using the XML Composer and Sort steps</td>
<td>departments.xml (input file), employee.xsd (schema), organization.xsd (schema), xml_sort_example.dsx (job), employees_output.xml (output file)</td>
</tr>
<tr>
<td>Example 9: Using the XML Composer and OrderJoin steps</td>
<td>address.xml (input file), employee.xml (input file), employee1.xsd (schema), organization1.xsd (schema), xml_orderjoin_example.dsx (job), employees_output.xml (output file)</td>
</tr>
<tr>
<td>Example 10: Using the XML Parser and V-Pivot steps</td>
<td>departments.xml (input file), employee.xsd (schema), organization.xsd (schema), xml_vpivot_example.dsx (job), employees_output.txt (output file)</td>
</tr>
</tbody>
</table>

**Messages**

Identify an error or problem and resolve the problem by using the appropriate recovery action.

The following messages appear in the user interface at design time or in the job log at run time. Messages have the following severity levels: Informational, Warning, or Error. The last character in each message number indicates the severity.
Design time messages (XML stage)
These messages might appear when you design a job that includes the XML stage.

**CDIUI2001I** {0} started. Input arguments follow: {1}

*Explanation:* This informational message is seen in the log and indicates that the application has started.

*User response:* None.

**CDIUI2002I** {0} closed.

*Explanation:* This informational message is seen in the log and indicates that the application has ended.

*User response:* None.

**CDIUI2003I** Changed log level from {0} to {1}.

*Explanation:* This informational message occurs when the logging level is changed in the application. The logging “Level” can be changed in the Options dialog, under the heading Logging.

*User response:* No action needed. This is an informational message only.

**CDIUI2004I** Child list includes additional fields due to changes you have made to the assembly.

*Explanation:* A change to the upstream schema has occurred. This change has resulted in additional fields to the child list.

*User response:* This message is informational. No resolution is required.

**CDIUI2006I** This randomly generated data, not actual data, is used to test the assembly.

*Explanation:* The informational message indicates that test data, not input files, is used to test the assembly.

*User response:* None needed.

**CDIUI2201W** Detected step mismatch for deletion.
Removing correct saved step: {0} with id {1}. Ignoring this step {2} with id {3}.

*Explanation:* This message is a warning of a condition that has been detected and corrected regarding assembly step deletion.

*User response:* None.

**CDIUI2202W** This feature is not available in this release.

*Explanation:* The feature that you are trying to use is not available.

*User response:* None.

**CDIUI2203W** XSLT/Xquery document size must be within memory limits.

*Explanation:* Warning that the XSLT/Xquery document must fit within the memory limits defined for this XML Stage.

*User response:* If required, update the XML Stage properties to specify more memory.

**CDIUI2206W** Request to get assembly context ignored, the assembly context id is not yet set.

*Explanation:* An error has been detected that will prevent the application from obtaining the assembly editing session context. This situation detected indicates the application is in the bring-up phase. It must be completely operational to obtain this information. This request could be due to a query by the user from the Options dialog while the application is loading.

*User response:* Retry the operation when the application is fully operational.

**CDIUI2214W** Call for service '{0}' ignored. The path '{1}' associated with this service is no longer valid. Existing mapping follows: {2}.

*Explanation:* A previous mapping is now incorrect, possibly because of an upstream schema change.

*User response:* Correct the mapping or upstream schema, and retry the operation.

**CDIUI2216W** The requested action is in progress. Wait until the request has completed before attempting another action.

*Explanation:* An action was attempted that is invalid because a prior action is still processing.

*User response:* Retry the action after the initial one has completed.

**CDIUI2218W** Warning, you are about to exit this application. You have made changes to the assembly that are not saved.

*Explanation:* The user was asked to save changes before exiting and responded "No." Any changes that were made to the assembly are not saved.

*User response:* None necessary.

**CDIUI2401E** Select an item, and then perform the action.
User response: Some graphical trees in the application require the selection of a tree node before performing an action upon it.

User response: Use the left-mouse button to select the node. Then attempt the action on the node.

Configuration mismatch detected. Received step [0] from server that is not defined in [1] property file. The step will be ignored.

Explanation: This message occurs when a new step type is only partially provisioned by a third party.

User response: Contact the system administrator to ensure that all software updates to the server and UI have been made.

Item [0] was selected for both lists. Choose a different item for each list.

Explanation: For some steps, it is invalid to select the same item for two lists.

User response: Choose a different item for one of the lists and retry the operation.

Pairs of items must be unique. This pair of items was selected twice: [0] and [1]. Select a unique pair of items.

Explanation: An attempt was made to select the same pair of items twice, such as a selection of the same parent and child keys in an HJoin. This message indicates that duplicate selections are not allowed.

User response: Specify a different pair of items for the second selection and retry the operation.

This item was selected twice: [0]. Select a different item for the second selection.

Explanation: An attempt was made to select the item twice, such as a selection of the same key in a Regroup. This message indicates that duplicate selections are not allowed.

User response: Specify a different item for the second selection and retry the operation.

An internal error was detected during the step removal process. Please try again. Make sure the step for removal is not the Overview, Input, or Output step.

Explanation: An attempt was made to remove a step, such as the Input or Output step, which cannot be removed.

User response: Do not attempt to remove this step.

Download failed with error "[0]."

Explanation: Download of an item such as the assembly context or the recording of the client-server transactions failed. The error code shown in this message provides an indication as to why this failure occurred. A download failure typically means the server is no longer available for downloads.

User response: Check that the server used by this application is operational and accessible, and then retry the operation.

Column names must be unique.

Explanation: A column name was entered that has already been chosen.

User response: Retry the operation and specify name that is unique.

Enter a name that begins with an alphabetic character and contains only alphanumeric characters, underscores, dollar signs ($), and periods.

Explanation: A name was entered that is invalid.

User response: Retry the operation and specify a name that contains only alphanumeric characters, underscores, dollar signs ($), and periods.

Enter a name that begins with an alphabetic character or an underscore.

Explanation: A name was entered that is invalid.

User response: Specify a name that begins with an alphabetic character or an underscore.

Another target with that name already exists.

Explanation: A name was entered that is invalid.

User response: Retry the operation and specify a name that is unique.

The '{1}' version of the Adobe Flash Player ActiveX control is not supported. Install a supported version.

Explanation: The application detected an out-of-date version of Adobe Flash Player.

User response: Install the required version of the Flash Player from Adobe.
CDIUI2413E An associated editor was not specified or cannot be found for this step.

CDIUI2414E One or more fields are no longer valid.

Explanation: A field has an invalid value.

User response: Change the value in this field to be valid, or delete the field if applicable.

CDIUI2415E The addition of step type "[0]" failed. You cannot add a step when one is still loading.

Explanation: If a step is added while another step is still loading, this error occurs.

User response: Allow the step to complete loading before adding a new step.

CDIUI2416E A valid mapping target was not selected. Select a valid target item in the mapping table, and then retry the action.

Explanation: A mapping target is chosen by selection of a row to map in the mapping table. A valid row was not selected.

User response: Select a valid row in the mapping table and retry the action.

CDIUI2417E There are no mappings for the selected target item "[0]" because an upstream step has errors. Correct the errors, and then retry the action.

Explanation: An attempt to map a source item to a target item was made, but no mappings suggestions were found for the target item. The lack of mapping suggestions is often caused by errors in an upstream step in the assembly.

User response: Correct the errors in the previous steps, and then retry the action.

CDIUI2418E The source item "[0]" is not a valid mapping choice for the target item "[1]". Possible causes: the mapping results in an invalid type conversion; the source item and target item do not have the same dimension; or the source item is not a child of the previously mapped parent item.

Explanation: An invalid source item was selected for the target item.

User response: Select a valid source item.

CDIUI2501I Either no Output data is available, or you need to test the assembly before Output data can be displayed.

Explanation: This message is seen if a step is testable, but the test assembly action has not been performed.

User response: Click the Test Assembly button and then click Run Test. When the test completes, close the dialog, and retry this operation.

CDIUI2502E The step has errors. Therefore, you cannot test it.

Explanation: This message appears on the Test tab of a step if the step has errors. Only steps that have no errors can be tested.

User response: Go to the Configuration tab of the step, correct the step errors, and then retry the operation.

CDIUI2503E The upstream step has errors; therefore, no output data will be available when running tests.

Explanation: This message displays on the Test tab of a step when the step can be tested, but the preceding step in the assembly has errors. The errors in the preceding step prevent the current step from being tested.

User response: Correct the errors in the preceding step and then test the assembly.

CDIUI2504E There was a problem parsing the XML input test data.

Explanation: The input XML test data must be valid XML.

User response: Correct the input test data and retry the operation.

CDIUI2601E An unrecoverable error has occurred. Method '[0]' failure. Server '[1]' request URI is: '[2]'. Error condition is '[3]'. Client session ID is '[4]'. The application must exit. For more information, see log file '[5]'.

Explanation: This error occurs when an HTTP service call from the UI to the server fails. It can indicate an unexpected error condition; it can also indicate that the server has become unavailable. This information gathered in this message is for the IBM Software Support team.

User response: Check that the server used by this application is operational and accessible and retry the operation when it becomes available. If the problem persists, contact IBM Software Support.
CDIUI2603E  The requested item was not found. It has either been renamed or removed. Either restore the item or refresh the view and retry the action. Server status code is 410.

Explanation: The item requested is not in the contract library.

User response: Check that all of the items are imported in the library and that they are correct. Re-import any items needed. Then retry the operation.

CDIUI2801E  This mapping is invalid. The target item {2} has the data type {1} which cannot be converted to data type {0}.

Explanation: This message is an error that is seen while configuring a step in the assembly editor. It is seen when a source mapping selection for a field would cause a type cast that is not valid.

User response: Either change the field specified to one of a similar type, or change the selection in the mapping list to a field that has a compatible type to the specified field.

CDIUI2607E  The request timeout of {0} seconds occurred before the server responded. It is possible that the server was processing other requests or that the server is unavailable or unreachable. Server URL: '{1}'.

Explanation: The HTTP call placed by the UI did not return before the timeout occurred.

User response: If you have determined the server is functioning and reachable, retry the action again.

CDIUI2608E  The request timeout of {0} seconds occurred before the server responded. It is possible that the server is unavailable or unreachable. Server URL: '{1}'.

The application must exit.

Explanation: This error is seen when an HTTP service call from the UI to the server has not returned before the timeout occurred. It can indicate an unexpected error condition; it can also indicate that the server has become unavailable.

User response: Check the server used by this application is operational and accessible and retry the operation when it becomes available. If the problem persists, contact the IBM Software Support.

CDIUI2802E  Item {1} is not valid for {0}. A prior step modified the schema, either by renaming or deleting the item.

Explanation: A required item is now incorrect, possibly because of an upstream schema change.

User response: Correct the upstream schema, or change the mapping and retry the operation.

CDIUI2606E  A required item is not in the schema because a previous step renamed or deleted the item or its parent. Restore the item, or change the mapping; then retry the action. Server status code is 410.

Explanation: A required item is incorrect, possibly because of an upstream schema change.

User response: Correct the upstream schema or change the mapping and retry the operation.

CDIUI2803E  The mapping of source item {1} to target item {0} is not valid because a previous step modified the schema, by either renaming, deleting or modifying the source item.

Explanation: A required item is now incorrect, possibly because of an upstream schema change that occurred because of a mapping action.
User response: Correct the upstream schema, or change the mapping and retry the operation.

**CDIUI2804E** The value [0] as input for [1] is not valid.

**Explanation:** This message is an error that is seen while configuring a step in the assembly editor. It is seen when an invalid value is detected.

**User response:** Correct the error indicated for this step. Choose a different value for the input specified.

**CDIUI2805E** Input is required for [0].

**User response:** This message is an error that is seen while configuring a step in the assembly editor. In general, seen when performing a mapping such as in the mapping table. It indicates that a mapping has not been performed for the field specified.

**User response:** Perform a mapping for the field specified.

**CDIUI2806E** The [0] expression is not valid: [1].

**Explanation:** This message is an error that is seen while configuring a step in the assembly editor, such as creating an expression in the Switch step. It is seen when the specified expression is not valid.

**User response:** Correct the expression.

**CDIUI2807E** An internal error occurred [0].

**Explanation:** This error indicates an internal error in the server-side services.

**User response:** Examine the context surrounding this error message in the log file. If other errors were logged prior to this message, such as any error indicating the server became unavailable, this message could be due to that prior condition. Follow the actions identified in the prior message. Otherwise, contact IBM Software Support with the information contained in this message.

**CDIUI2808E** Key [0] of type [1] is not compatible with key [2] of type [3].

**Explanation:** This message is an error that is seen while configuring a step in the assembly editor. It indicates the key types are incompatible.

**User response:** Select keys that have compatible types.

**CDIUI2809E** The type [1] of [0] is not a valid key type.

**Explanation:** This message is an error that is seen while configuring a step in the assembly editor. The type of the selected field is not a valid type for a key.

**User response:** Select a key with a valid type.

**CDIUI2810E** There is no matching parent key defined for [0].

**Explanation:** This message is an error that is seen while configuring a step in the assembly editor. A parent key is required but not selected for the specified field.

**User response:** Select a parent key.

**CDIUI2811E** There is no matching child key defined for [0].

**Explanation:** This message is an error that is seen while configuring a step in the assembly editor. A child key is required but not selected for the specified field.

**User response:** Select a child key.

**CDIUI2812E** The mapping of the constant value [0] to the target item [2] is invalid because the constant value cannot be converted to data type [1].

**Explanation:** This message is an error that is seen while configuring a step in the assembly editor. It is seen when a constant mapping is specified that is cannot be converted to the type for the target field.

**User response:** Input a valid constant for the target that is compatible for the specified type.

**CDIUI2813E** Link [0] does not have a column.

**Explanation:** The input or output link in the job does not have a column associated with it.

**User response:** Return to the stage editor and create some columns for the links or create a column in the Input or Output step.

**CDIUI2814E** The child list cannot be contained in the parent list.

**Explanation:** You cannot select a child list that is already contained in the parent list.

**User response:** Select a different child or change the parent list.

**CDIUI2815E** The processing instructions are not valid: [0].

**Explanation:** Invalid processing instructions have been entered.

**User response:** Review the processing instructions and verify that they are valid.
CDIUI2816E  The mapping for target item {0} is not valid because of a missing or invalid ancestor list mapping {1}.

Explanation:  The mapping of this item cannot be performed until its parent is mapped.

User response:  Map the parent item first, then retry this action.

CDIUI2817W  {0} ***CANCELLED*** queued step: '{1}' selected step: '{2}' queue length: '{3}' request URL: '{4}'.

Explanation:  A request to the server has been canceled because of a change to a step.

User response:  It is possible that an assembly update was not made because of this step change. Check the old step to make sure all changes needed were performed.

CDIUI2820E  The mapping is not applicable due to an invalid type conversion or difference in the source and target list dimensions.

Explanation:  The mapping attempted was invalid. Either the source and target types are incompatible. Or when the list dimensions between the source and target are different, such as one versus a two dimensional list.

User response:  Retry the mapping with a valid source for the selected target.

CDIUI2821E  This mapping requires a data type conversion that might cause a runtime error.

Explanation:  The mapping of source and target items may not be convertible at runtime.

User response:  Map the source and target items with items that can be converted at runtime.

CDIUI2822E  This mapping requires a data type conversion that might cause values to lose precision.

Explanation:  At runtime, this type mapping might cause data precision to be lost.

User response:  No action required if this loss is acceptable; otherwise, change the mapping to types that are compatible.

CDIUI2823E  This mapping requires a data type conversion.

Explanation:  The source and target items are not of the same type and a runtime conversion will occur.

User response:  Unless this mapping is invalid, no action is required.

CDIUI2824E  The source data type cannot be converted into the target data type.

Explanation:  The source column data type cannot be converted to the target type.

User response:  Re-map a with a compatible type.

CDIUI2825E  When mapping a target List, the source node must be a List.

Explanation:  This error occurs when you attempt to map a non-list item to a list.

User response:  Map only similar items.

CDIUI2826E  When mapping a content Item (Non-List), the source node must be a content Item (Non-List).

Explanation:  This error occurs when you attempt to map a list item to a non-list item.

User response:  Map only similar items.

CDIUI2903W  Select a pair of keys; otherwise, the step will calculate a Cartesian product.

Explanation:  This message occurs when no keys have been specified for a step such as the HJoin step.

User response:  Select a pair of keys for the step.

CDIUI2904W  Data loss is possible for the data type conversion from type {0} to type {1} for item {2}.

Explanation:  This message occurs when a type conversion might result in data loss – for example, a conversion from a decimal number to an integer.

User response:  If data loss is acceptable, ignore this message; otherwise, choose an item that has a type that will not result in a loss of data.

CDIUI2905W  A runtime data conversion error might result in data loss when converting from data type {0} to data type {1} for item {2}.

Explanation:  This message occurs when a type conversion might result in data loss – for example, a conversion from a decimal number to an integer.

User response:  If data loss is acceptable, then ignore this message; otherwise, choose an item that has a type that will not result in a loss of data.

CDIUI3001I  Validation is not available until the previous tabs are completed.

Explanation:  The validation for this step is dependent on the completion of prior actions in the step.

User response:  Perform any prior actions to the
validation tab. Then retry the operation.

**CDIUI4401E** The specified file must have the file extension *.xsd.*

Explanation: The import operation in the “Libraries” tab of the application detected an attempt to upload a file that is not valid type. Only files of type XSD is allowed.

User response: Retry the import, specifying a type of file that is allowed.

---

**CDIUI4402E** The library failed validation because one or more resources failed validation.

Explanation: The validation of a contract library failed.

User response: Examine the detailed validation messages and correct the errors indicated.

---

**CDIUI4403E** One or more resources failed validation. Click Validate to view the errors.

Explanation: The validation of a contract library failed.

User response: Examine the detailed validation messages and correct the errors indicated.

---

**CDIUI4605E** Library names must be unique. Enter a unique name.

Explanation: A library create failed because the library with the specified name already exists.

User response: Specify a different name when creating a new library.

---

**CDIUI4606E** Unable to display the library type list due to an XML parsing error. The library must parse cleanly before the type list can be displayed. Server status code is 409.

Explanation: Only libraries that are valid (parse) can be opened. An attempt to open a library with errors to view its contained types list was detected.

User response: Correct the library parse errors, validate the library, and then retry the operation.

---

**CDIUI4607E** Unable to display the requested type schema due to an XML parsing error. The library must parse cleanly before its types can be displayed. Server status code is 409.

Explanation: Only libraries that are valid (parse) can be opened. An attempt was made to view a type schema in a contract library.

User response: Correct the library parse errors, validate the library, and then retry the operation.

---

**CDIUI4608E** Unable to display the library service list due to an XML parsing error. The library must parse cleanly before the service list can be displayed. Server status code is 409.

Explanation: Only libraries that are valid (parse) can be opened. An attempt to open a library with errors to view its contained services list was detected.

User response: Correct the library parse errors, validate the library, and then retry the operation.

---

**CDIUI4609E** The requested item was not found. It has either been renamed or removed. Either restore the item or refresh the view and retry the action. Server status code is 410.

Explanation: Only libraries that are valid (that parse) can be opened. An attempt was made to view a service in a contract library.

User response: Correct the library parse errors, validate the library then retry the operation.

---

**CDIUI4610E** A resource with the same file location already exists.

Explanation: The operation attempted is invalid because the resource already exists in the library.

User response: Specify a different resource and retry the operation.

---

**CDIUI4611E** The requested resource was not found in the specified library.

Explanation: The operation attempted is invalid because the resource requested is not on the library. This could happen if the library is out of sync with the resource view, possibly due to actions performed by another user on the same library.

User response: Refresh the view of the contract library using the refresh icon and then select the library again to view its resources.

---

**CDIUI4612E** Specify the file location of the resource.

Explanation: An attempt to import a resource into the contract library without specifying a file location was made. A file location is required.

User response: Retry the operation with a file location.
User response: Check temp file directory permissions and available disk space. Contact support if the problem cannot be resolved.

**CDIUW2021E** Unable to write file `{0}`.

Explanation: Launching the Assembly Editor from the stage editor dialog, or launching the Schema Library Manager from the Designer Import menu. This message can also appear in the log.

User response: Check the specified file path to see whether the target directory exists and is accessible. Contact support if the problem cannot be resolved.

User response: Check the server used by this application is operational and accessible and retry the operation when it becomes available. If the problem persists, contact IBM Software Support.

**CDIUW2022E** Unable to create log file in directory `{0}`.

Explanation: Launching the Assembly Editor from the stage editor dialog, or launching the Schema Library Manager from the Designer Import menu.

User response: Check the specified directory path to see whether the directory exists and is accessible. Contact support if the problem cannot be resolved.


**CDIUW2023E** Unable to determine the log file directory.

Explanation: Launching the Assembly Editor from the stage editor dialog, or launching the Schema Library Manager from the Designer Import menu.


**CDIUW2024E** Unable to open `{0}`.

Explanation: Launching the Assembly Editor from the stage editor dialog, or launching the Schema Library Manager from the Designer Import menu.


**CDIUW2025E** The application server is unavailable or not operational.

Explanation: Launching the Assembly Editor from the stage editor dialog, or launching the Schema Library Manager from the Import menu in the InfoSphere DataStage and QualityStage Designer.

User response: Check the server used by this application is operational and accessible and retry the operation when it becomes available. If the problem persists, contact IBM Software Support.

**CDIUW2026E** An unrecoverable error has occurred. Server `{0}` request URI is `{1}`. Error condition is `{2}`. Client session ID is `{3}`. The application must exit. For more information, see log file `{4}`.

Explanation: Launching the Assembly Editor from the stage editor dialog, or launching the Schema Library Manager from the Designer Import menu.

User response: Check the server used by this application is operational and accessible and retry the operation when it becomes available. If the problem persists, contact IBM Software Support.

**Runtime messages (XML stage)**

These messages might display when you run a job that includes the XML stage.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIER0001E</td>
<td>The XML namespace was not defined in the schema, name={1}, line number={2}, column number={3}.</td>
</tr>
<tr>
<td>CDIER0002E</td>
<td>The XML element was not defined in the schema, name={1}, line number={2}, column number={3}.</td>
</tr>
<tr>
<td>CDIER0003E</td>
<td>The XML type was not defined in schema, name= {1}, line number= {2}, column number= {3}.</td>
</tr>
<tr>
<td>CDIER0004E</td>
<td>The XML attribute was not defined in schema, name={1}, line number={2}, column number={3}, source={4}.</td>
</tr>
<tr>
<td>CDIER0006E</td>
<td>Unable to create the XML input stream, cause={0}.</td>
</tr>
<tr>
<td>CDIER0007E</td>
<td>An error occurred while reading the XML: cause={1}, systemId={2}, line number={3}, column number={4}, step={5}, source={0}.</td>
</tr>
<tr>
<td>CDIER0008E</td>
<td>The value for this XML attribute must be a fixed value: attribute name={0}, fixed value={1}, value found={2}, item name={3}, line number={4}, column number={5}, source={6}.</td>
</tr>
<tr>
<td>CDIER0009E</td>
<td>Missing required XML attribute: attribute name={0}, item name={1}, line number={2}, column number={3}, source={4}.</td>
</tr>
<tr>
<td>CDIER0010E</td>
<td>One or more required XML elements are missing from element with content type 'all': name of parent element={0}, missing element={1}, line number={2}, column number={3}, source={4}.</td>
</tr>
<tr>
<td>CDIER0011E</td>
<td>Missing XML data type: type name={0}, item name={1}, line number={2}, column number={3}.</td>
</tr>
<tr>
<td>CDIER0012E</td>
<td>The value specified for this data type is invalid: item name={0}, line number={1}, column number={2}, name={3}, cause={1}.</td>
</tr>
<tr>
<td>CDIER0013E</td>
<td>A duplicate value was specified for this ID attribute: value={0}, item name={1}, line={2}, column={3}, step={4}, source={5}.</td>
</tr>
<tr>
<td>CDIER0014E</td>
<td>If an element has content, the xsi:nil attribute value cannot be set to true, an error occurred because of an attempt to assign xsi:nil='true' to an element that contains data: value={0}, line number {1} column number={2}, item name={3}.</td>
</tr>
<tr>
<td>CDIER0015E</td>
<td>The data for this item does not conform to the data type: step={5}, item name={1}, toType={0}, value={2}, line={3}, column={4}, source={6}.</td>
</tr>
<tr>
<td>CDIER0016E</td>
<td>Non-whitespace characters are invalid in non-mixed case content in XML data: item name={1}, line number={2}, column number={3}.</td>
</tr>
<tr>
<td>CDIER0017E</td>
<td>An unexpected start or end element name was encountered: line number={1}, column number={2}, actual name={3}, expected name={4}.</td>
</tr>
<tr>
<td>CDIER0018E</td>
<td>A missing start or end element name was encountered: line number={1}, column number={2}, item name={3}.</td>
</tr>
<tr>
<td>CDIER0019E</td>
<td>A missing end element name was encountered: item name={0}, line number={1}, column number={2}.</td>
</tr>
<tr>
<td>CDIER0020E</td>
<td>A missing start element name was encountered: item name={0}, line number={1}, column number={2}.</td>
</tr>
<tr>
<td>CDIER0021E</td>
<td>An unexpected element was encountered: item encountered={0}, line number={1}, column number={2}, expected event names={3}, source={4}.</td>
</tr>
<tr>
<td>CDIER0022E</td>
<td>An element that is not nillable was set to xsi:nil='true': item name={0}, line number={1}, column number={2}, source={3}.</td>
</tr>
<tr>
<td>CDIER0023E</td>
<td>The number of array elements exceeds the specified maximum: item name={0}, number of elements={3}, maximum number allowed={4}, line number={1}, column number={2}, step={5}, source={6}.</td>
</tr>
</tbody>
</table>
The number of array elements is fewer than the specified minimum: item name={0}, number of elements={3}, minimum number allowed={4}, line number={1}, column number={2}, step={5}, source={6}.

Missing required item: item name={0}.

The data for this item does not conform to the item facet: source type={0}, item name={1}, facet={2}, value={3}, line={4}, column={5}, step={6}, source={7}.

An element that is not nillable was set to xsi:nil='true': item name={0}.

This output file cannot be created: output file={0}, exception={1}.

The data is illegal for the target item type and cannot be converted: target type={0}, item name={1}, value={2}, step={3}.

The data does not conform to this facet: type={0}, value={1}, facet={2}, step={3}.

Invalid data for target item type and cannot be converted. step={3}, targetItem={4}, source type={0}, target type={1}.

The source type cannot be converted to the target type: step={2}, target item={3}, source type={0}, target type={1}.

The facet type cannot be converted to the target type: step={3}, target item={4}, target type={0}, facet={2}, value={1}.

The data does not conform to the column definition: step={3}, target item={4}, target type={0}, attribute={2}, value={1}.

An error occurred during XML parsing because={0}.

Cannot read file={0}.

Cannot write to file={0}.

XSLT exception encountered, cause={0}, description={1}.

XML parsing exception encountered={0}.

SOAP exception encountered={0}.

XQuery exception encountered={0}.

Attempted to assign a null value to an output link column that is not nullable in the output step: column={0}.

The input data in a comma separated format file used in a test assembly does not match the input schema, step={0}, column={1}.

Error in step={1}, cause={0}.

The Expression written in the Switch step for filtering records failed={0}.

The Expression written in the Switch step for filtering records failed={0}, result type={1}.

Expression compilation failed, cause={0}.

Invalid expression: mapping={0}, cause={1}.

Cannot create directory={0}.

Directory name is not a directory, name={0}.

Cannot write to directory={0}.

Cannot delete file={0}.
**Working with IBM Software Support to diagnose problems**

Use these settings when you work with IBM Software Support to solve problems.

Choose **Options** in the Assembly Editor to display the following settings:

**Assembly Context File**
- Click **Download** to save a copy of the assembly context file. This file, which has the extension e2d, is useful for problem determination and testing.

**Logging**
- The file location of the log file and the log level.

**Server Tracing**
- Enable tracing to record a trace log.

**Flash Plug-in Diagnostics**
- This is informational.

**Application Diagnostics**
- This is informational.

**Note:** If you change the default settings, they apply only to the current assembly.

**Links to developer articles**

Use the developer article links in this topic for more information about XML stage.

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](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](http://www.ibm.com/able/about) for more information about the commitment that IBM has to accessibility.
Appendix B. Reading command-line syntax

This documentation uses special characters to define the command-line syntax.

The following special characters define the command-line syntax:

- [ ] Identifies an optional argument. Arguments that are not enclosed in brackets are required.
- ... Indicates that you can specify multiple values for the previous argument.
- | Indicates mutually exclusive information. You can use the argument to the left of the separator or the argument to the right of the separator. You cannot use both arguments in a single use of the command.
- {} Delimits a set of mutually exclusive arguments when one of the arguments is required. If the arguments are optional, they are enclosed in brackets ([ ]).

**Note:**
- The maximum number of characters in an argument is 256.
- Enclose argument values that have embedded spaces with either single or double quotation marks.

For example:

```
wssetsrc [-S server] [-l label] [-n name] source
```

The `source` argument is the only required argument for the `wssetsrc` command. The brackets around the other arguments indicate that these arguments are optional.

```
wlsac [-l | -f format] [key...] profile
```

In this example, the `-l` and `-f format` arguments are mutually exclusive and optional. The `profile` argument is required. The `key` argument is optional. The ellipsis (…) that follows the `key` argument indicates that you can specify multiple key names.

```
wrb -import {rule_pack | rule_set}...
```

In this example, the rule_pack and rule_set arguments are mutually exclusive, but one of the arguments must be specified. Also, the ellipsis marks (…) indicate that you can specify multiple rule packs or rule sets.
Appendix C. How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The '-----' symbol indicates the beginning of a syntax diagram.
  - The '---' symbol indicates that the syntax diagram is continued on the next line.
  - The '----' symbol indicates that a syntax diagram is continued from the previous line.
  - The '----<' symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).

```
-----required_item
```

- Optional items appear below the main path.

```
-----required_item
       \optional_item
```

If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

```
-----required_item
       \optional_item
```

- If you can choose from two or more items, they appear vertically, in a stack.
  If you must choose one of the items, one item of the stack appears on the main path.

```
-----required_item
     \required_choice1
     \required_choice2
```

If choosing one of the items is optional, the entire stack appears below the main path.

```
-----required_item
     \optional_choice1
     \optional_choice2
```

If one of the items is the default, it appears above the main path, and the remaining choices are shown below.

```
-----required_item
     \default_choice
     \optional_choice1
     \optional_choice2
```

- An arrow returning to the left, above the main line, indicates an item that can be repeated.
If the repeat arrow contains a comma, you must separate repeated items with a comma.

A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.

**Fragment-name:**

- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown.
- Variables appear in all lowercase italic letters (for example, `column-name`). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).
Appendix D. 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 93. 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 E. 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&wv=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/](http://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).
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, LACS\textsuperscript{link}, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACS\textsuperscript{link} licensee of the United States Postal Service.

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

A
Aggregate step 130
assemblies 124
aggregate 131
creating the assembly 131, 143, 159, 171
creation 78, 87, 97, 110, 131, 143, 159, 171, 194
data examples 64, 180
JSON stage 180
order join 159
overview 8, 10
sort 143
testing 12
Union example 110
vpivot 171
XML stage 64
XML Stage 131, 143, 159, 171
Assembly Editor
access 11, 67, 183
overview 10
assembly steps
Aggregate 134
Aggregate step 60
configure XML composer step 149
configuring sort step 146
configuring Union step 113
data files
assembly context 25, 41
H-Pivot step 61
HJoin 61, 84, 87
input 9
JSON Composer 190, 198
Order Join 62
output 9
overview 8
Regroup 62, 75, 79, 192, 195
Sort 62, 146, 149
Switch 62, 100
Union 64, 113, 118
V-Pivot 64
vpivot 174
XML Composer 75, 81, 84, 88, 118, 149
XML stage 60

C
columns
displaying details 16
command-line syntax
conventions 221
commands
syntax 221
composing 141, 157
configuration 121
configuring order join step 162
order join 162
OrderJoin 162
Configure
calculate XML composer step 163
OrderJoin 163
XML Composer 163
configuring H-Pivot step 124
Creating the job 122
customer support
contacting 225

D
data parsing 130, 141, 157, 169
default values
XML parsing 32
Developer article links 217
Developer articles 217

E
developer example 121, 127, 130, 141, 157, 169
developer examples
JSON stage 180
Switch 204
Union 204
XML stage 64, 204

F
data files
XML examples 204

H
H-Pivot 122, 124, 125
H-Pivot example 124, 127
H-Pivot step 121, 127
HJoin step
configuration 61, 87

I
icons
schema tree 13
input step
overview 9
item mapping
configuration 24
overview 19
XML stage 18
items
mapping 18
XML 18

J
jobs 93, 107, 122
JSON parser step
efile 180
example 180
JSON Parser step
calculate configuration 180, 185
data examples 180, 183, 184, 189
JSON schemas
importing 184
JSON stage
data parsing 41
data examples 180

L
legal notices 229

M
mapping 125
messages
XML stage 205

O
order join 157
Order Join step
configuration 62
OrderJoin step 157
output step
overview 9
Output step
Aggregate step 136
configuration 70, 102, 118, 136, 150, 164, 175, 185
order join 164
OrderJoin step 164
Sort step 130
Union output step 118
vpivot 175
VPivot step 175
overview step
overview 8

P
parsing 130, 169
product accessibility
accessibility 219
product documentation
accessing 227

R
Regroup step
configuration 62

S
Schema Library
creating an example json schema
schema library 40