Conversion Agent Studio User Guide

SAP Conversion Agent by Informatica
(Version 8.5)

SAP Conversion Agent Studio User Guide

Version 8.5
June 2008

Copyright (c) 2001–2008 Informatica Corporation. All rights reserved.

This software and documentation contain proprietary information of Informatica Corporation and are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright law. Reverse engineering of the software is prohibited. No part of this document may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording or otherwise) without prior consent of Informatica Corporation. This Software is protected by U.S. Patent Numbers and other Patents Pending.

Use, duplication, or disclosure of the Software by the U.S. Government is subject to the restrictions set forth in the applicable software license agreement and as provided in DFARS 227.7202-1(a) and 227.7702-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (OCT 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as applicable.

The information in this software and documentation is subject to change without notice. Informatica Corporation does not warrant that this software or documentation is error free.

Informatica, PowerCenter, PowerCenterRT, PowerCenter Connect, PowerCenter Data Analyzer, PowerExchange, PowerMart, Metadata Manager, Informatica Data Quality, Informatica Data Explorer, Informatica Complex Data Exchange, Informatica On Demand Data Replicator, and Informatica B2B Data Exchange are trademarks or registered trademarks of Informatica Corporation in the United States and in jurisdictions throughout the world. All other company and product names may be trade names or trademarks of their respective owners.

Portions of this software and/or documentation are subject to copyright held by third parties, including without limitation: Copyright © Sun Microsystems. All rights reserved. Copyright 1985-2003 Adobe Systems Inc. All rights reserved. Copyright 1996-2004 Glyph & Cog, LLC. All rights reserved.

This product includes software developed by Boost (http://www.boost.org/). Permissions and limitations regarding this software are subject to terms available at http://www.boost.org/LICENSE_1_0.txt.

This product includes software developed by Mozilla (http://www.mozilla.org/). Your right to use such materials is set forth in the GNU Lesser General Public License Agreement, which may be found at http://www.gnu.org/licenses/lgpl.html. The Mozilla materials are provided free of charge by Informatica, “as-is”, without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/) which is licensed under the Apache License, Version 2.0 (the “License”). You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This product includes software developed by SourceForge (http://sourceforge.net/projects/mpxj/). Your right to use such materials is set forth in the GNU Lesser General Public License Agreement, which may be found at http://www.gnu.org/licenses/lgpl.html. The SourceForge materials are provided free of charge by Informatica, “as-is”, without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

This product includes Curl software which is Copyright 1996-2007, Daniel Stenberg, <daniel@haxx.se>. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http://curl.haxx.se/docs/copyright.html. Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

This product includes ICU software which is Copyright (c) 1995-2003 International Business Machines Corporation and others. All rights reserved. Permissions and limitations regarding this software are subject to terms available at http://www-306.ibm.com/software/globalization/icu/license.jsp.

This product includes OSSP UUID software which is Copyright (c) 2002 Ralf S. Engelschall, Copyright (c) 2002 The OSSP Project Copyright (c) 2002 Cable & Wireless Deutschland. Permissions and limitations regarding this software are subject to terms available at http://www.opensource.org/licenses/mit-license.php.

This product includes Eclipse software which is Copyright (c) 2007 The Eclipse Foundation. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http://www.eclipse.org/org/documents/epl-v10.php.

libstdc++ is distributed with this product subject to the terms related to the code set forth at http://gcc.gnu.org/onlinedocs/libstdc++/17_intro/license.html.

DISCLAIMER: Informatica Corporation provides this documentation “as is” without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of non-infringement, merchantability, or use for a particular purpose. The information provided in this documentation may include technical inaccuracies or typographical errors. Informatica could make improvements and/or changes in the products described in this documentation at any time without notice.

Part Number: DT-SUG-86000-0001

Table of Contents

Preface

The *Conversion Agent Studio User Guide* is written for developers, analysts, and other users who are responsible for designing and implementing transformations. The book explains how to design, configure, test, and deploy transformations by using Conversion Agent Studio. It contains detailed reference sections documenting the transformation components and their properties.

Before reading this book, you need a basic knowledge of how to use Conversion Agent. You can obtain that knowledge by performing the hands-on lessons in *Getting Started with Conversion Agent*.

In parallel with this book, you can refer to *Using Conversion Agent in Eclipse* for instructions on using the Studio menus, toolbars, views, and editors.

Obtaining Conversion Agent Documentation

On Windows platforms, the Conversion Agent documentation is supplied as online help with the software. You can download PDF copies of the Conversion Agent manuals from:

http://help.sap.com/saphelp_nw70/helpdata/en/43/fc39c16bfb025ee10000000a1553f7/content.htm

CHAPTER 1

Designing Transformations

This chapter includes the following topics:

Overview

Conversion Agent Studio is the design and configuration environment of the SAP Conversion Agent system. Using Conversion Agent Studio, you can design and implement transformations that operate on any kind of data.

This book is a learning and reference manual for designing transformations. The book contains:

- Explanations of the Conversion Agent concepts
- Details on how to use all the Conversion Agent components, such as parsers, serializers, transformers, mappers, anchors, and actions
- Examples and tips on how to design transformations that work with many different kinds of input and output
- Instructions for deploying transformations that you have designed in Conversion Agent Studio to the Conversion Agent Engine runtime environment

Transformation Architecture

When you construct a transformation, you build it in modular fashion from components of the Conversion Agent system. The components are arranged in a hierarchy or tree, which you can view in the IntelliScript editor of Conversion Agent Studio.

The components work with input and output documents and with the data holders that store Conversion Agent data.

This section provides a brief overview of the components and terminology that are used in this architecture. For detailed information about each component type, see the following chapters of this book.

Transformation Components

Top-Level Components

At the top level of the hierarchy, a transformation can run a parser, serializer, mapper, transformer, or streamer. These components are defined as follows:

Component	Description
Parser	A component that converts source documents in any format to XML.
Serializer	A component that converts XML documents to output documents in any format.
Mapper	A component that converts XML documents to a different XML structure or schema.
Transformer	A component that modifies data. The input and output can be in any format.
Streamer	A component that splits large inputs into segments that are processed separately by the other components.

Of these component types, parsers, serializers, and mappers are the most powerful and generally useful. By running a parser and serializer in sequence, for example, you can convert any format to any other format. Using these components, you can perform conversions of unlimited complexity.

As top-level components, transformers are useful for relatively simple data conversions, such as replacing predefined strings. Usually, the input and output documents have the same, non-XML format. Because of this limitation, transformers are more often used as nested components, and not as top-level components. For more information, see “Nested Components” on page 2.

Streamers are special-purpose components for processing large inputs such as gigabyte data streams. They do not perform transformations on their own. Instead, they activate other components such as parsers to perform the transformations.

Note: There is a distinction between transformation, which is the generic term for the operations that Conversion Agent performs on data, and transformer, which is a specific type of Conversion Agent component.

Nested Components

Within a parser, serializer, or mapper component, you can nest components such as:

Component	Description
Formats	Define the overall format of documents, such as the delimiters, that Conversion Agent should use to interpret the documents.
Document processors	Operate on a document as a whole, performing preliminary or final conversions.
Anchors	Define the data in a source document that a parser should process and extract. The anchors specify how a parser should search for the data and where it should store the data that it finds.
Serialization anchors	Define how a serializer should write XML data to an output document. Serialization anchors are the inverse of anchors. An anchor writes data from a source document to XML, whereas a serialization anchor writes data from XML to an output document.
Mapper anchors	Define how a mapper should write XML data to another XML structure or schema. The anchors specify where to find the data in the source XML and where to write the data in the output XML.
Actions	Perform operations on data in the scope of a transformation, for example, concatenating strings that a parser has extracted from a source document, summing numbers that a serializer finds in an XML input document, or querying a database for additional data.
Transformers	In addition to their use as top-level components, you can nest transformers within a parser or a serializer. For example, within a parser, you can nest a transformer that modifies the output of the anchors. As a nested component, a transformer operates on a portion of a document, not on the complete document.

Indirectly, you can also nest parsers and serializers within each other. For example, within a parser, you can nest an action that runs another parser on a portion of the same document or on a second document.

Subcomponents

In addition to the main components that are described above, Conversion Agent has a large number of subcomponents that are used for special purposes within the main components. It is also possible to develop custom components, such as custom document processors or custom transformers, to serve special needs.

Data Holders

Data holders are the XML elements, XML attributes, and variables that transformations use for data storage.

The elements and attributes are defined in XSD schemas, which are standard XML schema definitions. Conversion Agent uses XSD to define data holders, to help it process XML input, and to help it construct valid XML output.

The variables are defined in the Conversion Agent configuration, using XSD data types.

For more information about data holders and XSD schemas, see “Data Holders” on page 51.

Documents

The input and output of a transformation are called the source document and the output document.

A document can have any size. It can contain any text or binary data. It can be stored or accessed in a file, buffer, stream, database, messaging system, or any other location.

For a parser, the source document can have any format. The output document of a parser has an XML format.

The source document of a serializer is XML, and the output document can have any format. For a mapper, both the source and the output are XML. For a transformer, the source and output can have any format.

A special source document is called the example source. The example source is a document associated with a parser, serializer, and mapper, used to help configure and test the transformation.

XML is the common language connecting transformations together. For example, you can run a parser that converts a source document from any format to XML, and a serializer that converts the XML to any output format. By chaining the parser and serializer together, you can convert any input format to any output format.

For more information about document structures, see “Formats” on page 39.

Conversion Agent Services

After you configure a transformation, you can deploy it as a Conversion Agent service. Deployment allows Conversion Agent Engine to execute the transformation.

A Conversion Agent service can run a parser, serializer, mapper, transformer, or streamer as its top-level component.

For more information, see “Deploying Conversion Agent Services” on page 231.

Project Architecture

A transformation is stored in a project. Each project has a project folder that contains the project files.

Required Project Files

A project has two required types of files:

File	Description
CMW file	The CMW file is the main project configuration file. Every project contains exactly one CMW file.
TGP script files	A TGP script file defines the components that perform a transformation. A project can contain one or more script files. The scope of a script file is the entire project. This means, for example, that a parser in one script file can use a variable that is defined in another script file. You can import or copy the script files between projects. You can organize the project components in different script files.

Additional Project Files

In addition to the CMW and TGP script files, a project can contain many other files and folders, for example:

Component	Description
XSD schemas	Most transformations require an XSD schema defining the XML elements and attributes that the transformation can use. Parsers, serializer, and mappers use XSD schemas to define the structure of their XML output or input. The main schema is usually stored in the top-level project folder. If the main schema includes other schema files, they are stored in a subfolder.
Results folder	As you develop and test a project, Conversion Agent Studio creates a subfolder, within the project folder, where it stores the parser or serializer output. By default, the name of the subfolder is `Results`. When you deploy a service and run it in Conversion Agent Engine, you can continue to use the `Results` folder, or you can instruct the service to store its output in other locations.
Example source document	A sample of the input that a parser, serializer, or mapper processes. You can use the example source to help develop and test a transformation. You can provide the example source as a file, a text string, or any other document type that Conversion Agent can process. If the example source is a file, store it in the project folder. This helps keep the example source together with the project. For more information, see “Documents” on page 3.
Test documents	In addition to the example source, you can store other input documents that you use to test a transformation.
Any other desired files	The project folder is a convenient location to store any other files or folders that are associated with a project, such as documentation or readme files.

Workflow for Designing Transformations

The following procedure is a summary of a typical workflow for using Conversion Agent Studio and developing a transformation. The example is a workflow that you might use to configure a new parser or serializer.

1. Analyze the transformation requirements, such as the required inputs and outputs.
2. Create and configure a Conversion Agent project that implements the transformation.
3. Deploy the transformation as a service that runs in Conversion Agent Engine.

The following paragraphs provide more information on the steps.

Analyzing the Inputs and Outputs

The first step in any design or development project is to examine the inputs and outputs carefully and determine how they are related.

For a parser, some of the analysis steps are to:

- Examine whether the document structure is amenable to parsing. Sometimes, a simple step such as converting the document to an alternative format, which you can do by applying a document processor, makes the document much easier to parse.
- Plan which data you need to extract from the source document and where you will insert the extracted data in the XML. In Conversion Agent, you implement the data extraction by using the `Content` anchor.
- Analyze the structure of the source document and identify the features you can use to locate the data fields. In Conversion Agent, these features translate to `Marker` anchors or to various other types of anchors.
- Find repetitive or structured features of the documents that might help extract the data. You can implement such features using anchors such as `Group`, `EnclosedGroup`, `RepeatingGroup`, or `DelimitedSections`.
- Decide whether you need to transform any of the data during or after the extraction process. The Conversion Agent components that operate on the extracted data are transformers and actions.
- Determine whether there are any additional data sources, such as a linked document or a database, that you need to access to prepare the output. Access such data by using certain anchors, transformers, or actions.

For a serializer or a mapper, you can invert the steps. Plan the data that you need to extract from the source XML and where to insert it in the output document. It is often easier to design a serializer or a mapper than a parser because the input is fully structured XML.

Creating and Configuring a Project

After you analyze the inputs and outputs, it is time to implement the processing steps in Conversion Agent.

To create and configure a new project:

1. Create a new Conversion Agent project.
2. Add one or more XSD schemas to the project.

   The schemas must define the XML elements and attributes with which you will work.
3. Create the top-level transformation component such as a parser or serializer, and define its properties.

   For a parser, you might define an example source document and a format component. The format component can contain features such as a delimiters definition and transformers.

   For a serializer, you might define properties such as the type of output file. You can also create a serializer automatically, by inverting a parser that you have already created.

   The parser or serializer appears in the IntelliScript editor of Conversion Agent Studio. The example source appears in the example pane of the IntelliScript editor.
4. Configure the subcomponents of the transformation.

   If the top-level component is a parser, the main components are anchors. You can define the anchors by graphical procedures or you can edit the IntelliScript. Conversion Agent Studio helps you do this by highlighting and color-coding the anchors in the example source.

   For a serializer, the main components are serialization anchors, which you can create in the IntelliScript.
5. Use the Conversion Agent Studio tools to test and execute the transformation. Correct any configuration errors that you detect during the testing.

Deploying the Project as a Conversion Agent Service

When the transformation is operating correctly, use the Deploy command to make it available as a Conversion Agent service, which can run in the Engine.

Online Samples

As you use this book, you can view online samples that illustrate many Conversion Agent features. The samples are Conversion Agent projects, located in the `samples` subfolder of the main Conversion Agent installation folder. The default location is:

```
c:\Program Files\SAP\ConversionAgent\samples
```

To view the samples, import the projects to Conversion Agent Studio. For more information about importing projects, see *Using Conversion Agent Studio in Eclipse*.

In addition to the project samples, the `samples` folder contains sample program code for features such as custom processors and transformers.

CHAPTER 2

Using Conversion Agent Studio

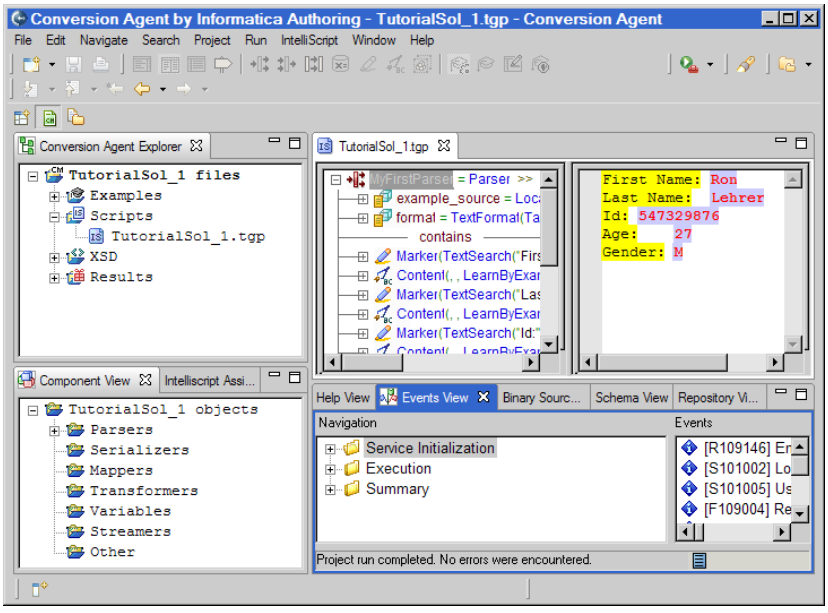
This chapter includes the following topic:

Overview

Conversion Agent Studio is the design and configuration environment of Conversion Agent. You use it to develop and edit Conversion Agent projects.

If you have performed the exercises in *Getting Started with Conversion Agent*, you already have experience using Conversion Agent Studio. The exercises teach many aspects of the Studio operation.

Figure 2-1. Conversion Agent Studio



Using the Studio in Eclipse

Conversion Agent Studio runs as a plug-in module hosted in the Eclipse development environment.

Eclipse is a versatile platform, designed to support both Java development and plug-in development tools. Conversion Agent Studio works seamlessly within Eclipse, allowing you to develop transformations easily.

The Eclipse platform is supplied at no additional cost with the Conversion Agent software. You do not need any previous experience with Eclipse to use Conversion Agent Studio.

For more information about using the Eclipse platform, see *Using Conversion Agent Studio in Eclipse*.

CHAPTER 3

Parsers

This chapter includes the following topics:

Creating a Parser

Parsers are Conversion Agent components that convert a source document to XML.

The output of a parser is always XML. The input can have any format, such as text, HTML, Word, PDF, or HL7. The input can even be an XML document that the parser processes as string data.

This chapter explains the procedures for creating and running a parser component. Further information, such as how to support specific document formats and how to define the anchors that process the text of a source document, is in the succeeding chapters.

You can create a parser by either of the following methods:

- By using the New Parser wizard
- By editing the IntelliScript and inserting a `Parser` component

Using the New Parser Wizard to Create a Parser

The easiest way to create a parser is by using the New Parser wizard. The wizard lets you configure the parser with typical options.

After you finish the wizard, you can edit the parser properties as required. Nested within the parser, you can insert components that perform the parsing operations, such as document processors, anchors, transformers, and actions.

To create a new project that contains a parser:

1. Click File > New > Project.
2. Under the Conversion Agent category, select a Parser Project and click Next.
3. Follow the wizard prompts to enter the parser options.

When you finish, the Conversion Agent Explorer view displays the new project containing the parser. The Component view displays the parser.

To create a new parser in an existing project:

1. Click File > New > Parser.
2. Follow the wizard prompts to enter the parser options.

   When you finish, the Conversion Agent Explorer view displays a new TGP script file defining the parser. The Component view displays the parser.

The following table describes the wizard options. At each stage, the wizard suggests options that seem appropriate based on your previous entries. If the wizard does not suggest the precise options that you need, you can refine the configuration afterwards by editing the IntelliScript.

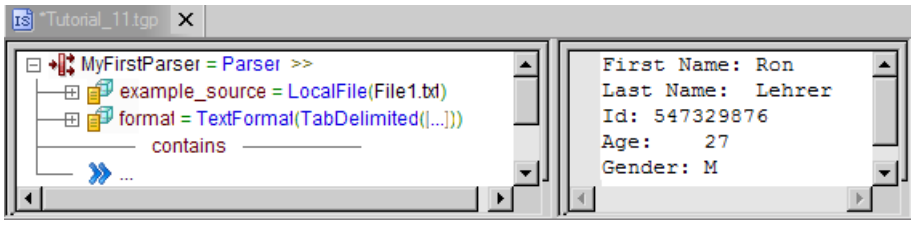
Table 3-1. Options in the New Parser Wizard

Option	Description
Project name	An identifier for the project.
Project contents	The storage location of the project folder. The default is the Studio workspace folder.
Parser name	A name for the parser.
Script name	A name for a TGP script file, where the wizard stores the parser definition.
Schema file path	The name of an XSD schema that defines the XML structure of the parser output.
Source type, source path	Define an example source document that you will use to configure and test the parser. The example source should illustrate the features that you expect the parser to process in production documents. Choose the example source carefully. After you configure a parser, it might be difficult to change the example source. You can select the following source types: - **File**: Browse to a file on the local computer or network. - **URL**: Specify the URL of a document on a network or web site. - **Text**: Type a text string that the parser will use as an example source. - **None**: Do not use an example source. You can add an example source later, or you can configure the parser without using an example source.
Content type	Select the content type of the source documents, such as ASCII or Binary.
Document preprocessor	If required, select a document processor that converts the source documents to a format that is amenable for parsing. The wizard suggests processors that seem appropriate for the content type. For example, if you select a Microsoft Word example source, the wizard suggests processors that convert Word documents to text, HTML, or XML formats.
Format	Select the format of the source documents, for example, tab-delimited or HTML. The wizard suggest formats that seem appropriate for the content type. If you selected a document processor, the format is that of the processor output, rather than the original source format.

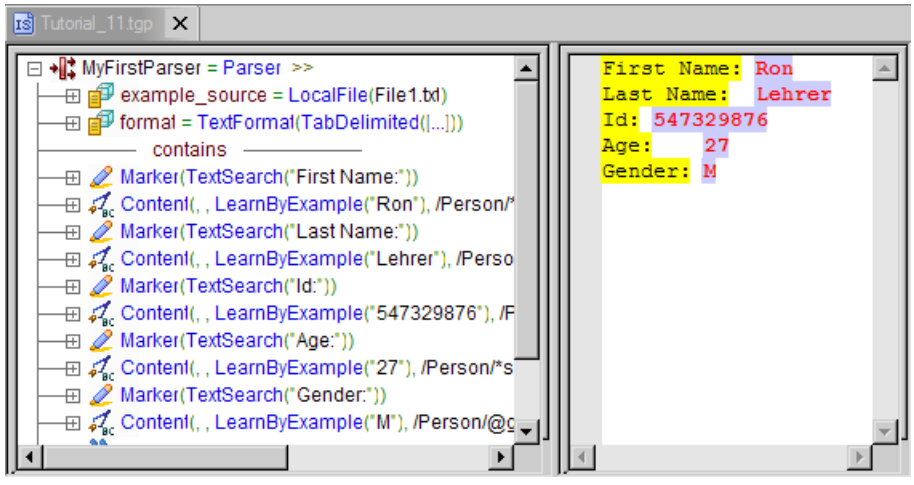
To complete the parser configuration:

1. Display the parser in an IntelliScript editor. You can do this by double-clicking the parser in the Component view or by double-clicking the TGP script file in the Conversion Agent Explorer.

   Figure 3-1. IntelliScript Editor



2. Under the `contains` line, add a sequence of nested anchors and actions.



3. Run and test the parser and modify the IntelliScript as required.

   For more information, see “Running a Parser” on page 12.

Creating a Parser by Editing the IntelliScript

Instead of using the New Parser wizard, you can create a parser by editing the IntelliScript directly. The result is identical to a parser created by using the wizard.

To create a parser in the IntelliScript:

1. At the top level of the IntelliScript, select the three dots (. . .) symbol. Press Enter and type a name for the parser.
2. To the right of the name, press Enter. Select a `Parser` component from the list.
3. Expand the tree under the `Parser` component. Assign its properties such as the `example_source` and the `format`.
4. If necessary, add an XSD schema defining the XML syntax of the parser output.

   For more information, see “Data Holders” on page 51.
5. Under the `contains` line, add a sequence of nested anchors and actions.

   For more information, see “Anchors” on page 67 and “Actions” on page 133.
6. Run and test the parser and modify the IntelliScript as required.

   For more information, see “Running a Parser” on page 12.

Running a Parser

To run a parser in Conversion Agent Studio, follow this procedure. For more information, see “Running and Testing Projects” on page 221.

To run a parser:

1. In the IntelliScript editor or in the Component view, right-click the parser and click Set as Startup Component.

   Alternatively, click Run > Run and set the startup component in the dialog box.

2. Click Run > Run MyParser, where MyParser is the name of the parser that you have set as the startup component.

3. After a few seconds, the Studio displays the Events view. Examine it for any failures or warnings.

4. To display the parsing results, double-click the file `Results\output.xml` in the Conversion Agent Explorer view.

Platform-Independent Parsers

Conversion Agent runs on both Microsoft Windows and UNIX-type systems. Most parser features run equally well on both platforms.

There are a few exceptions to this rule. If you plan to run a parser on both Windows and UNIX, here are a few tips that can help ensure platform independence.

Document Processors

Use document processors that do not have platform-specific system requirements. For more information, see “Document Processors” on page 21.

For example, use the `ExcelToXml` processor instead of `ExcelToHTML`. The former is platform-independent. The latter requires that Microsoft Excel be installed on the computer.

Custom Components

Conversion Agent supports custom document processors, transformers, and actions. Use platform-independent versions of the custom components, such as:

- `ExternalJavaPreProcessor`, programmed in Java
- `ExternalPreProcessor` or `ExternalTransformer`, programmed in C++ and compiled for both Windows and UNIX

Do not use `ExternalCOMPreProcessor` and `ExternalCOMAction`, which are supported only on Windows.

For more information about external components, see the *Conversion Agent Engine Developer Guide*.

Newline Markers

Avoid defining `Marker` anchors that search for a newline character followed by a carriage return character (`\n\r`). This combination is commonly used in Windows but often not in UNIX.

Instead, configure a `Marker` with the built-in `NewlineSearch` component, which searches for both the `\n\r` sequence and the `\n` or `\r` character alone.

Encoding

Confirm that the input, output, and working encoding are supported on the platforms. For more information, see “Project Properties” on page 213.

File Paths

Use relative, as opposed to absolute, file paths. Remember that file paths on UNIX are case-sensitive.

Parser Component Reference

The main `Parser` component is documented in this section. For subcomponents such as input ports, see the following sections of this chapter.

Parser

A `Parser` is a component that converts a source document to XML.

A `Parser` contains many nested components. Directly under the `contains` line of the `Parser`, you can nest anchors and actions. Under various `Parser` properties, you can assign components such as formats, delimiters, document processors, and transformers. For detailed information about all these components, see the following chapters of this book.

Example

The following is an example of a parser that processes tab-delimited text documents.

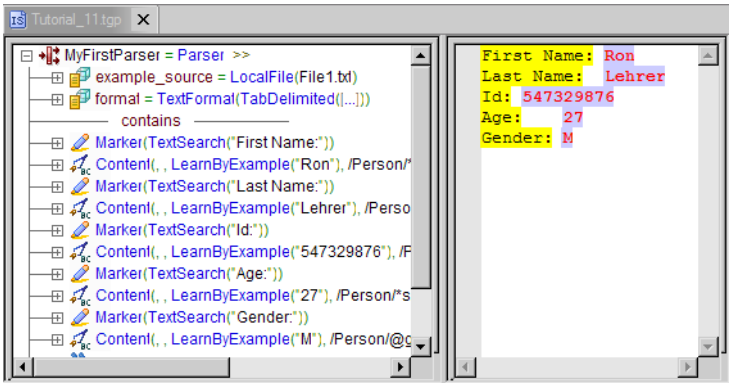


Table 3-2. Basic Properties

Property	Description
`example_source`	An example source document that you use to configure the parser operation. The document should be representative of the source documents that the parser will process. The value of the property is an input port such as `LocalFile` or `Text`. For more information, see “Ports” on page 15. Nested within this property, you can assign a preprocessor that converts the source documents to a format that the parser can accept. For more information, see “Document Processors” on page 21. To view the example source, right-click the parser and click Open Example Source.
`format`	Specifies the format of the source document, such as: - Whether the document contains text, HTML, or binary code - The delimiters that separate data fields in the document - Transformers that the parser should apply by default to all `Content` anchors For more information, see “Formats” on page 39.

Table 3-3. Advanced Properties

Property	Description
reject_recurring_pages	If selected, the parser does not parse the same page twice in the same execution. This is useful, for example, if a parser is following the links on a web site, and you want to prevent it from parsing duplicate links to the same page. The ResetVisitedPages action resets the history list and allows a parser to process a page again, even if reject_recurring_pages is selected.
no_initial_phase	If selected, the parser runs without an initial phase. Components that are configured to run in the initial phase run in the main phase, instead.
sources_to_extract	A specification of the source documents that the parser should process. The value of the property is an input port. For more information, see “Ports” on page 15. If you assign sources_to_extract and you run the parser in Conversion Agent Studio, the parser processes the specified documents. If you leave sources_to_extract blank, the parser processes the example_source.
serialization_mode	This property specifies how the Studio should process portions of the example source that the parser does not output to XML, when you create a serializer from a parser. For more information, see “Controlling How the Create Serializer Command Works” on page 164. The possible values of the serialization_mode are: Full. The Create Serializer command copies the non-XML text to the serializer configuration. Outline. The Create Serializer command copies only the delimiters of the non-XML text to the serializer configuration. Under the Outline option, you can select the use_markers option. This causes the Create Serializer command to copy the content of the Marker anchors but only the delimiters of other non-XML text.
name	A name that you assign to the parser. The name is displayed in the event log.
remark	A comment describing the parser.
example_values	This property contains simulated values that another transformation might pass to the parser. The property is useful when designing a parser that is to be activated by another parser. Conversion Agent uses the property only when it learns the example source. It ignores the property when it parses a source document. In the nested ExampleValue components, specify the data holders that the main parser passes to this parser and their simulated values.
source target	These properties are useful in situations where the parser must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.
on_fail	If the parser fails, writes an entry in the user log. For more information, see “Failure Handling” on page 226.

Online Samples

In the samples and tutorials folders, you can find many examples of parsers.

CHAPTER 4

Ports

This chapter includes the following topics:

Overview

A port is a component that specifies an input or output of a transformation, such as a source document or an output document. For example, in a `Parser` component, the values of the `example_source` and `sources_to_extract` properties are input ports.

A port can define a document that is stored on the local computer, on a network, or in a string. Ordinarily, a port defines a default input or output, for example, a file that is used to develop and test the transformation. At runtime, the application that activates the transformation specifies the actual input or output document, overriding the defaults. For example, if you use the Conversion Agent API to activate a parser, the API application specifies the input document that the parser should process, overriding the `example_source`.

In many transformations, the ports are implicitly defined. For example, the default output of a parser is a file called `output.xml`. You do not need to define a port that references the file `output.xml`.

By default, each transformation has a singe input and a single output. Optionally, you can configure transformations that have multiple input and output ports.

Port Quick Reference

Port Component	Description
`AdditionalInputPort`	Defines an additional, non-default input of a transformation.
`AdditionalOutputPort`	Defines an additional, non-default output of a transformation.
`DocList`	Defines a list of documents.
`FileSearch`	Defines a search criterion for a file.
`InputPort`	Defines a document by referencing an additional input port.
`LocalFile`	Defines a file path.
`Text`	Defines a text string that is the input of a transformation.
`URL`	Defines a URL where a document is located.

Port Component Reference

AdditionalInputPort

This component defines an additional input port.

Define the component at the global level of the IntelliScript, and assign it a name. In locations such as the `example_source` of a parser, you can insert an `InputPort` component that references the name.

Example

Suppose you have two text files:

- `IdsAndSalaries.txt` is a table of employee IDs and salaries.
- `IdsAndNames.txt` is a table of employee IDs and names.

You want to parse these files jointly, generating an XML output file containing the employee names and salaries. You can configure the transformation in the following way:

- The main parser, called `EmployeeParser`, processes `IdsAndSalaries.txt`.
- The main parser activates a secondary parser, called `IdsToNamesParser`, which processes `IdsAndNames.txt` and stores the result in an XML table.
- The main parser uses a `LookupTransformer` to convert the IDs to names. The lookup table is the output of the secondary parser.

The following IntelliScript illustrates this configuration. The secondary parser references an `AdditionalInputPort` that retrieves the file `IdsAndNames.txt`.

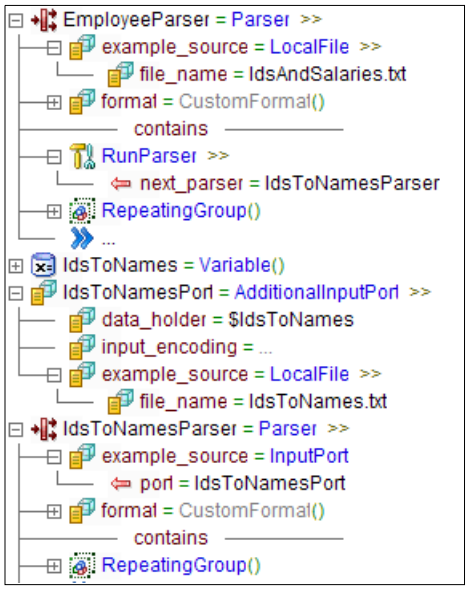


Table 4-1. Basic Properties

Property	Description
`data_holder`	A data holder where the system stores the content of the input when the transformation begins.
`input_encoding`	Encoding of the input, such as a code page. For more information about encoding support, see “Encoding Properties” on page 214.
`example_source`	The default location of the additional input. The value is an input-port component such as `LocalFile`, `URL`, or `Text`.

Table 4-2. Advanced Properties

Property	Description
`pre_processor`	The name of a preprocessor that the transformation should apply to the files. For more information, see “Document Processors” on page 21.
`disabled`	If selected, the transformation ignores the component.

AdditionalOutputPort

This component defines an additional output port.

Define the component at the global level of the IntelliScript, and assign it a name. In locations such as a `WriteValue` action, you can insert an `OutputPort` component that references the name.

Example

A transformation writes its main output to a file called `output.xml`. The transformation uses an `AdditionalOutputPort` to define a secondary output file. A `WriteValue` action generates output to the secondary file.

Table 4-3. Basic Properties

Property	Description
`output_encoding`	The encoding of the additional output, such as a code page. For more information about encoding support, see “Encoding Properties” on page 214.

Table 4-4. Advanced Properties

Property	Description
disabled	If selected, the transformation ignores the component.
file_extension	The file extension for the output file. The default is .xml.
other_properties	Enables you to configure properties of the output such as an XML header. For more information about the output properties, see “Output Control Properties” on page 218 and “XML Generation Properties” on page 219.

DocList

A document list. The component allows you to specify multiple source documents that a transformation should process.

Within the component, you can nest multiple input ports such as FileSearch, LocalFile, or Text, each of which specifies a single document.

FileSearch

The criteria for a file search.

You can use this input port, for example, in the sources_to_extract property of a Parser. It lets you specify source documents using wildcards.

Table 4-5. Basic Properties

Property	Description
directory	The folder to be searched.
wildcard	The search criterion. You may use * as a wildcard character. For example, *.txt finds all text files. The default is *.*, which finds all files in the directory.

Table 4-6. Advanced Properties

Property	Description
recursive	If selected, the search includes subfolders of the specified directory.
pre_processor	The name of a preprocessor that the transformation should apply to the files. For more information, see “Document Processors” on page 21.

InputPort

This component specifies that the input should be taken from a named port that is defined by using an AdditionalInputPort component.

Table 4-7. Basic Properties

Property	Description
input	The name of the AdditionalInputPort component defining the input.

LocalFile

A file on the local computer.

Table 4-8. Basic Properties

Property	Description
file_name	Browse to the file.

Table 4-9. Advanced Properties

Property	Description
simulated_url	A URL that Conversion Agent should assign to the file. This property instructs Conversion Agent to treat the file as if it were located on a web server. If the file contains relative links, Conversion Agent resolves the links relative to the URL. The host-name portion of the URL is not case sensitive. Internally, Conversion Agent processes HTTP host names as lower case.
pre_processor	The name of a preprocessor that the transformation should apply to the files. For more information, see “Document Processors” on page 21.

OutputPort

This component specifies that the input should be taken from a named port that is defined by using an `AdditionalOutputPort` component. You can use an **`OutputPort`** component to define the output of a `WriteValue` action.

Table 4-10. Basic Properties

Property	Description
port	The name of the `AdditionalOutputPort`.

Text

A text string.

Table 4-11. Basic Properties

Property	Description
quote	The text string.

Table 4-12. Advanced Properties

Property	Description
simulated_url	A URL that Conversion Agent should assign to the string. This property instructs Conversion Agent to treat the string as if it were a file located on a web server. If the string contains relative links, Conversion Agent resolves the links relative to the URL.
pre_processor	The name of a preprocessor that the transformation should apply to the files. For more information, see “Document Processors” on page 21.
size	A static size for the text buffer. This property is typically used when working with binary sources. The default is -1, which means that the buffer is dynamically sized.

URL

The URL of a document that is available on a web server.

Table 4-13. Basic Properties

Property	Description
stable_url	The URL address, for example, `http://www.example.com/index.html`. The host name, `www.example.com`, is not case sensitive. Internally, Conversion Agent processes HTTP host names as lower case.

Table 4-14. Advanced Properties

Property	Description
`post_data`	Data that the transformation should post to the URL. To determine the correct format of the data string, you can use the technique described in the `SubmitForm` action.
`retries`	If the transformation cannot access the URL on the first attempt, the number of retries that it performs before reporting a failure. Default = 0.
`seconds_to_wait`	The number of seconds to wait between retries. Default = 60.
`pre_processor`	The name of a preprocessor that the transformation should apply to the files. For more information, see “Document Processors” on page 21.

CHAPTER 5

Document Processors

This chapter includes the following topics:

Overview

Document processors are components that convert the format of a complete document to another format that is desired for processing.

You can use a document processor as a pre-processor that converts the format of a source document prior to a transformation. For example, if the source document of a parser is in the PDF format, you might apply the `PdfToTxt_4` processor. This converts the source document to text, which is much easier to parse than the binary PDF format.

Do not confuse document processors with format preprocessors. For more information about format preprocessors, see “Formats” on page 39.

Installation

The document processors are supplied in an optional setup component. If you plan to use the processors, select the option to install the Processors when you run the Conversion Agent setup.

Defining Document Processors

You can define any document processor to pre-process the source document of a parser. If the processor output is XML, you can use it to pre-process the source document of a parser, serializer, or mapper.

To define a document processor:

1. Assign the `example_source` property of the transformation.The value of the `example_source` is an input port, such as `LocalFile` or `Text`.

   For more information, see “Ports” on page 15.

2. Assign the `pre_processor` property of the input port.

   Conversion Agent applies the processor that you define under **`example_source`** to all sources on which you run the transformation.

3. If you use the processor in a serializer or mapper project, add the XSD schema of the processor output to the project.

   In a parser project, you do not need to add the schema.

Note: You can also define a pre-processor in the `sources_to_extract` property of a parser. The processor that you define there applies only to the source documents that you define in `sources_to_extract`, and not to any other document that the parser processes.

Display of Document Processor Output

If you assign a document processor to the example source, the example pane of the IntelliScript editor displays the processor output.

Document Processor Quick Reference

Document Processor	Description
AFPToXML	Converts the IBM Advanced Function Presentation print-stream format to XML.
ExcelToDataXml	Converts Microsoft Excel data to XML, without preserving Excel formulas, formatting, or code.
ExcelToHtml	Converts Microsoft Excel documents to HTML.
ExcelToTextML	Converts Microsoft Excel files to the TextML XML schema.
ExcelToTxt	Converts Microsoft Excel documents to plain text.
ExcelToXml	Converts Microsoft Excel documents to XML, while preserving the Excel formulas, formatting, and optionally macro code.
ExpandFrameSet	Opens an HTML frameset, letting a parser run on the content of the frames.
ExternalCOMPreProcessor	Runs a custom document processor, implemented as a COM DLL.
ExternalJavaPreProcessor	Runs a custom document processor, implemented in Java.
ExternalPreProcessor	Runs a custom document processor, implemented as a C++ DLL.
PdfFormToXml_1_00	Converts PDF forms to XML.
PdfToTxt_4	Converts PDF documents to plain text.
PowerpointToHtml	Converts Microsoft PowerPoint documents to HTML.
PowerpointToTextML	Converts Microsoft PowerPoint presentations to the TextML XML schema.
ProcessByTransformers	Runs transformers as document processors.
ProcessorPipeline	Runs a sequence of document processors on a single document.
RtfToTextML	Converts RTF files to the TextML XML schema.
WordPerfectToTextML	Converts Corel WordPerfect documents to the TextML XML schema.
WordToHtml	Converts Microsoft Word documents to HTML.
WordToRtf	Converts Microsoft Word documents to RTF.
WordToTextML	Converts Microsoft Word files to the TextML XML schema.
WordToTxt	Converts Microsoft Word documents to plain text.
WordToXml	Converts Microsoft Word documents to XML.
XmlToExcel	Converts XML documents to Microsoft Excel.

Document Processor Component Reference

This section describes the document processor components that are available in Conversion Agent.

AFPToXML

This document processor converts the IBM Advanced Function Presentation print-stream format to XML.

The processor output is in the UTF-8 encoding. If a transformation receives input from the processor, you must set the input encoding of the project to UTF-8. For more information, see “Encoding Properties” on page 214.

ExcelToDataXml

This document processor converts Microsoft Excel documents to XML.

The XML contains the data and the results of formulas that existed in the original Excel document. It does not preserve the formulas themselves, formatting information, or macro code. In cases where the latter information is required, use `ExcelToXml` rather than `ExcelToDataXml`.

The XML representation conforms to a subset of the `ExcelToXml.xsd` schema, which you can find in the `doc` subdirectory of the Conversion Agent installation directory.

The processor output is in the UTF-8 encoding. If a transformation receives input from the processor, you must set the input encoding of the project to UTF-8. For more information, see "Encoding Properties" on page 214.

The processor support Excel version 97 and later. It accesses its input directly, not via Excel. You do not need to install Excel on the computer. The processor supports both the XLS format and the XLSX format introduced in Excel 2007.

The processor is implemented in Java. If you experience any difficulty using the processor, confirm that you have configured the Java Runtime Environment (JRE) correctly. For more information about the JRE, see the *Conversion Agent Administrator Guide*.

Table 5-1. Basic Properties

Property	Description
`Display_raw_data_when_different`	Formatted data in the source file may appear differently from the raw data. For example, the raw data 1 appears as 1.00 if its cell is formatted as Number with two decimal places. If you enable this property, the processor includes both the raw data and the formatted data in its output, if they differ. If you disable the property, the processor includes only the formatted data.

ExcelToHtml

This document processor converts Microsoft Excel documents to HTML.

The processor uses the Excel save-as-HTML feature to perform the conversion. It operates only on a Microsoft Windows platform where Excel version 97 or higher is installed. Due to Excel limitations, the processor does not support multithreading.

ExcelToTextML

This document processor converts Microsoft Excel files to the TextML XML schema. For more information, see "TextML XML Schema" on page 31.

The processor support Excel version 97 and higher. It accesses its input directly, not via Excel. You do not need to install Excel on the computer.

The processor is implemented in Java. If you experience any difficulty using the processor, confirm that you have configured the Java Runtime Environment (JRE) correctly. For more information the JRE, see the *Conversion Agent Administrator Guide*.

ExcelToTxt

This document processor converts Microsoft Excel documents to plain text.

The processor uses the Excel save-as-text feature to perform the conversion. It operates only on a Microsoft Windows platform where Excel version 97 or higher is installed. Due to Excel limitations, the processor does not support multithreading.

ExcelToXml

This document processor converts Microsoft Excel documents to XML.

The XML preserves the data, formulas, formatting, and optionally the macro code that existed in the original Excel document. If only the data is required, consider using the `ExcelToDataXml` processor, which offers smaller output and better performance.

The XML representation conforms to the `ExcelToXml.xsd` schema, which is in the `doc` subdirectory of the Conversion Agent installation directory.

The processor output is in the UTF-8 encoding. If a transformation receives input from the processor, you must set the input encoding of the project to UTF-8. For more information, see "Encoding Properties" on page 214.

The processor support Excel version 97 and later. It accesses its input directly, not via Excel. You do not need to install Excel on the computer.

The processor is implemented in Java. If you experience any difficulty using the processor, confirm that you have configured the Java Runtime Environment (JRE) correctly. For more information about the JRE, see the *Conversion Agent Administrator Guide*.

Table 5-2. Basic Properties

Property	Description
`include_sheets`	Defines the sheets of the Excel workbook to include in the XML. In the XML output, each sheet is represented by a `<sheet>` element. In the list under this property, you may enter any of the following values: - `All`: includes all sheets - The sheet names - Data holders containing the sheet names If you list a sheet that doesn't exist in the workbook, the processor generates a `<sheet>` element containing a warning message. The other sheets are processed normally.
`include_empty_cells`	Deselect this property to omit empty cells from the XML.
`include_macro_information`	Select this property to include Excel macro code in the XML.

ExpandFrameSet

This document processor opens all the frames of an HTML document.

This processor is appropriate if the source document of a parser is an HTML frameset. The parser runs on the content of all the frames.

ExternalCOMPreProcessor

This component allows you to run a custom document processor.

Because the component uses the Microsoft COM architecture to activate the processor, it runs only on Microsoft Windows platforms.

To create a custom COM processor:

1. Program an ActiveX DLL implementing the following function:

   ```
   function pre_process(ByVal in_file As String) As String
   ```

   The `in_file` parameter is the content of the source document. The function returns the processed text.
2. Register the DLL on the Conversion Agent computer.
3. Define an `ExternalCOMPreProcessor` that references the ProgID of the DLL.

4. Optionally, add the `ExternalCOMPreProcessor` to the component list that Conversion Agent Studio displays.

Table 5-3. Basic Properties

Property	Description
`ProgID`	The `ProgID` of the class.

ExternalJavaPreProcessor

This component allows you to run a custom document processor that is implemented in Java.

Note: This component is supported for backwards compatibility with existing custom processors. For more information about custom processors and other external components, see the *Conversion Agent Engine Developer Guide*.

To create a custom Java processor:

1. Create a new Java project and package, for example, named `MyJavaPreprocessor`.
2. Create a class, for example, named `JavaDemoPreprocessor`.
3. In the class, define a method having the following syntax. The method can have any name.

   ```
   public static String main(String input_file, String output_file)
   ```

   The `input_file` parameter is the path of the source document on which the processor should operate. The `output_file` parameter is the path of a temporary file where the processor should write its output.

   The function returns an extension that Conversion Agent appends to the name of the temporary file. For example, if the output of the processor is XML, the function can return the string `"xml"`.
4. Create a jar file containing the class.
5. Store the jar file in the `externLibs\user` subfolder of your Conversion Agent installation folder.
6. Define an `ExternalJavaPreProcessor` that references the class and method.
7. Optionally, add the `ExternalJavaPreProcessor` to the component list that Conversion Agent Studio displays.

If you experience any difficulty using the processor, confirm that you have configured the Java Runtime Environment (JRE) correctly. For more information about the JRE, see the *Conversion Agent Administrator Guide*.

Example

The following is the source code of a processor that repairs numeric values by removing commas between the numbers.

```
package MyJavaPreprocessor;
import java.io.*;
public class JavaDemoPreprocessor {
  private static final int MAX_SIZE = 4096;
  public static String main(String input_file, String output_file){
    try {
      FileInputStream in = new FileInputStream(input_file);
      FileOutputStream out = new FileOutputStream(output_file);
      int bytes_read=0;
      while(bytes_read != -1)
      {
        byte [] in_buf = new byte[MAX_SIZE];
        byte [] out_buf= new byte[MAX_SIZE];
        bytes_read = in.read(in_buf);
        int j = 0;
        for (int i=1;i<bytes_read;i++)
        {
```

```
            if (in_buf[i] == ',')
            {
              if (Character.isDigit((char)in_buf[i-1]) &&
                  Character.isDigit((char)in_buf[i+1]))
              {
                    // Do Nothing
              }
              else
                out_buf[j++] = in_buf[i];
            }else
              out_buf[j++] = in_buf[i];
          }
          out.write(out_buf, 0, j);
          in.close();
          out.close();
        }
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      } catch (IOException e) {
      }
      //return output file extension type
      return "txt";
    }
  }
```

Table 5-4. Basic Properties

Property	Description
`jclass`	The path of the Java class, for example, `MyJavaPreprocessor/JavaDemoPreprocessor`.
`jmethod`	The method to run, for example, `main`.

Online Sample

For an online sample of the Java code, similar to the above example, see the following file in the Conversion Agent installation folder:

```
samples\SDK\ExternalPreprocessor\External_JavaPreprocessor.java
```

ExternalPreProcessor

This component allows you to run a custom document processor that is implemented as a C++ DLL.

Note: This component is supported for backwards compatibility with existing custom processors. For more information about custom processors and other external components, see the *Conversion Agent Engine Developer Guide*.

The following instructions are for the Microsoft Visual C++ compiler, running on a Microsoft Windows platform.

To create a custom C++ processor:

1. Copy the following file from the Conversion Agent installation folder:

   ```
   samples\SDK\ExternalPreprocessor\External_Preprocessor.cpp
   ```

2. Using the Visual C++ compiler, create a Win32 dynamic-link library project, and insert the C++ file into the project.

3. Edit the following function:

   ```
   declspec(dllexport) bool process_buffer(istream& in, ostream& out)
   ```

   In the sample implementation, the function repairs numeric values, removing commas between the values. Replace the sample code with your implementation.

4. Compile the DLL.

5. Store the DLL in the `externLibs\user` subfolder of the Conversion Agent installation folder.
6. Define an `ExternalPreProcessor` that references the DLL.
7. Optionally, add the `ExternalJavaPreProcessor` to the component list that Conversion Agent Studio displays. For more information about customizing the component list, see *Using Conversion Agent Studio in Eclipse*.

Table 5-5. Basic Properties

Property	Description
`import_dll`	Browse to the DLL in the `ExternLib\Users` folder.

PdfFormToXml_1_00

This document processor converts PDF forms to XML. The processor supports forms that conform to the Adobe AcroForms standard, for example, forms that were created by Foxit or Open Office.

Generating an XML Output Schema

You can use the processor in a parser, mapper, or serializer project.

In a mapper or serializer project, you must provide an XSD schema of the processor XML output. The schema exposes the internal structure of the PDF form and differs for each kind of form.

Conversion Agent Studio provides an import utility that auto-generates the schema. By default, the utility generates a mapper project that uses the processor. You can change the mapper to a serializer as required.

To generate a mapper or serializer project including a schema:

1. On the Studio menu, click File > New > Project.
2. In the Conversion Agent category, select an Import Project and click Next.
3. Enter a name for the project.
4. As the import type, select PdfForm.
5. Browse to a sample PDF form file of the kind that you plan to process.
6. Verify the path of the PDF form file and click Finish.

   The Studio creates a mapper project having the following characteristics:

   - The project is configured with a `PdfFormToXml_1_00` processor.
   - The output of the processor is the input of the mapper.
   - The example source of the mapper is the sample PDF form that you selected.
   - The project contains a schema for the XML output of the processor.
7. If you plan to map the form data to another XML schema, configure the mapper.
8. If you plan to serialize the form data, do one of the following:

   - Edit the IntelliScript of the mapper project, changing the startup component from a mapper to a serializer using the same example source.
   - Create an independent serializer project using the same example source. Add the schema file that you generated in the mapper project to the serializer project.

   You can then use the `PdfFormToXml_1_00` processor with the serializer.

Tip: To verify that PDF source documents conform to the schema, configure the mapper or serializer with the property `validate_source_documents = Strict`.

PdfToTxt_4

This document processor converts PDF files to text or XML.

The processor output is in the UTF-8 encoding. Since the parser receives input from the processor, you must set the input encoding of the project to UTF-8. For more information, see “Encoding Properties” on page 214.

The processor provides a graphical table configuration editor. You can use the editor to improve the processing by defining table locations and column widths. For more information, see “PdfToTxt_4 Table Configuration Editor” on page 32.

`PdfToTxt_4` does not require Adobe Acrobat or other PDF software to convert PDF files.

By default, the processor generates text output. Optionally, if you use the graphical table editor, you can select XML output. The XML conforms to the `PDF4.xsd` schema, stored in the `doc` subdirectory of the Conversion Agent installation directory.

Table 5-6. Basic Properties

Property	Description
`PdfLayout`	Defines the PDF table layout. Double-click the value of this property to open the table configuration editor.

Note: If you open a project that was created in a previous Conversion Agent version, you might observe that it uses an older `PdfToTxt` processor such as `PdfToTxt_3_02`. The older versions are supplied for backwards compatibility. In new projects, use `PdfToTxt_4`.

PowerpointToHtml

This document processor converts Microsoft PowerPoint documents to HTML.

The processor uses the PowerPoint save-as-HTML feature to perform the conversion. It operates only on a Microsoft Windows platform where PowerPoint version 97 or higher is installed. Due to PowerPoint limitations, the processor does not support multithreading.

PowerpointToTextML

This document processor converts Microsoft PowerPoint (`*.PPT`) presentations to the TextML XML schema. For more information, see “TextML XML Schema” on page 31.

The processor supports PowerPoint version 97 and higher. It accesses its input directly, not via PowerPoint. You do not need to install PowerPoint on the computer.

The processor is implemented in Java. If you experience any difficulty using the processor, confirm that you have configured the Java Runtime Environment (JRE) correctly. For more information about the JRE, see the *Conversion Agent Administrator Guide*.

ProcessByTransformers

This component allows you to run transformers as document processors.

The component runs a transformer or a sequence of transformers on the entire document, as opposed to the normal transformer usage, which is to run on the text retrieved by an anchor. A transformation can then run on the output of the transformers.

Conversion Agent offers a large number of transformers. Hence, the `ProcessByTransformers` component greatly expands the set of processing operations that you can apply to a document. For more information, see “Transformers” on page 101.

Table 5-7. Basic Properties

Property	Description
`transformers`	The sequence of transformers that the component should run.

ProcessorPipeline

This component allows you to run a sequence of document processors on a document. A transformation can run on the output of the sequence.

Within this component, enter the sequence of processors.

RtfToTextML

This document processor converts RTF files to the TextML XML schema. For more information, see “TextML XML Schema” on page 31.

The processor output is in the UTF-8 encoding. If a transformation receives input from the processor, you must set the input encoding of the project to UTF-8. For more information, see “Encoding Properties” on page 214.

WordPerfectToTextML

This document processor converts Corel WordPerfect documents to the TextML XML schema. For more information, see “TextML XML Schema” on page 31.

The processor output is in the UTF-8 encoding. If a transformation receives input from the processor, you must set the input encoding of the project to UTF-8. For more information, see “Encoding Properties” on page 214.

WordPerfect does not need to be installed on the computer.

WordToHtml

This document processor converts Microsoft Word documents to HTML.

The processor uses the Word save-as-HTML feature to perform the conversion. It operates only on a Microsoft Windows platform where Word version 97 or higher is installed. Due to Word limitations, the processor does not support multithreading.

WordToRtf

This document processor converts Microsoft Word documents to RTF.

The processor uses the Word save-as-RTF feature to perform the conversion. It operates only on a Microsoft Windows platform where Word version 97 or higher is installed. Due to Word limitations, the processor does not support multithreading.

WordToTextML

This document processor converts Microsoft Word files to the TextML XML schema. For more information, see “TextML XML Schema” on page 31.

The processor supports Word version 97 and higher. It accesses its input directly, not via Word. You do not need to install Word on the computer.

The processor is implemented in Java. If you experience any difficulty using the processor, confirm that you have configured the Java Runtime Environment (JRE) correctly. For more information about the JRE, see the *Conversion Agent Administrator Guide*.

WordToTxt

This document processor converts Microsoft Word documents to plain text.

The processor uses the Word save-as-text feature to perform the conversion. It operates only on a Microsoft Windows platform where Word version 97 or higher is installed. Due to Word limitations, the processor does not support multithreading.

The output is encoded according to the system code page.

WordToXml

This document processor converts Microsoft Word documents to XML.

The processor output is in the UTF-8 encoding. If a transformation receives input from the processor, you must set the input encoding of the project to UTF-8. For more information, see "Encoding Properties" on page 214.

The processor supports Word version 97 and higher. It accesses its input directly, not via Microsoft Word. You do not need to install Word on the computer.

The processor is implemented in Java. If you experience any difficulty using the processor, confirm that you have configured the Java Runtime Environment (JRE) correctly. For more information about the JRE, see the *Conversion Agent Administrator Guide*.

XmlToExcel

This document processor converts XML documents to Microsoft Excel format.

The processor operates on an XML representation of an Excel workbook. The XML representation must be in the UTF-8 encoding and it must conform to the `ExcelToXml.xsd` schema. You can find the schema in the `doc` subdirectory of the Conversion Agent installation directory. The schema file is provided for your information. You can use the processor without adding the schema to your project.

The processor reverses the operation of `ExcelToXml`. For example, you can use `ExcelToXml` to convert an Excel workbook to XML. You can then alter some of the XML data and use `XmlToExcel` to convert the data back to an Excel workbook.

The processor support Excel version 97 and higher. It writes its output directly, not via Microsoft Excel. You do not need to install Excel on the computer.

The processor is implemented in Java. If you experience any difficulty using the processor, confirm that you have configured the Java Runtime Environment (JRE) correctly. For more information about the JRE, see the *Conversion Agent Administrator Guide*.

TextML XML Schema

Some of the document processors convert documents to an XML vocabulary called TextML. This is a simple XML vocabulary for saving document content without layout.

The TextML schema, `textML.xsd`, is available in the `doc` subfolder of the Conversion Agent installation folder.

The following is a sample TextML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <docinfo>
    <title>TextML Sample</title>
    <author>Tex Tomiller</author>
    <company>Acme Gizmos, Inc.</company>
    <modified>2004-03-14T14:39:00</modified>
    <created>2004-03-12T09:15:00</created>
    <last_author>Tex Tomiller</last_author>
    <word_count>16</word_count>
    <char_count>105</char_count>
    <version>2</version>
  </docinfo>
  <docbody>
    <p>This is a sample of the TextML XML vocabulary.</p>
    <p>TextML saves document content without layout information.<p>
  </docbody>
</document>
```

PdfToTxt_4 Table Configuration Editor

The `PdfToTxt_4` document processor converts PDF files to text or XML. The processor can handle all types of text found in PDFs, whether in paragraph format or in tables. For more information about the processor, see “PdfToTxt_4” on page 29.

Due to the internal PDF structure, table processing can raise issues such as incorrect column alignment, incorrect word wrapping, incorrect spacing between lines, and overflow from one cell to another. `PdfToTxt_4` helps overcome these issues by providing a graphical table configuration editor. This section explains how to use the editor.

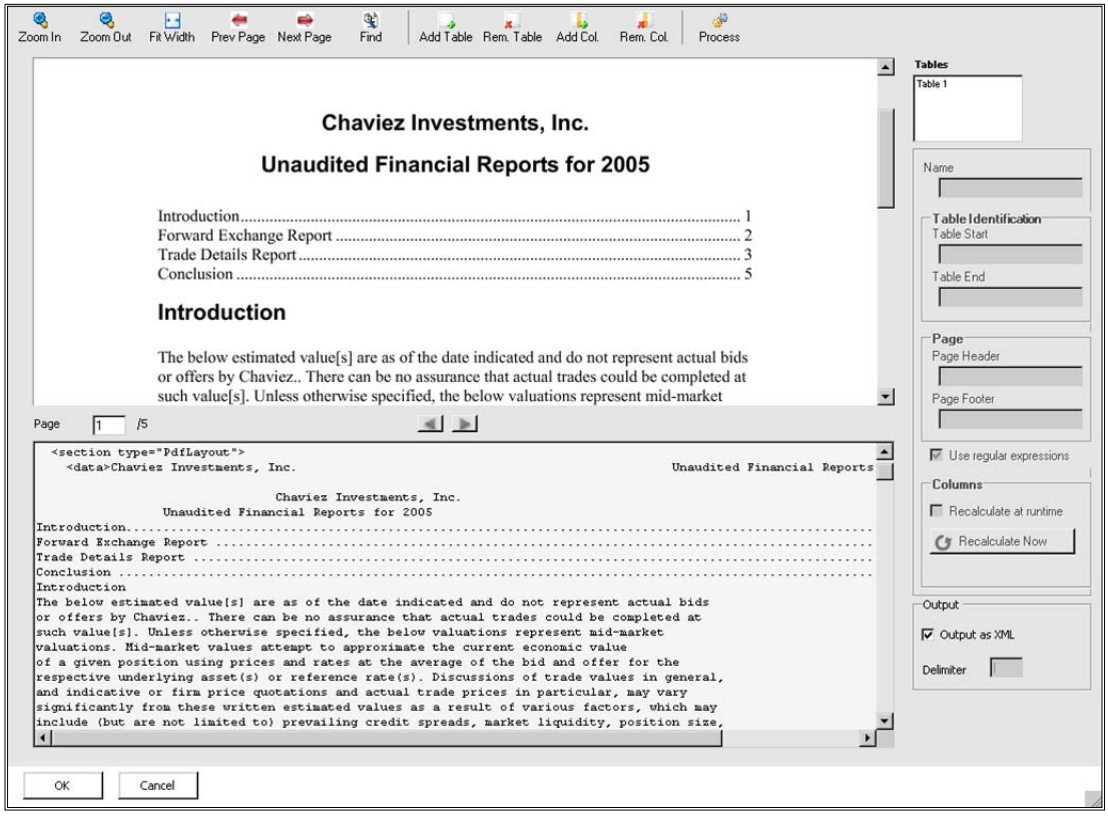
Note: If you use the processor with PDF documents that do not contain tables, or if the default table processing is sufficient, you do not need to use the graphical table editor. In that case, you can skip the instructions in this section.

To configure the table processing:

1. In the IntelliScript, configure an `example_source` that is a PDF file, and insert a `PdfToTxt_4` processor.
2. Nested within the `PdfToTxt_4` processor, double-click the `value` property.

   The Studio opens the PDF table configuration editor. The upper portion of the editor displays the input PDF document. The lower portion displays the `PdfToTxt_4` output.

Table editing commands are available on the toolbar at the top of the screen. You can right-click to display an editing menu.



3. Browse to a table in the PDF document and click Add Table.

   The system displays the name of the table in the Tables field and in the Name field.

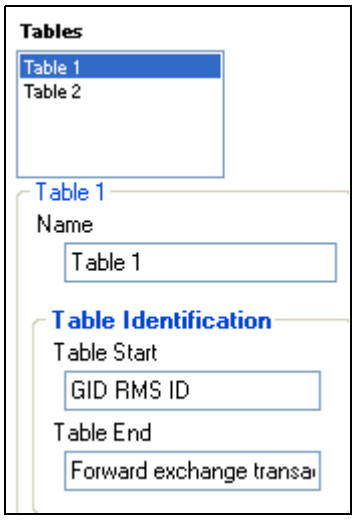
4. Define the start of the table by entering a regular expression in the Table Start field. The expression must define the upper left corner of the table.

   Tip: Use the headings of the first two columns as the regular expression. Add more column headings as needed to make Table Start unique. Separate the headings by a single space character, even if the columns are widely separated. You can use the `^` and `$` symbols to force the regular expression to match the start and end of a line. For more information about regular expression syntax, see “RegularExpression” on page 121. To use regular expressions, you must select the Use Regular Expressions checkbox.

   For example, if the first two column headings are `GID` and `RMS ID`, enter the value `GMS RMS ID` in the Table Start field. If the string `GMS RMS ID` might occur elsewhere in the document, enter `^GMS RMS IS`.

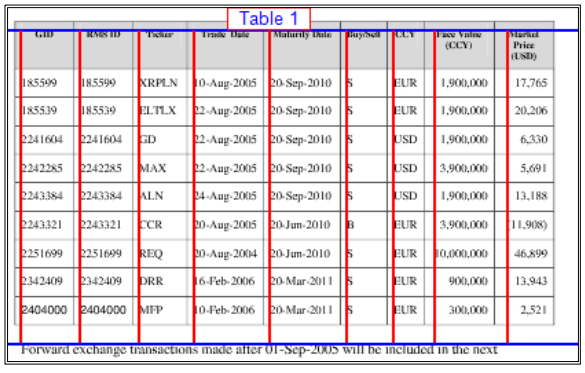
5. Define the end of the table by entering a regular expression in the Table End field. The expression must define the text that immediately follows the table. For example, the first few words of body text

immediately following the table might be a good definition for Table End. The value of Table End must appear in the body of the document, not in a page footer.



6. Click Process.

   The editor displays the table configuration that `PdfToTxt_4` detects. The top and bottom of the table appear as horizontal blue lines. The default column borders appear as vertical red lines.



7. Edit the column borders by dragging them left or right as required.

   To add a column border, click Add Column, and position it in the table. To delete a column border, click Remove Column and then click the border to remove.

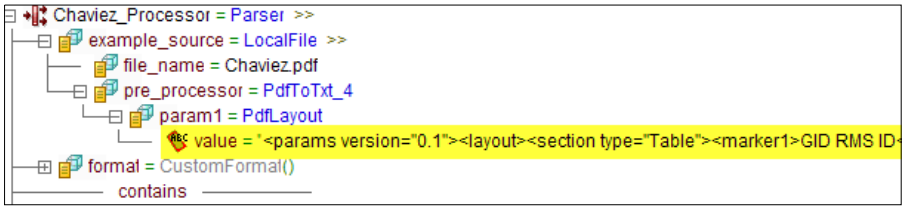
   Note: If the table contains horizontally merged cells, `PdfToTxt_4` might truncate the entries.

8. Examine the output window to confirm that the table is converted properly. If not, correct the table definitions.

9. Repeat these steps for each table in the PDF document.

10. Click OK to return to the Studio.

    The `value` property of the `PdfToTxt_4` processor now contains an XML string that defines the table configuration. The example pane displays the `PdfToTxt_4` output. You can continue configuring the transformation in the IntelliScript.



Editor Options

The following table describes the controls and fields in the `PdfToTxt_4` table configuration editor.

Control or Field	Description
Zoom In	Make the PDF display larger.
Zoom Out	Make the PDF display smaller.
Fit Width	Display the PDF document according to the width of the window.
Prev Page	Go to the previous page.
Next Page	Go to the next page.
Find	Search for a string in the PDF.
Add Table	Add a table to the configuration.
Rem. Table	Remove a table from the configuration.
Add Column	Add a column border to the current table.
Rem. Column	Delete the currently selected column border.
Process	Apply the current table definitions. Click Process after every table and column-related action to apply that action.
Tables	A list of tables defined in the input PDF. You can select a table by clicking it.
Name	Name of the currently selected table.
Table Start	An expression defining the upper left corner of the table.
Table End	An expression defining the first text after the table.
Page Header	An expression defining the end of the page header. Use this option to exclude the header from the table processing.
Page Footer	An expression defining the end of the page footer. Use this option to exclude the footer from the table processing.
Use Regular Expressions	If selected, the processor interprets the Table Start, Table End, Page Header, and Page Footer as regular expressions and searches for matching text. If not selected the processor interprets these fields as literal text.
Recalculate at Runtime	If you select this option, `PdfToTxt_4` ignores the table configurations that you specified using the table configuration editor. This feature is useful if the tables in a PDF are simple enough for the `PdfToTxt_4` to process without special configuration. For example, suppose a simple PDF financial statement contains a table whose columns may vary slightly from month to month. Select the Recalculate at Runtime option to have `PdfToTxt_4` adjust the column widths at runtime.
Recalculate Now	If you have changed the table definition, for example by changing column borders or adding a Page Header or Page Footer, click Recalculate Now to update the table definition.
Page	Number of the PDF page that is currently displayed.
Output as XML	Generates the `PdfToTxt_4` output as XML instead of text.
Delimiter	Enter a character to use as the column separator in the text output. The default is a vertical bar (|).
OK	Click to save the table configuration and return to the Studio.
Cancel	Click to return to the Studio project without saving the table configuration.
Table Navigation Aid	The table navigation aid displays the number of times a table is found in the PDF document. An example of a navigation aid is `Table 'Table 1' found 2 times.` The arrows next to this information let you jump back and forth among the instances of the same table structure.

Sample PDF Conversion

This example illustrates the `PdfToTxt_4` table configuration procedure using a sample parser project and a sample PDF document.

The processor input is a small financial report in PDF format. The report contains some text and two tables. We will use the table configuration editor to ensure that the processor converts the tables correctly to text.

Configuring the First Table

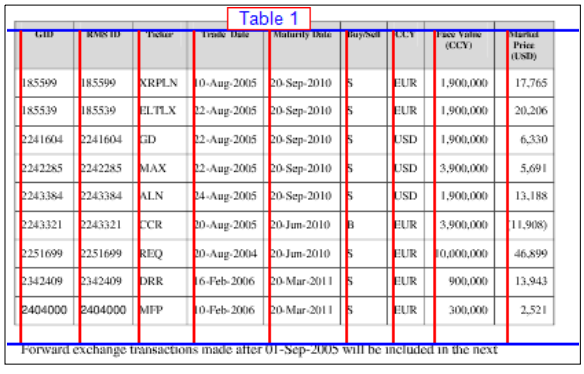
To start the procedure, we configure a parser project and assign the PDF document as the `example_source`. Double-click on the `value` property to open the table configuration editor.

In the PDF display, browse to the first table.

GID	RMS ID	Ticker	Trade Date	Maturity Date	Buy/Sell	CCY	Face Value (CCY)	Market Price (USD)
1855998	185599	XRPLN	10-Aug-2005	20-Sep-2010	S	EUR	1,900,000	17,765
1855391	185539	ELTLX	22-Aug-2005	20-Sep-2010	S	EUR	1,900,000	20,206
2241604	2241604	GD	22-Aug-2005	20-Sep-2010	S	USD	1,900,000	6,330
2242285	2242285	MAX	22-Aug-2005	20-Sep-2010	S	USD	3,900,000	5,691
2243384	2243384	ALN	24-Aug-2005	20-Sep-2010	S	USD	1,900,000	13,188
2243321	2243321	CCR	20-Aug-2005	20-Jun-2010	B	EUR	3,900,000	(11,908)

Set `Table Start = GID RMS ID`, the headings of the first two columns of the table. Note that the expression is case sensitive.

Set `Table End = Forward exchange transactions`, the first text following the table. The editor displays the table configuration:



If necessary, adjust the table definition and the columns. You can drag, add, or remove column borders. The following figure shows the text output.

```
Forward Exchange Report

GID       |  RMS ID    |  Ticker    |  Trade Date     |  Maturity Date
1855998   |  185599    |  XRPLN     |  10-Aug-2005    |  20-Sep-2010
1855391   |  185539    |  ELTLX     |  22-Aug-2005    |  20-Sep-2010
2241604   |  2241604   |  GD        |  22-Aug-2005    |  20-Sep-2010
2242285   |  2242285   |  MAX       |  22-Aug-2005    |  20-Sep-2010
2243384   |  2243384   |  ALN       |  24-Aug-2005    |  20-Sep-2010
2243321   |  2243321   |  CCR       |  20-Aug-2005    |  20-Jun-2010
2251699   |  2251699   |  REQ       |  20-Aug-2004    |  20-Jun-2010
2342409   |  2342409   |  DRR       |  16-Feb-2006    |  20-Mar-2011
2404000   |  2404000   |  MFP       |  10-Feb-2006    |  20-Mar-2011
```

Configuring the Second Table

The second table extends over multiple pages. The following figure shows the first few lines:

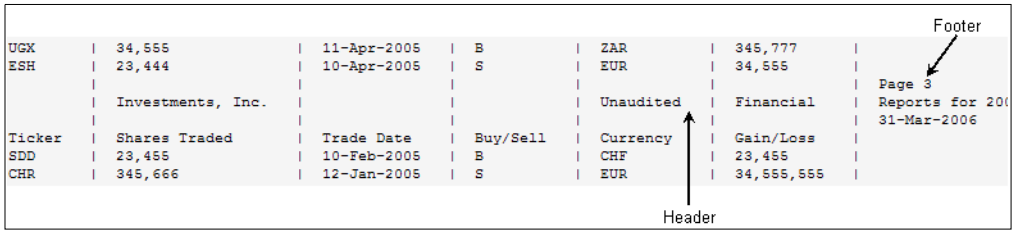
Trade Details Report

Ticker	Shares Traded	Trade Date	Buy/Sell	Currency	Gain/Loss
XRPLN	2,500	10-Aug-2005	S	EUR	900,000
ELTLX	1,000	22-Aug-2005	S	EUR	90,000
GD	65,233	22-Aug-2005	S	USD	19,000
MAX	22,333	22-Aug-2005	S	USD	3,900,000
ALN	12,788	24-Aug-2005	S	USD	1,900,000
CCR	12,300	20-Aug-2005	B	EUR	3,900,000

We configure the table as follows:

- Click Add Table. The system displays Table 2 in the `Tables` and `Name` fields.
- Set Table Start = `Ticker Shares Traded`
- Set Table End = `Conclusion`, the first body text after the table.
- Click Process to configure the table.
- Adjust the right borders of the `Shares Traded` and `Currency` columns.

A fragment of the formatted table is shown below. Notice how the page header and footer appear on each page of the formatted document, breaking the table into sections.



We can eliminate the page header and footer from the output document as follows:

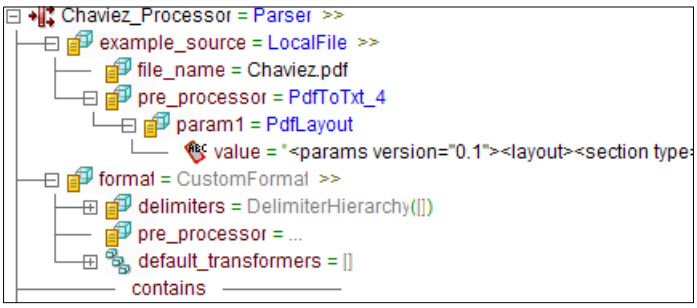
- Set Page Header = `Gain/Loss`
- Set Page Footer = `Page [1-9]`
- Click Process

These options remove the page header and footer from the formatted table:

```
XRPLN  |  2,500    |  10-Aug-2005  |  S  |  EUR  |  900,000     |
ELTLX  |  1,000    |  22-Aug-2005  |  S  |  EUR  |  90,000      |
GD     |  65,233   |  22-Aug-2005  |  S  |  USD  |  19,000      |
MAX    |  22,333   |  22-Aug-2005  |  S  |  USD  |  3,900,000   |
ALN    |  12,788   |  24-Aug-2005  |  S  |  USD  |  1,900,000   |
CCR    |  12,300   |  20-Aug-2005  |  B  |  EUR  |  3,900,000   |
REQ    |  5,600    |  20-Aug-2004  |  S  |  EUR  |  10,000,000  |
DRR    |  43,777   |  16-Feb-2006  |  S  |  EUR  |  900,000     |
MFP    |  99,333   |  10-Feb-2006  |  S  |  EUR  |  300,000     |
FMX    |  32,399   |  10-Aug-2005  |  S  |  USD  |  348,000     |
```

Returning to Conversion Agent Studio

Click OK to return to the Studio. The table configuration parameters are copied into the `value` property of the `PdfToTxt_4` processor.



To view the converted document, right-click the parser and click Open Example Source. The example source document appears in text format.

```
any way arising there from to you or any other entity for any loss or damage, direct or
indirect, arising from the use of this information.


Chaviez Investments, Inc. Unaudited Financial Reports for 2005
                                                31-Mar-2006
Forward Exchange Report

          |            |          |                |                |           |        |
GID       |  RMS ID    |  Ticker  |  Trade Date    |  Maturity Date |  Buy/Sell |  CCY   |  Face Value
          |            |          |                |                |           |        |  (CCY)
          |            |          |                |                |           |        |
1855998   |  185599    |  XRPLN   |  10-Aug-2005   |  20-Sep-2010   |  S        |  EUR   |  1,900,000
1855391   |  185539    |  ELTLX   |  22-Aug-2005   |  20-Sep-2010   |  S        |  EUR   |  1,900,000
2241604   |  2241604   |  GD      |  22-Aug-2005   |  20-Sep-2010   |  S        |  USD   |  1,900,000
2242285   |  2242285   |  MAX     |  22-Aug-2005   |  20-Sep-2010   |  S        |  USD   |  3,900,000
```

Alternatively, return to the table configuration editor and select the Output as XML option. The processor output is now displayed as XML.

```
Chaviez Investments, Inc. Unaudited Financial

Forward Exchange Report
</data>
  </section>

  <section name="Table 1" type="Table">
    <table>
      <row>
        <Value column="1"></Value>
        <Value column="2"></Value>
        <Value column="3"></Value>
        <Value column="4"></Value>
        <Value column="5"></Value>
        <Value column="6"></Value>
        <Value column="7"></Value>
        <Value column="8"></Value>
        <Value column="9"></Value>
        <Value column="10"></Value>
      </row>
```

CHAPTER 6

Formats

This chapter includes the following topics:

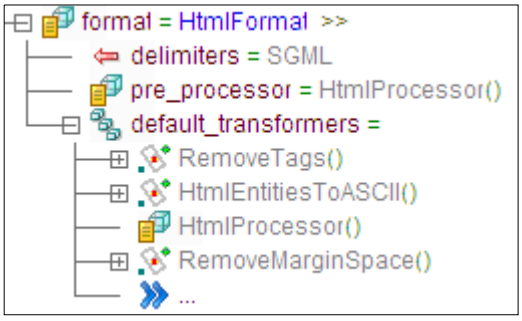
Defining Document Formats

The `format` property of a parser specifies the format of the documents that the transformation should process. The value of the property is one of the following format components:

```
BinaryFormat
CustomFormat
HtmlFormat
RtfFormat
TextFormat
XmlFormat
```

The format has properties of its own, which further define how the parser should interpret and process the input. Within a format, you can nest the following subcomponents:

Subcomponent	Description
Delimiters	A hierarchy of characters or strings that organize the information in the document, such as newlines and tabs.
Format preprocessors	Components that cleans up the source before the parser starts searching for anchors.
Default transformers	Transformers that the parser applies to the output of each anchor.



By configuring the properties and subcomponents, you can support an extraordinarily broad range of source documents.

This chapter describes the formats, delimiters, and format preprocessors that are available for your use. The subject of default transformers is discussed briefly here. For more information about default transformers, see “Transformers” on page 101.

Standard Properties of Formats

The format components have a standard set of properties, which are explained in the following tables.

Table 6-1. Basic Properties

Property	Description
`delimiters`	A hierarchy of characters or strings that organize the information in the document, such as newlines, spaces, tabs, commas, or vertical bars. You can also use a wildcard pattern to define the delimiters. The delimiter concept also encompasses positionally-structured data, where the fields are located at fixed offsets from one another. The value of the property is a delimiters component. For more information, see “Delimiters Component Reference” on page 43.
`pre_processor`	An optional format preprocessor that converts the source to a format that the parser can process. The format preprocessor acts on the source after any document processor that you defined. The purpose of the format preprocessor is to clean up whitespace or markup before the parser starts to search for anchors. For more information, see “Format Preprocessor Component Reference” on page 48.
`default_transformers`	A list of transformers that the parser applies in sequence to the output of each anchor. The purpose of the transformers is typically to clean up the output and remove markup codes. For more information, see “Transformers” on page 101.

Table 6-2. Advanced Properties

Property	Description
`name`	A name that you assign to the format. The name is displayed in the event log.
`remark`	A comment describing the format.

Format Component Reference

This section documents the format components that you can assign to the `format` property of a `Parser`.

BinaryFormat

This format is suitable for parsing binary files. It is also suitable for text files that you want to treat as a buffer of binary bytes.

By default, the `delimiters` property of this component has a value of `Positional`. The `pre_processor` and `default_transformers` properties are empty. For more information about the properties, see “Standard Properties of Formats” on page 40.

CustomFormat

This is a generic format, which you can use to process any type of source document.

By default, the `delimiters`, `pre_processor`, and `default_transformers` properties of this component are empty. You must configure the properties yourself. For more information about the properties, see “Standard Properties of Formats” on page 40.

Example

A source document has the following structure:

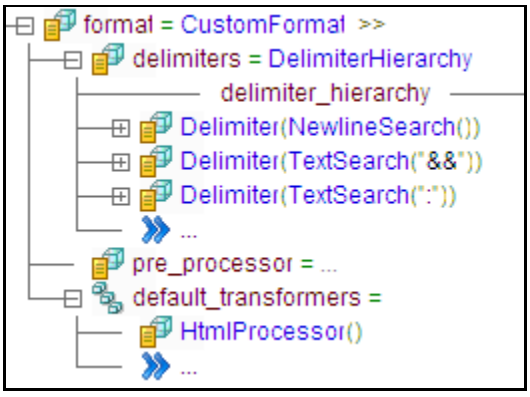
```
Ron      Lehrer && 547329876:27
Evelyn     Kern   &&     9875424:  53
```

Each line of the document is a record containing a person's name, ID number, and age. The fields are separated by the symbols `&&` and `:`. The fields contain multiple space characters at random locations.

One way to parse this document is by using `CustomFormat`. In the `delimiters` property of the format, assign a `DelimiterHierarchy` containing the symbols:

```
newline
&&
:
```

In the `default_transformers` property, assign the `HtmlProcessor`, which removes the extra spaces from the output.



HtmlFormat

This format is suitable for parsing HTML files.

It is also suitable for processing Microsoft Office documents. For this purpose, assign a document processor such as `WordToHtml` or `ExcelToHtml` to convert the Office document to HTML.

By default, the `delimiters` property of this component has a value of `SGML`. This causes the format to recognize the HTML delimiters such as < and >. The `pre_processor` is `HtmlProcessor`. The `default_transformers` are:

- `RemoveTags`. Removes HTML tags from the output
- `HtmlEntitiesToASCII`. Converts HTML entities such as `<` and `&quot;` to their plain text equivalents < and ", respectively.
- `HtmlProcessor`. Normalizes the whitespace, reducing any sequence of tabs, newlines, and spaces to a single space character.
- `RemoveMarginSpace`. Removes leading and trailing space.

For more information about the properties, see “Standard Properties of Formats” on page 40.

RtfFormat

This format is suitable for parsing RTF files.

By default, the `delimiters` property of this component has a value of `RTF`, which recognizes the standard RTF delimiter characters such as `\`. The `pre_processor` is `RtfProcessor`. The `default_transformers` are:

- `RtfToASCII`. Removes RTF control words from the output.
- `RemoveRtfFormatting`. Removes RTF formatting instructions from the text.
- `HtmlProcessor`. Normalizes the whitespace, reducing any sequence of tabs, newlines, and spaces to a single space character.
- `RemoveMarginSpace`: Removes leading and trailing space.

For more information about the properties, see “Standard Properties of Formats” on page 40.

TextFormat

This format is suitable for parsing text files.

In combination with a document processor, this format is also suitable for processing other types of documents. For example, you can use the `PdfToTxt_4`, `WordToTxt`, or `ExcelToTxt` processor to process PDF, Microsoft Word, or Microsoft Excel documents with this format.

By default, the `delimiters` property of this component has a value of `DelimiterHierarchy`, which allows you to define your own set of delimiters. The `pre_processor` is empty. The `default_transformers` are:

- `HtmlProcessor`. Normalizes the whitespace, reducing any sequence of tabs, newlines, and spaces to a single space character.
- `RemoveMarginSpace`. Removes leading and trailing space.

For more information about the properties, see "Standard Properties of Formats" on page 40.

XmlFormat

This format is suitable for parsing XML files.

Parsing an XML file means converting an XML source document to an XML output document. To do this, Conversion Agent treats the source XML as ordinary text. You can define delimiters, anchors, and other components just as you do for a regular text document.

This behavior is different from that of serializers or mappers that process XML documents. In serialization and mapping, Conversion Agent uses the XSD schema and the formal XML syntax rules to interpret the source document.

By default, the `delimiters` property of this component has a value of `SGML`. This causes the format to recognize the XML delimiters such as < and >. The `pre_processor` is `HtmlProcessor`. The `default_transformers` are:

- RemoveTags. Removes XML tags from the output.
- HtmlEntitiesToASCII. Converts XML entities such as `<` and `>` to their plain text equivalents, < and > respectively.
- `HtmlProcessor`. Normalizes the whitespace, reducing any sequence of tabs, newlines, and spaces to a single space character.
- `RemoveMarginSpace`. Removes leading and trailing space.

For more information about the properties, see "Standard Properties of Formats" on page 40.

Delimiters Component Reference

This section documents the delimiters components, which you can assign to the `delimiters` property of a format.

A delimiters component defines a hierarchy of characters or strings that organize the information in a document, such as newlines, spaces, tabs, commas, or vertical bars. You can also use a wildcard pattern to define the delimiters.

The delimiter concept is applicable both to rigidly structured documents that use predefined delimiter characters to separate the data fields, and to loosely structured text or HTML documents that are delimited by newlines and syntactic markup. The delimiter concept also encompasses positionally-structured data, where the fields are located at fixed offsets from one another.

Conversion Agent uses the delimiters for purposes such as determining the search criteria of `Content` anchors configured with the `LearnByExample` option.

For example, suppose you configure a format with the `TabDelimited` delimiters component. This defines a hierarchy using the following characters as delimiters:

```
Newline
Tab
```

You might define a `Content` anchor that is located two tab characters after the preceding `Marker` anchor in the example source, like this:

```
MARKER<tab>abc<tab>CONTENT
```

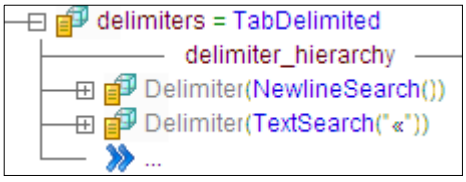
When Conversion Agent processes a source document, it searches for the `Content` two tabs after the `Marker`.

In a second example, you might define a `Content` anchor that is located three newlines and one tab after a `Marker` anchor, in the example source.

```
MARKER
abc<tab>de
fghi<tab>jkl<tab>mnop
pqrst<tab>CONTENT
```

Within the intermediate lines, the tabs are not counted because the newlines are higher in the hierarchy.

Many of the delimiters components, such as `TabDelimited` or `CommaDelimited`, display a predefined hierarchy of delimiters, which you can edit as required.



The `DelimiterHierarchy` component does not have a predefined hierarchy. You can insert whatever delimiters you need.

Some delimiter components, such as `SGML` or `PostScript`, have a built-in hierarchy that you cannot edit.

CommaDelimited

This delimiters component defines the following delimiter hierarchy:

```
Newline
Comma
```

`CommaDelimited` is suitable if each line of a text file contains a record, and each record contains data fields separated by commas.

You can add additional delimiters or edit the predefined hierarchy. The procedure is the same as for the `DelimiterHierarchy` component.

Example

In the example source document, suppose that a `Content` anchor follows a `Marker` anchor by two lines. In the third line, there are three commas, plus any other text, before the `Content` anchor, like this:

```
MARKER
abcdef, ghij
abc, def,ghi,CONTENT
```

If you assign the `CommaDelimited` component, the parser learns from the example source that the `Content` anchor always follows the `Marker` by two newlines and three commas. In another source document, the parser will successfully find the following `Content` anchor:

```
MARKER
     xyz, uvw, rst
,,,CONTENT
```

DelimiterHierarchy

This delimiters component allows you to define a custom delimiter hierarchy.

Under `DelimiterHierarchy`, you can nest any number of `Delimiter` or `EnclosingDelimiters` components.

Example

In the example source document, suppose that the anchors are separated by commas and surrounded by brackets, like this:

```
MARKER,,[CONTENT]
```

You might define a `DelimiterHierarchy` that contains:

```
comma //defined as a Delimiter component
[]    //defined as an EnclosingDelimiters component
```

From this example, the parser learns that the `Content` anchor follows the `Marker` by two commas and is surrounded by brackets. In another source document, the parser will successfully find the following `Content` anchor:

```
MARKER,abc,def[CONTENT]
```

Online Sample

For an online sample, see `samples\Projects\EDI\EDI.cmw`. The sample uses a `DelimiterHierarchy` to define the newline and `*` characters as delimiters, in an EDI source document.

HL7

This delimiters component defines the hierarchy that is used for parsing HL7 messages:

```
newline
vertical bar (|)
caret (^) or tab
```

You can add additional delimiters or edit the predefined hierarchy. The procedure is the same as for the `DelimiterHierarchy` component.

The HL7 messaging standard permits a message to define its own delimiters. You can parse the delimiter declaration of an HL7 message and create a dynamic delimiter definition in the following way:

1. Use `Content` anchors to retrieve the delimiter characters from the HL7 message header. Store the characters in variables.
2. Add `Delimiter` components under the `HL7` component.
3. To each `Delimiter` component, assign `TextSearch`.
4. Under the `TextSearch` component, assign one of the variables to the `text` property.

Positional

This delimiters component specifies that the parser should interpret the source document without using delimiters. Instead, it should locate each anchor by counting the characters from the beginning of the search scope. For more information about search scope, see “Anchors” on page 67.

Example

In the example source document, suppose that a `Content` anchor follows a `Marker` anchor by five characters, possibly including spaces, tabs, and so forth:

```
MARKERab cdCONTENTefg
```

If you assign the `Positional` component, the parser learns from the example source that the `Content` anchor always follows the `Marker` by five characters, and that it is seven characters long. In another source document, the parser will successfully find the following `Content` anchor:

```
MARKERd<tab>cbaCONTENTzy,xwv
```

Using Positional Parsing Together with Delimiters

You cannot add delimiters to the `Positional` component.

Sometimes, you might want to define a parser that uses delimiters to locate some anchors, and uses a positional definition for other anchors. To do this, select one of the other delimiters components. Do not use `Positional`. To define the location of an anchor positionally, you can assign the `OffsetSearch` option in the anchor properties. For more information, see “Anchors” on page 67.

PostScript

This delimiters component defines a delimiter hierarchy that is used for parsing Adobe PostScript documents.

You cannot edit the delimiter hierarchy of the `PostScript` component.

RTF

This delimiters component defines a delimiter hierarchy that is used for parsing RTF documents.

You cannot edit the delimiter hierarchy of the `RTF` component.

SGML

This delimiter component defines a delimiter hierarchy that is used for parsing SGML documents. It is often used for parsing HTML and XML, which are derivatives of SGML.

You cannot edit the delimiter hierarchy of the `SGML` component.

SpaceDelimited

This delimiters component defines the following delimiter hierarchy:

```
Newline
String of one or more space characters
```

`SpaceDelimited` is suitable if each line of a text file contains a record, and each record contains data fields separated by spaces.

You can add additional delimiters or edit the predefined hierarchy. The procedure is the same as for the `DelimiterHierarchy` component.

Example

In the example source document, suppose that a `Content` anchor follows a `Marker` anchor by two lines. In the third line, there are two space characters and one string containing multiple spaces before the `Content` anchor, like this:

```
MARKER
abcdef
abc def ghi         CONTENT
```

If you assign the `SpaceDelimited` component, the parser learns from the example source that the `Content` anchor always follows the `Marker` by two lines and three strings of spaces. In another source document, the parser will successfully find the following `Content` anchor:

```
MARKER
     xyz
ghi      def abc CONTENT
```

TabDelimited

This delimiters component defines the following delimiter hierarchy:

```
Newline
Tab
```

`TabDelimited` is suitable if each line of a text file contains a record, and each record contains data fields separated by tabs.

You can add additional delimiters or edit the predefined hierarchy. The procedure is the same as for the `DelimiterHierarchy` component.

Example

In the example source document, suppose that a `Content` anchor follows a `Marker` anchor by two lines. In the third line, there are three tab characters, plus any other text, before the `Content` anchor, like this:

```
MARKER
abcdef
abc<tab> de,f<tab>ghi<tab>CONTENT
```

If you assign the `TabDelimited` component, the parser learns from the example source that the `Content` anchor always follows the `Marker` by two lines and three tabs. In another source document, the parser will successfully find the following `Content` anchor:

```
MARKER
     xyz
<tab><tab><tab>CONTENT
```

Delimiter Subcomponent Reference

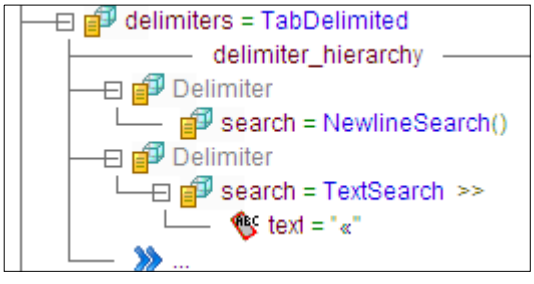
This section documents subcomponents that are used within delimiters components. For more information, see “Delimiters Component Reference” on page 43.

Delimiter

This subcomponent defines a delimiter character or string that separates anchors. You can add `Delimiter` subcomponents within a delimiter hierarchy.

Example

The `TabDelimited` component contains two `Delimiter` subcomponents. The first uses `NewlineSearch` to define the newline character as a delimiter. The second uses `TextSearch` to define the tab character as a delimiter. The tab is graphically represented as a « character.



The `SpaceDelimited` component also contains two `Delimiter` subcomponents. The first is identical to that of `TabDelimited`. The second uses a `PatternSearch` to define any string of one or more spaces as a delimiter. The regular expression `[]+` means “one or more space characters”.

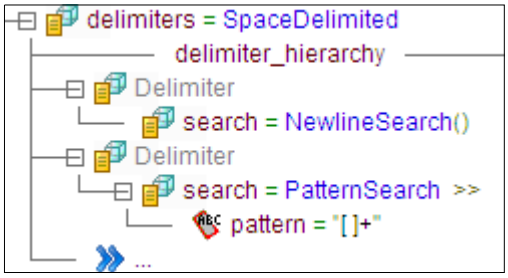


Table 6-3. Basic Properties

Property	Description
`Search`	The delimiter definition. The value is one of the following searcher components. For more information, see "Searcher Component Reference" on page 94. - `NewlineSearch`. The delimiter is a newline. - `PatternSearch`. The delimiter is defined by a regular expression. - `TextSearch`. The delimiter is an explicit string or a string that you retrieve dynamically from the source document.

EnclosingDelimiters

This subcomponent defines a pair of delimiter characters or strings, which surround anchors. You can add `EnclosingDelimiters` subcomponents within a delimiter hierarchy.

For example, the component is useful, to define the `{}` delimiters that surround blocks of C program code.

Table 6-4. Basic Properties

Property	Description
`opening`	The opening delimiter.
`closing`	The closing delimiter.
`escape_sequence`	A prefix in the source document, such as a backslash character `\`, which causes the parser to ignore an instance of the opening or closing delimiter.

Format Preprocessor Component Reference

This section documents the format preprocessor components that you can assign to the `pre_processor` property of a format. For more information, see "Format Component Reference" on page 41.

Do not confuse format preprocessors with document processors. The differences are as follows:

- You can assign a document processor to the `pre_processor` property of an input port, located under the `example_source` or `sources_to_extract` property of a transformation. You can assign a format preprocessor only to the `pre_processor` property of a format.
- Conversion Agent runs a document processor on the source document before it performs any other operations. The example pane of the IntelliScript editor displays the output of the document processor.
- Conversion Agent runs a format preprocessor on the text that appears in the example pane, before it searches for anchors. The output of the format preprocessor is not displayed.

For more information, see "Document Processors" on page 21.

HtmlProcessor

This format preprocessor, which is also available as a transformer, normalizes whitespace according to HTML conventions. It reduces any sequence of tabs, line breaks, and space characters to a single space character.

You can use this preprocessor to normalize whitespace in any type of text. It is not restricted to HTML documents.

For more information, see “Transformers” on page 101.

RtfProcessor

This format preprocessor normalizes the code of RTF files.

For more information, see “Transformers” on page 101.

CHAPTER 7

Data Holders

This chapter includes the following topics:

Overview

A data holder is an object that has one of the following types:

- An XML element
- An XML attribute
- A variable

XML elements and attributes are typically used for permanent storage. A parser, for example, stores its output in data holders of these types.

Variables are used for temporary storage. For example, a parser can store data that it extracts from a source document in a variable. It can process the data further before creating the output.

A common feature of all data holders is that they have XSD data types. In the case of elements and attributes, the data holders are defined in an XSD schema that you must supply. Variables are defined in an internal schema, which you can customize by adding user-defined variables.

XSD Schemas

When you create a parser, serializer, or mapper, you must supply one or more XSD schemas that define the structure of the XML. The schema defines the elements and attributes that the transformation can use.

You must add the schema to your project. You can then map the content of a document to elements and attributes that are defined in the schema.

About XSD

XSD is the commonly-used name for XML Schema, which is the industry-standard language for XML schema definitions. XSD originally stood for XML schema description, but this term is not used in the official XML Schema standard. The schema files typically have `*.xsd` filenames.

The XSD standard is maintained by the World Wide Web Consortium, http://www.w3.org. Since the standard was first published in 2001, XSD has rapidly replaced earlier schema languages such as DTD and XDR.

The following is a simple example of an XSD schema:

```
<?xml version="1.0" encoding="Windows-1252"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="First" minOccurs="0" type="xs:string"/>
              <xs:element name="Last" minOccurs="0" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Id" minOccurs="0" type="xs:string"/>
        <xs:element name="Age" minOccurs="0" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="gender" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The schema defines the elements and attributes that can occur in an XML document. The syntax lets a schema author specify the hierarchy and sequence of elements, whether the elements are mandatory or required, their data types, their possible values, and many other features.

The above sample schema defines an XML structure such as the following:

```
<Person gender="M">
  <Name>
    <First>Ron</First>
    <Last>Lehrer</Last>
  </Name>
  <ID>547329876</ID>
  <Age>27</Age>
</Person>
```

If you trace through the schema, you can observe the correspondence between definitions such as

```
<xs:element name="Person">
```

or

```
<xs:attribute name="gender" type="xs:string"/>
```

and the elements and attributes of the XML.

The elements and attributes have XSD data types, such as `xs:string`. An element that contains nested elements or attributes has a type of `xs:complexType`. The elements have many other properties, such as their required `sequence` and the minimum number of times that must occur in an XML document, `minOccurs`.

For more information about XSD syntax, see the following websites:

URL	Description
`http://www.w3.org`	The web site of the World Wide Web Consortium, which created and maintains the XML Schema standard.
`http://www.w3schools.com`	See this site for an excellent tutorial introduction to XSD.

How to Create XSD Schemas

You can create a schema in any XSD editor. Typically, an XSD editor has a user-friendly interface, which lets you create and edit schemas even if you do not know the XSD syntax. Some editors also let you convert an existing DTD or XDR schema to XSD, or to create an XSD schema from a sample XML file.

If you know the XSD syntax, you can also edit XSD schemas in a text editor such as Notepad.

Encoding of the XSD Schema

Save the schema in one of the supported input encodings. For more information about encodings, see “Encoding Properties” on page 214.

The schema encoding must be compatible with the working encoding that you use for the IntelliScript. This means that:

- The schema encoding is identical to the working encoding,

  or

- Every character in the schema has an equivalent in the working encoding. For example, if the schema uses the UTF-8 encoding, and the working encoding is Windows-1252, the schema must not contain Unicode characters that have no Windows-1252 equivalent.

When you add a schema from an external location to a project, Conversion Agent translates the project copy of the schema to the working encoding.

Included XSD Files

An XSD file can reference additional XSD files. This feature lets you maintain a large schema in a modular fashion.

Namespaces

If you plan to work with XML namespaces, assign the `targetNamespace` attribute of the schema. In Conversion Agent Studio, you can edit the alias that is assigned to the namespace. For more information, see “Namespaces Properties” on page 217.

To prevent ambiguities, Conversion Agent Studio displays a warning if you try to add two schemas that use the same alias for different namespaces. It assigns a different alias to one of the namespaces.

The Studio also warns if you try to add two schemas that use an empty alias for different namespaces.

Mixed Content

Conversion Agent supports XML elements that have mixed content, that is, elements containing both character data and nested elements. You can use the `mixed` attribute in a schema.

Conversion Agent distinguishes between character data before and after each element. For more information, see “Mapping Mixed Content” on page 57.

Unsupported XSD Features

The current Conversion Agent version does not support certain uses of XSD features. The following table lists the limitations, as of the time this book was written.

XSD Feature	Limitation
`redefine`	Redefining types and groups is not supported and should not be used.

`unique`
`key`
`keyref`
	Identity constraints can be used in a schema but are ignored.

`group`, in place of an entity	Model groups in place of XML entities, for example: `<city>Montr<c:eacute/>al</city>`, are not supported and should not be used. Other uses of groups are supported.
`long` `unsignedLong`	Data holders having the XSD type `long` or `unsignedLong` currently support integers with absolute values up to 2147483647. Larger values are not supported and may give incorrect results.

Precision of Numerical Data

Conversion Agent stores `xs:decimal` and `xs:float` data as strings, preserving the precision of the data.

In calculations, Conversion Agent converts decimal and float data to double-precision floating point, and it rounds the result to 15 decimal digits. This means that decimal data may lose some precision. For example, the result of `xs:decimal` 5.28 * 1 may be displayed as 5.28000000000001.

Conversion Agent normalizes `xs:decimal` values. For example, it stores 0004 as 4, -0 as 0, and 1.200 as 1.2.

Adding XSD Schemas to a Project

You must add or create at least one XSD schema in every Conversion Agent parser, serializer, or mapper project. The schema specifies the XML structure that the project needs to process.

If a schema includes other schema, add the main schema. When you do this, Conversion Agent adds the included schemas automatically.

To add an existing XSD schema to a project:

1. In the Conversion Agent Explorer view, right-click the `XSD` node of a project and click Add File.
2. Browse to the XSD file.

   If the XSD file is not in your project folder, Conversion Agent copies the file to the project folder.

   The `XSD` folder of the Conversion Agent Explorer displays the schema file. If the schema references any other XSD files, the `Include` folder of the Conversion Agent Explorer displays their names.

   Optionally, if the schema defines a target namespace, you can edit the namespace alias in the Conversion Agent project properties. For more information, see “Namespaces Properties” on page 217.

To create and edit a schema:

1. In the Conversion Agent Explorer, right-click the `XSD` node and click New > XSD.

   The Conversion Agent Explorer displays the new file with a default name such as `untitled1.xsd`.
2. Rename the file immediately.

   This is important to prevent errors in the references that a project creates to the schema. Conversion Agent Studio does not allow you to change the name of an existing schema file.

3. To edit the schema, double-click the XSD file.

   This opens the schema in an editor window. You can configure the Studio to open an editor of your choice. For more information about the XSD editor configuration, see *Using Conversion Agent Studio in Eclipse*.

Reloading a Schema after Editing

If you edit or modify an existing schema, Conversion Agent Studio prompts you to reload schemas. This ensures that the edited schema is available throughout the project.

You can reload schemas at any time by right-clicking the `XSD` node in the Conversion Agent Explorer and clicking Reload XSDs.

Validating Data Holders

If you edit a schema that belongs to an existing project, data holders referenced in the project may become invalid. For example, an anchor might reference an element that no longer exists in the schema.

To identify any problems of this sort, click Project > Validate. Any validation errors are displayed in the Problems view.

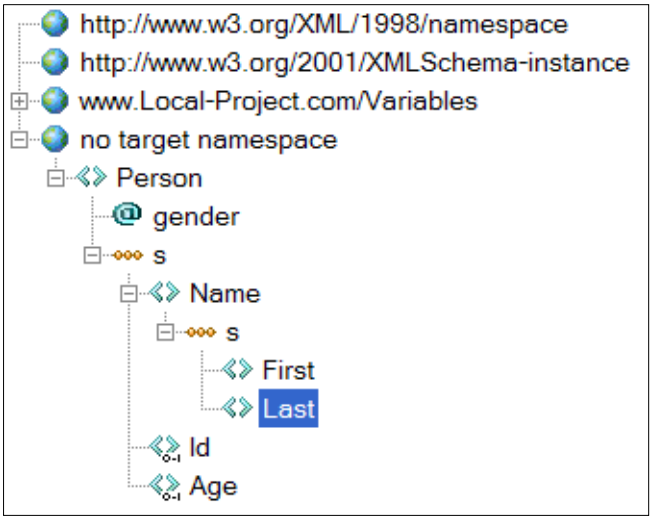
Viewing a Schema

In the Schema view of Conversion Agent Studio, you can view the namespaces, elements, and attributes of all schemas that belong to a project.

The namespace listing includes:

- Default entries for the Worldwide Web Consortium schema namespaces, beginning with http://www.w3.org. In most projects, you can ignore these entries.
- A `Variables` namespace, containing the variables defined in your project.
- An entry for each target namespace that is defined in the schemas that you have added to the project.

If you add one or more schemas that do not define a target namespace, they are displayed under the `no target namespace` heading.



Displaying an XML Sample of a Schema

You can generate and display a sample XML file that illustrates a schema.

To generate an XML sample:

1. Right-click the schema in the Conversion Agent Explorer.
2. Click Create Example XML.

The XML sample illustrates features such as:

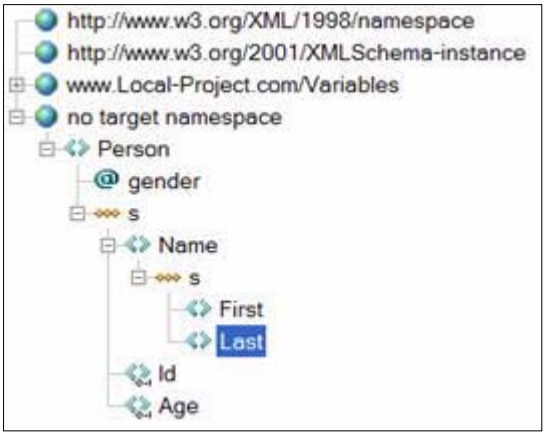
- The data type of an element or attribute. The sample displays `"a"` for string data, `"1"` for integer data, and `"1.1"` for floating point.
- The multiplicity of an element. If an element can occur more than once, the sample displays it more than once.

```
- <Message type="a" id="a">
  - <Patient id="a" gender="a">
      <f_name>a</f_name>
      <l_name>a</l_name>
      <birth_date>a</birth_date>
    </Patient>
    <Test_Type test_id="a">a</Test_Type>
  - <Result num="1">
      <type>a</type>
      <value>a</value>
      <range>a</range>
      <comment>a</comment>
      <status>a</status>
    </Result>
  - <Result num="1">
      <type>a</type>
      <value>a</value>
```

Using a Schema to Map Anchors

When you define a parser, you must map `Content` anchors to output data holders. When you define a serializer, you must map input data holders to `ContentSerializer` serialization anchors.

When you edit the `data_holder` property of an anchor, Conversion Agent displays a Schema view. Select the appropriate data holder.



Alternatively, when you configure a parser, you can drag text from the example source to a data holder in the Schema view. Conversion Agent Studio creates a `Content` anchor that maps the selected text to the data holder.

IntelliScript Representation of Data Holders

In the IntelliScript, data holders are identified by a modified XPath expression, such as:

```
data_holder = /Person/*s/Name/*s/First
```

Do not attempt to type this value. If you wish to modify the mapping, select the `data_holder` property and press Enter. This opens a Schema view, where you can select the new mapping.

The Conversion Agent XPath syntax is slightly different from the standard XPath syntax, which is `Person/Name/First`. Conversion Agent inserts `*s`, `*c`, and `*a`, which refer to the XSD terms `sequence`, `choice`, and `all`. The modifications resolve ambiguities when Conversion Agent uses XSD to construct XML output.

Mapping Mixed Content

If the schema supports mixed content, Conversion Agent considers each element to have before and after data holders. For example, consider the following mixed content:

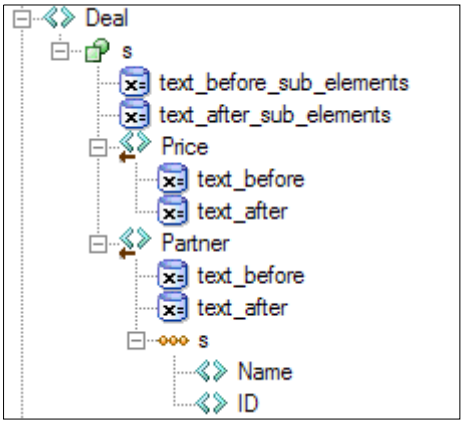
```
<Deal>
  We are pleased to offer you a price of
  <Price>34</Price>
  dollars. This is a special price for
  <Partner>
    <Name>Acme Gizmos, Inc.</Name>
    <ID>98765</ID>
  </Partner>
  valid only until December 31.
</Deal>
```

Conversion Agent considers this structure to contain data holders in the following locations:

- Immediately after the `<Deal>` tag, before any of the sub-elements.
- Before the `Price` element
- The `Price` element
- After the `Price` element
- Before the `Partner` element
- The `Partner/Name` and `Partner/ID` elements
- After the `Partner` element
- Immediately before the `</Deal>` tag, after all the sub-elements.

You can map the text `"We are pleased to offer you a price of"` to the data holder before the `Price` element. You can map `"dollars. "` to the data holder after `Price`, and `"This is a special price for "` to the data holder before `Partner`.

The Schema view displays the mixed-content data holders.



The IntelliScript displays the mixed content in a representation such as:

```
data_holder = /Deal/*s/Price/$text_before
```

Mapping XSI Types

Note: In this section, we use the conventional alias `xsi` for the namespace `http://www.w3.org/2001/XMLSchema-instance`. For more information about `xsi:type`, see http://www.w3.org/TR/xmlschema-1.

A schema can define derived XSD data types that can be used in place of a base type. In such cases, an XML document can define the actual data type of an element by specifying an `xsi:type` attribute.

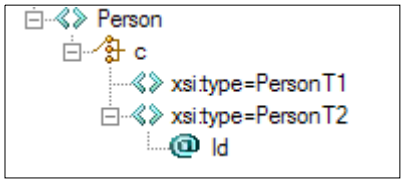
For example, an XSD schema defines a `Person` element having a type `PersonT1` and containing string content. It defines a type called `PersonT2` that extends `PersonT1` by adding an `Id` attribute. The following are valid `Person` elements:

```
<!-- base type PersonT1 -->
<Person>Ron Lehrer</Person>

<!-- derived type PersonT2 -->
<Person Id="547329876" xsi:type="PersonT2">Ron Lehrer</Person>
```

Conversion Agent interprets `xsi:type` attributes in input XML documents. It adds `xsi:type` attributes where necessary to output XML documents.

When you map a data holder to an element that can have multiple types, the Schema view displays the types.



Select the appropriate type according to the data that the transformation processes. For example, if you want a `Content` anchor to store data in a `Person` element having type `PersonT2`, select `xsi:type=PersonT2`. The IntelliScript displays the selection as follows:

```
data_holder=/Person/*c/xsi:type=PersonT2
```

In cases where the content might require either a `PersonT1` or `PersonT2` data holder, you can configure an `Alternatives` anchor that contains two `Content` anchors. One of the `Content` anchors is mapped to `PersonT1`, and the other to `PersonT2`. For more information, see “Alternatives” on page 79.

If you map a data holder to the unqualified element `Person`, the data holder defaults to the base type `PersonT1`. Thus the following mappings are equivalent:

```
data_holder=/Person
data_holder=/Person/*c/xsi:type=PersonT1
```

Generating Valid XML

By default, Conversion Agent generates XML that is valid according to the XSD schema that you have defined.

The Conversion Agent approach to validation differs from the conventional approach used in most XML applications. In the conventional approach, an application generates an XML document, and then checks that the output is conforms to a schema. The schema is applied after the generation, when the XML document already exists.

In the Conversion Agent approach, the schema is used as a guide while the XML is being generated. The schema is applied during the generation, and not afterwards. This approach helps transformations to succeed. It ensures the validity continually as the transformation proceeds.

Role of XSD in Parsing

This section explains some of the ways in which a parser uses XSD to ensure that it outputs valid XML.

The discussion presents examples of the behavior. For more information about the parameters that control the behavior, see “XML Generation Properties” on page 219.

Sequence of Elements

When Conversion Agent runs a parser, it organizes the output in the sequence that is required by the XSD schema.

For example, a schema may require that a `LastName` element precede a `FirstName` element. Conversion Agent creates the output in the locations defined by the schema, even if the anchors that produce the output are defined in the opposite sequence.

Number of Occurrences

A parser may attempt to insert multiple instances of an element in the output XML. Conversion Agent uses the schema to determine whether the new instances should be appended or should overwrite the existing elements. The parser deletes any excess elements beyond those that the schema permits, and it writes warnings in the event log.

In another example, suppose the schema defines an element without specifying a `minOccurs` or `maxOccurs` attribute. According to the XSD standard, the default `minOccurs` and `maxOccurs` values are 1, which means that the element must occur exactly once in the parser output. If the element is missing from the output, the parser can add it.

For more information, see “Multiple-Occurrence Data Holders” on page 64.

Missing or Empty Elements

In the project properties, you can configure whether a parser should insert empty elements to comply with an XSD schema. For more information, see “XML Generation Properties” on page 219.

XSD Data Types

Conversion Agent ensures that the text it stores in a data holder has the required XSD type. For example, if a `Content` anchor retrieves the string `"oranges 5 for a dollar"`, and the XSD type of the data holder is `xs:integer`, the anchor stores only the integer `5` in the data holder.

For more information, see “Using XSD Data Types to Narrow the Search Criteria” on page 76.

Role of XSD in Serialization and Mapping

A serializer or mapper checks that its input is valid according to the XML schema. There are two validation modes:

- **Partial validation.** Some deviations are allowed between the XML source document and the schema.
- **Strict validation.** The XML source document must conform strictly to its schema.

To define the validation level, assign the `validate_source_document` property of the `Serializer` or `Mapper` component.

If you use the strict mode, a validation error causes the serializer or mapper to fail. The Events view displays the errors.

If you use the partial mode, the transformation might proceed despite certain validation errors. For example, if there are more occurrences of an element than the schema permits, a serializer typically ignores the excess elements and processes the valid ones, and it writes a warning in the events log. Similarly, it might ignore an element containing an invalid data type.

Conversion Agent uses the Xerces C XML parser, version 2.7, to perform validation. For more information about the validation characteristics, see http://xerces.apache.org/xerces-c.

Note: Conversion Agent 4.4 and earlier used partial validation. For compatibility with existing transformations developed in these earlier versions, partial validation remains the default.

Variables

Variables are temporary data holders that you can use in place of XML elements or attributes. Variables are useful if you need to store a value temporarily during the operation of a transformation, and you do not need to output the value in the XML.

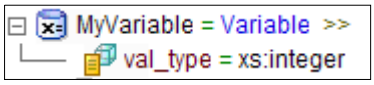
For example, suppose you want a parser to read two `Content` anchors and concatenate their values. You might map each `Content` anchor to a user-defined variable. You can then use an action to concatenate the variables and output the result to an XML element.

In addition to the user-defined variables, Conversion Agent has several pre-defined system variables. The system variables are used to store information that is needed in certain operations.

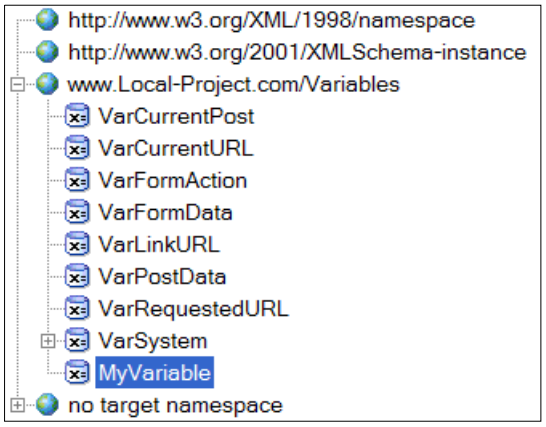
User-Defined Variables

To define a variable:

1. On the Studio menu, click IntelliScript > Insert > Variable.
2. Enter a name for the variable and press Enter.
3. Select the XSD data type that the variable can store, such as `xs:string` or `xs:integer`.



The variable appears under the `Variables` namespace in the Schema view.



Alternatively, you can define a `Variable` component by editing the IntelliScript directly, without using the menu command. You can add the component only at the top level of the IntelliScript, not at a nested level.

System Variables

Several system variables are defined in every Conversion Agent project. The following paragraphs describe the variables and the ways in which they are used.

Variables Used to Access Source Documents

Several of the system variables store data that actions can use when they access source documents. For example, the `RunParser` action can use:

Variable	Description
`VarLinkURL`	A file path or a URL address.
`VarPostData`	A string containing form data that should be submitted.

The following variables are used in the `SubmitForm` and `SubmitFormGet` actions:

Variable	Description
`VarFormAction`	The URL to which a form should be posted. The variable corresponds to the `action` attribute of the HTML `<form>` element.
`VarFormData`	A string containing the form data that should be submitted. This is a multiple-occurrence variable. Each occurrence is a complete instance of the form data. For more information, see “Multiple-Occurrence Data Holders” on page 64.

Read-Only Access Variables

The following variables are read-only. A transformation can use them to revisit a source document.

Variable	Description
`VarRequestedURL`	The path or URL of the source document that a parser is processing.
`VarCurrentURL`	The path or URL of the current file that a parser is processing. Usually, this is the same as `VarRequestedURL`. If the parser is configured with certain preprocessors, `VarCurrentURL` might point to a temporary file rather than the original source document. `VarRequestedURL` always points to the source document.
`VarCurrentPost`	The form data that a parser submitted to retrieve the current page.

Read-Only System Time Variables

`VarSystem` is a read-only variable that returns system information. It is a structure containing several nested variables:

Variable	Description
`VarSystem/ExecStartTime/Year`	Year when the transformation began execution
`VarSystem/ExecStartTime/Month`	Numerical month
`VarSystem/ExecStartTime/MonthName`	Name of month
`VarSystem/ExecStartTime/Day`	Day of month
`VarSystem/ExecStartTime/DayName`	Day of week
`VarSystem/ExecStartTime/Hour`	Hour
`VarSystem/ExecStartTime/Minute`	Minute
`VarSystem/ExecStartTime/Second`	Second
`VarSystem/ExecStartTime/Millisecond`	Millisecond

You can use `VarSystem` to insert a timestamp in the output of a transformation.

Variables Used for Failure Handling

`VarLastFailure` stores the most recent component failure that occurred in a transformation. For example, it might record an instance of a `Marker` anchor that failed to find the marker text. You can configure a component to write `VarLastFailure` to a user log when a failure occurs. For more information, see “Failure Handling” on page 226.

`VarServiceInfo` stores the service name, directory location of the user log, and the file name of the user log.

`VarLastFailure` and `VarServiceInfo` are structures containing the following nested variables:

Variable	Description
`VarLastFailure/InternalId`	Failure identifier
`VarLastFailure/Text`	Failure description
`VarLastFailure/Location`	Failure location
`VarLastFailure/AnchorName`	Name of the component that failed
`VarLastFailure/Data`	Additional information about the failure
`VarServiceInfo/ServiceName`	Name of the Conversion Agent service
`VarServiceInfo/StandardError/StandardErrorDir`	Directory path of the user log
`VarServiceInfo/StandardError/StandardErrorName`	File name of the user log

Mapping Anchors to Variables

You can map a `Content` anchor to a variable in the same way that you map to any other data holder.

Do not map an anchor to a read-only system variable.

Using Variables in Actions

Variables are often used as the input of actions. You can use a variable in the same way as you use other data holders. For more information, see “Actions” on page 133.

Initializing Variables at Runtime

Optionally, you can initialize the values of variables before a transformation runs. There are several ways to do this:

- In the IntelliScript, you can configure the `initialization` property of the `Variable`.

  The initial values that you set by this approach are used when you run the transformation in Conversion Agent Studio or as a service.

- In the Run dialog box of the studio, you can click the Details button and set the initial values.

  The values that you set in this way are used when you test the transformation in the Studio. For testing purposes only, the values override the ones that you set in the `initialization` property. They have no effect when you run the transformation as a service.

- An application can pass the initial values as service parameters to a service at runtime.

  The service parameters override the `initialization` property of the variables. If the IntelliScript specifies an initial value, and you also pass a value from an application, the latter value is used.

  For more information about how to pass service parameters, see the API references.

Variable Component Reference

This section documents the `Variable` component, which you can add to the IntelliScript.

Variable

A `Variable` is a user-defined variable.

You can use variables for temporary storage, in the same locations that you use an XML element or attribute. For example, you can map a Content anchor to a variable, and you can use a variable as the input of an action.

Variables have XSD data types. They are displayed in the Schema view and in the IntelliScript. You can define a variable only at the top (global) level of the IntelliScript.

Table 7-1. Basic Properties

Property	Description
`val_type`	The data type that the variable can store. Assign a standard type such a: `xs:string` or `xs:integer` or a custom type that is defined in the schemas belonging to the project. A custom type can be either simple or complex. In the latter case, the variable is a structure containing nested fields.

Table 7-2. Advanced Properties

Property	Description
`list`	Select this option to create a multiple-occurrence variable. For more information, see “Multiple-Occurrence Data Holders” on page 64.
`initialization`	An initial value for the variable, assigned when the transformation starts. Select `InitialValue` and enter the value. You can initialize variables that have simple data types. Initialization of complex variables is not supported. Service parameters, which an application passes to a service at runtime, override the `initialization` property. You can use the `initialization` property to set a default that the transformation uses if an application does not pass a service parameter. For more information, see “Initializing Variables at Runtime” on page 63.

Multiple-Occurrence Data Holders

In an XSD schema, you can use the `maxOccurs` attribute to set the maximum number of times that sibling elements can occur in an XML document. Likewise, you can define a variable that can occur either once or multiple times. An element or variable that can occur only once is called a single-occurrence data holder. An element or variable that can occur more than once is called a multiple-occurrence data holder.

Single- and multiple-occurrence data holders behave differently when Conversion Agent stores data in them, for example, when you map `Content` anchors to a data holder.

- In a single-occurrence data holder, each assignment overwrites the preceding assignment.
- In a multiple-occurrence data holder, each assignment generates a new occurrence of the data holder.

To understand this, suppose that an XSD schema defines an XML element called `FirstName`. If `maxOccurs = 1`, this is a single-occurrence data holder. If a parser maps more than one `Content` anchor to the `FirstName` element, the output contains only the final mapping.

Consider what would happen if you parse a source document that is a list of first names:

```
Jack Jennie Larissa
```

We assume that each name is a `Content` anchor mapped to `FirstName`. Each name overwrites the value of `FirstName`. The output contains only the mapping:

```
<FirstName>Larissa</FirstName>
```

Now suppose that `maxOccurs = unbounded`. This means that `FirstName` is a multiple-occurrence data holder. If you map multiple `Content` anchors to the element, the parser generates a list of names. The output is:

```
<FirstName>Jack</FirstName>
<FirstName>Jennie</FirstName>
<FirstName>Larissa</FirstName>
```

The same principle applies to variables. If you map multiple anchors to a multiple-occurrence variable, each anchor generates a new occurrence of the variable. You can use this feature, for example, to prepare input for the `AppendListItems` and `CombineValues` actions, which concatenate the occurrences.

Attributes

An XML attribute is always a single-occurrence data holder. An attribute cannot be multiple-occurrence because XML does not permit the same attribute to appear more than once in the same element.

An attribute can have an XSD type that is a space-separated list. The `names` attribute in the following element is an example:

```
<Countries names="USA Canada Mexico"/>
```

Conversion Agent treats the attribute as a single-occurrence data holder with an XSD list type. For more information, see “Using XSD Data Types to Narrow the Search Criteria” on page 76.

Indexing

By default, Conversion Agent accesses the instances of a multiple-occurrence data holder sequentially. You can access the instances non-sequentially by using the indexing feature. For more information, see “Locators, Keys, and Indexing” on page 187.

Destroying the Occurrences

Under certain circumstances, you might want to destroy all existing occurrences of a multiple-occurrence data holder, and start creating new occurrences from the beginning of the list. This is useful, for example, if you are parsing an iterative structure, and you want to keep only the last iteration. You can destroy the occurrences that store data from the earlier iterations.

You can achieve this effect by defining a single-occurrence data holder that contains a nested, multiple-occurrence element. When you re-use the single-occurrence data holder, the nested occurrences are destroyed.

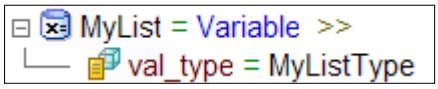
The following scenario is a typical example.

1. Add the following schema to a project:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:complexType name="MyListType">
    <xs:sequence>
      <xs:element name="item" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

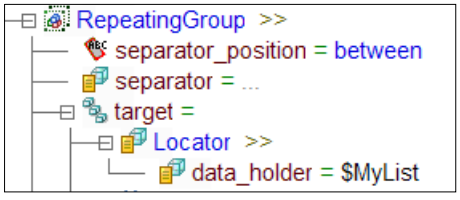
   The schema defines a custom XSD data type called `MyListType`. The type contains a nested, multiple-occurrence element called `item`.

2. Define a single-occurrence variable called `MyList`, which has the data type `MyListType`.



3. Use the variable as the target of an iterative structure.

   For more information, see “Locators, Keys, and Indexing” on page 187.



   Each iteration re-uses the single occurrence of `MyList`. At the start of the iteration, the nested `item` elements are destroyed. Anchors within the iterative structure, such as a nested `RepeatingGroup`, start assigning the `item` elements from the beginning of the list.

Online Sample

For an example of how to destroy multiple occurrences of a data holder, see the following online sample:

```
samples\Projects\ResetListVariable\ResetListVariable.cmw
```

CHAPTER 8

Anchors

This chapter includes the following topics:

Overview

Anchors are the components that let a parser hook into specific locations in a source document, for the purpose of finding data and storing it in data holders. An anchor is a signpost that you place in a document, indicating the position of the data.

This chapter explains the different types of anchors and how you can use them in parsers.

Marker and Content Anchors

The most commonly used anchors are called `Marker` and `Content` anchors. These anchors are often used as a pair:

- A `Marker` anchor labels a location in a document.
- A `Content` anchor retrieves text from the location.

To understand these anchors, imagine a printed questionnaire. The first line typically asks for the person's last name and first name, with each label followed by a blank space to receive the information. In Conversion Agent terminology, the printed labels `Last Name` and `First Name` are `Marker` anchors, and the blank spaces are `Content` anchors. The anchors provide a means to home in on the data and extract it from the source document.

Other Anchor Types

In addition to `Marker` and `Content` anchors, Conversion Agent provides many other anchor types that you can use to parse documents. For example, `Group` and `RepeatingGroup` anchors help you specify the organization of the data fields. An `Alternatives` anchor lets you specify multiple kinds of data that might occur at a particular location in a source document.

How Anchors and Delimiters Work Together

In Conversion Agent Studio, you can define the anchors in the example source document. The parser learns how to parse the document by examining the anchors and the delimiters that separate them. For more information about delimiters, see “Formats” on page 39.

For example, suppose you have specified that your document uses a tab-delimited format. A line in the example source reads

```
First name:<tab>Ron
```

where `<tab>` is a tab character.

You can define `First name:` as a `Marker` anchor. You can define `Ron` as a `Content` anchor. The parser learns from these definitions that it should search a source document for the string `First name:`. It should then skip over a single tab delimiter and retrieve the text that follows the tab.

Suppose you run the parser on another source document, which contains the following text:

```
First name:<tab>Jack
```

The parser finds the anchors as above and retrieves the text `Jack`.

Now suppose that the source document reads:

```
First name:<tab>Jack<tab>Age:<tab>34
```

The parser still retrieves the text `Jack`, rather than `Jack<tab>Age<tab>34`. This works because you have defined the tab character as a delimiter. Conversion Agent understands that the `Content` anchor starts after the first tab and ends before the second tab. Of course, you might define additional anchors that retrieve Jack's age, which is `34`.

Note: The above examples describe one possible behavior of the anchors and delimiters. The anchors have many properties that let you alter this behavior. For instance, you can define a `Content` anchor that ignores tabs, even in a tab-delimited format. For more information, see “How a Parser Searches for Anchors” on page 72.

Mapping Content Anchors to Data Holders

A `Content` anchor stores the text that it extracts from a source document in a data holder. For example, you might configure a `Content` anchor to store its result in an XML element called `FirstName`. If the `Content` anchor retrieves the text `Jack`, the parser produces the following output:

```
<FirstName>Jack</FirstName>
```

More precisely, you might specify that the anchor should store the retrieved text at the path `/Person/*s/FirstName`, which refers to the XSD schema. The actual parser output would be:

```
<Person>
  <FirstName>Jack</FirstName>
</Person>
```

On the other hand, suppose that the XSD schema defines `FirstName` as an attribute of the `Person` element. You might map the `Content` anchor to `/Person/@FirstName`. The output would be:

```
<Person FirstName="Jack" />
```

You must map to a data holder that has an appropriate data type. For example, do not map `Jack` to an XML element that has an XSD `integer` data type, or to an XML element that has a complex data type containing nested elements. For more information about this rule, see “Using XSD Data Types to Narrow the Search Criteria” on page 76.

Note: Do not attempt to type a path such as `/Person/*s/FirstName` in Conversion Agent Studio. When you edit a property whose value is a data holder, the Studio displays a Schema view, where you can select the data holder. The Studio displays the path in the IntelliScript.

Mapping to Variables

You can map an anchor to a data holder that is an XML element, an XML attribute, or a variable. The variable option is useful if you want to use the data in a subsequent processing step, but you do not want to include the raw data in the parser output.

For example, suppose you want to extract several numbers from a source document and output their sum in the XML. You do not want the individual numbers in the output. You can map the `Content` anchors that retrieve the numbers to variables, and use a `CalculateValue` action to compute and output the sum.

You might also map to a variable that you use in a subsequent anchor, for example, to define a dynamic search text for a `Marker` anchor.

Mapping to Multiple-Occurrence Data Holders

If you map `Content` anchors to a single-occurrence data holder, each assignment of the data holder overwrites the previous assignment.

If you map to a multiple-occurrence data holder, each assignment generates a new occurrence of the data holder. For example, if each `Content` anchor retrieves a person's name, the output is a list of names:

```
<FirstName>Jack</FirstName>
<FirstName>Jennie</FirstName>
<FirstName>Larissa</FirstName>
```

For more information, see “Multiple-Occurrence Data Holders” on page 64.

Mapping to Mixed-Content Elements

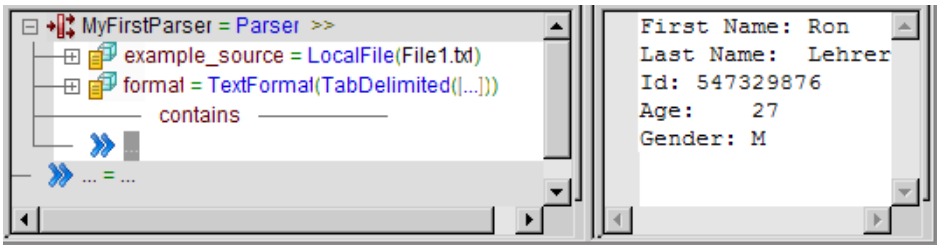
The term mixed content refers to an XML element that contains both character data and nested elements. If the XSD schema permits an element to have mixed content, the Schema view displays `before` and `after` data holders for the elements. This lets you map a `Content` anchor to character data that is located before or after a particular nested element. For more information, see “Mapping Mixed Content” on page 57.

Defining Anchors

When you define a `Parser` component, you must add a sequence of anchors. The parser operates by searching for the anchors in the source document and by running the operations that you have configured the anchors to perform.

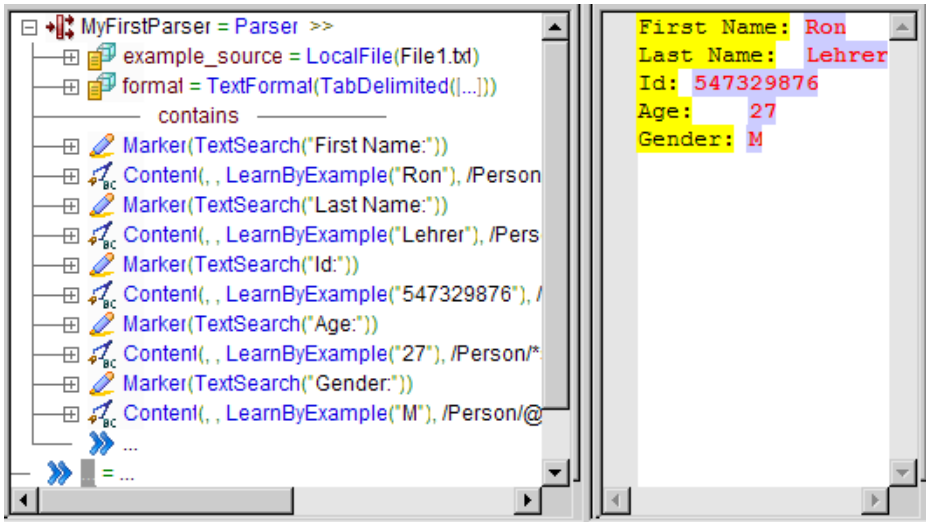
Where to Define Anchors

In the IntelliScript, the anchors are nested within a `Parser` configuration.



If you press Enter at the indicated location, Conversion Agent displays a drop-down list, which includes the anchors, as well as the other components that you can add.

After you add the anchors, the Studio highlights the anchors in the example source.



Some types of anchors can contain nested anchors. For example, you can nest anchors within an `Alternatives`, `Group`, or `RepeatingGroup` anchor.

Sequence of Anchors

The sequence of the anchors should be the sequence of text in the source document.

For example, suppose that the source document is:

```
First Name: Ron
Last Name: Lehrer
```

Assuming that you define `First Name` and `Last Name` as `Marker` anchors, and that you define `Ron` and `Lehrer` as `Content` anchors, the required sequence of anchors in the parser configuration is:

Anchor	Text in the Source Document
`Marker`	`First Name`
`Content`	`Ron`
`Marker`	`Last Name`
`Content`	`Lehrer`

Exception: Variable Source Sequence

Some source documents may have a variable sequence. For example, suppose that the source document may have either of the following formats:

```
First Name: Ron
Last Name: Lehrer
```

or

```
Last Name: Lehrer
First Name: Ron
```

In such cases, you can use the `marking` property to change the search scope of the anchors. For more information, see "How a Parser Searches for Anchors" on page 72.

Select-and-Click Approach for Marker and Content Anchors

Optionally, you can add `Marker` and `Content` anchors by a select-and-click approach.

To add anchors by the select-and-click approach:

1. Select the anchor text in the example source file.
2. Right-click the selected text and click Insert Marker or Insert Content.
3. In the IntelliScript editor or in the IntelliScript Assistant view, set the anchor properties.

Drag-and-Drop Approach for Content Anchors

You can add `Content` anchors by a drag-and-drop approach in the example source and Schema view.

To add anchors by the drag-and-drop approach:

1. In the example pane, select the anchor text.
2. Drag the text to a data holder in the Schema view.

   This creates a `Content` anchor that is mapped to the data holder.
3. Edit the IntelliScript and set the other anchor properties.

You can also drag and drop from the example pane to the IntelliScript pane. For example, you can drag to the `text` property of an anchor that is defined with the TextSearch option.

Using the IntelliScript to Define Anchors

You can create any type of anchor by editing the IntelliScript. The procedure is identical to editing any other component in the IntelliScript:

To configure anchors by editing the IntelliScript:

1. At the desired anchor location, select the three dots symbol (. . .) and press Enter.
2. Select or type the anchor name.
3. Press Enter again to confirm your selection.
4. Edit the anchor properties.

Standard Anchor Properties

In this section, we review certain standard properties that are found in many anchors. For more information about the properties of specific anchors, see the “Anchor Component Reference” on page 79.

Property	Description
`name`	A name that you assign to the anchor. Conversion Agent displays the name in the event log. This can help you find an event that was caused by the particular anchor.
`remark`	A comment describing the anchor.
`disabled`	If selected, the parser ignores the anchor. This is useful for testing and debugging, or for making minor modifications in a parser without deleting the existing anchors. Disabling an anchor disables all its nested components, nested anchors, transformers, etc.
`optional`	By default, if an anchor fails, its parent component also fails. If you select the `optional` property, the parent component does not fail. You can select the `optional` property to define an anchor that may or may not exist in a source document. If the anchor does not exist, the `Parser` in which the anchor is nested continues. If the anchor is nested within a `Group` anchor, the optional property prevents the `Group` from failing. If the anchor is in a `RepeatingGroup`, the property prevents an iteration of the `RepeatingGroup` from failing. For more information, see “Failure Handling” on page 226.
`on_fail`	If the anchor fails, writes an entry in the user log. For more information, see “Failure Handling” on page 226.
`direction`	The direction in which Conversion Agent searches for the anchor, within the search scope. If `direction = forward`, the parser finds the first instance of the anchor within the search scope. If `direction = backward`, the parser finds the last instance. For example, suppose the search scope for a `Marker` anchor contains five instances of the word `Balance`. If `direction = forward`, the parser finds the first instance of `Balance`. If `direction = backward`, it finds the last instance. For a `Marker` anchor, you can modify this behavior by using the `count` property. For example, if `direction = backward` and `count = 2`, the parser finds the second to last instance. For more information, see “How a Parser Searches for Anchors” on page 72.
`marking`	Specifies whether an anchor should be used as a reference point to find the succeeding anchor. The options are: - `full`. Places a reference point before and after the current anchor. - `begin position`. Before only. - `end position`. After only. - `none`. Neither. You can use this property to control the search scope for the succeeding anchor. For more information, see “How a Parser Searches for Anchors” on page 72.
`phase`	The processing phase during which Conversion Agent searches for the anchor, `initial`, `main`, or `final`. By default, Conversion Agent searches for `Marker` anchors during the initial phase and for `Content` anchors during the main phase. For more information, see “How a Parser Searches for Anchors” on page 72.
`no_initial_phase`	This property applies to components that have nested anchors. If the property is selected, the anchor has no initial phase. This overrides the option `phase = initial` in the immediately nested anchors, and changes it to `main`.

How a Parser Searches for Anchors

To design a parser correctly, it is important that you understand how Conversion Agent searches for the anchors in the parser configuration. There are three main concepts:

- Search phase

- Search scope
- Search criteria

This section explains the concepts, and how you can control each of them by setting the anchor properties.

Search Phases

Conversion Agent searches for a sequence of anchors in three phases:

- Initial
- Main
- Final

By default, all `Marker` anchors are in the initial phase and all `Content` anchors are in the main phase. This means that Conversion Agent first finds the `Marker` anchors, and then it finds the `Content` anchors between them.

To understand this, consider a parser that processes the following source document:

```
First name: Ron    Last name: Lehrer
```

Suppose you have defined the anchors in the following way, with default anchor properties:

Anchor	Text in the Source Document	Phase
`Marker`	`First name:`	`Initial`
`Content`	`Ron`	`Main`
`Marker`	`Last name:`	`Initial`
`Content`	`Lehrer`	`Main`

In the initial phase, Conversion Agent searches for the `Marker` anchors:

- It searches for `First name:`.
- It searches for `Last name:` at a location that follows `First name:`.

In the main phase, Conversion Agent searches for the `Content` anchors:

- It searches for the `Ron` anchor at a location between `First name:` and `Last name:`.
- It searches for the `Lehrer` anchor at a location after `Last name:`.

Nested Phases

Anchors that have nested anchors, such as `Group`, have nested phases. For example, if a `Group` anchor runs in the main phase of a parser, a `Marker` anchor that is nested in the `Group` runs in a nested initial phase. The nested initial phase is part of the parser main phase, but it is before the other anchors in the `Group`.

Another example is a `RepeatingGroup` anchor, which searches for both separators and for nested anchors. In order to identify the nested anchors correctly, it searches for the separators before it searches for the nested anchors.

Search Scope and Search Criteria

The above example of search phases illustrates the concepts of search scope and search criteria. The search scope is the portion of a document where Conversion Agent searches for an anchor. The search criteria are the rules by which Conversion Agent finds the anchor within the search scope.

In the initial phase, Conversion Agent starts searching for the `Marker` anchor containing `First name:` at the beginning of the document. The search scope for this anchor is the entire document. The search criterion is that the anchor must contain the text `First name:`.

The search scope for the `Last name:` anchor starts at the end of `First name:`, and extends to the end of the document. The search criterion is that the anchor must contain the text `Last name:`.

In the main phase, the parser interpolates the `Content` anchors between the `Marker` anchors. The search scope for the `Ron` anchor extends from the end of the `First name:` anchor to the beginning of the `Last name:` anchor. Assuming that the parser uses a space-delimited format, the search criteria are to retrieve all the text in the search scope, after the leading space character and before the second space character.

The search scope for the `Lehrer` anchor is from the end of `Last Name:` to the end of the document. The search criteria are similar to those for the `Ron` anchor.

We can add this analysis to the anchor table that we presented above. The table now describes the complete method by which the parser finds the anchors.

Anchor	Text in the Source Document	Phase	Search Scope	Search Criteria
`Marker`	`First name:`	`Initial`	Entire document	Text = `First name:`
`Content`	`Ron`	`Main`	End of `First name:` to start of `Last name:`	After the leading space Before the next space
`Marker`	`Last name:`	`Initial`	End of `First name:` to end of document	Text = `Last name:`
`Content`	`Lehrer`	`Main`	End of `Last name:` to end of document	After the leading space Before the next space

Adjusting the Search Phase

By assigning the `phase` property of an anchor, you can change the phase in which Conversion Agent searches for the anchor.

Consider the following source document:

```
CONTENT<10 characters>MARKER
```

In this example, the `Marker` anchor is located 10 characters after the `Content` anchor.

By default, Conversion Agent searches for the `Marker` in the initial phase, and it searches for the `Content` in the main phase. This won't work here, because Conversion Agent cannot find the `Marker` unless it has already found the `Content`!

The solution is to change the `phase` property of one of the anchors. You can change the `Content` to the initial phase, or the `Marker` to the main phase. In either case, Conversion Agent finds the anchors.

Adjusting the Search Scope

There are two ways to adjust the search scope for an anchor:

- By setting the `phase` property of the anchor or the surrounding anchors
- By setting the `marking` property of the surrounding anchors

Phase Property

If a `Content` anchor lies between two `Marker` anchors, then by default, the search scope for the `Content` is the segment between the `Marker` anchors.

If you change all the anchors to the same phase, the search scope of the `Content` is no longer bounded by the second `Marker`. It is from the end of the first `Marker` to the end of the document.

As an example, consider the following source document:

```
Tree Fig    Date<tab>October 27, 2003 (pruned)
Tree  Date Palm Date April 27, 2003<tab>(planted)
```

The example assumes that the source document has a loose structure, containing varying numbers of spaces, tabs, or other symbols interspersed in the text, so we cannot easily use the spaces and tabs as delimiters. An example like this might arise in parsing word-processor documents.

We can parse this document using a `RepeatingGroup` anchor, which contains nested `Marker` and `Content` anchors. The `Marker` anchors are the strings `Tree` and `Date`. The `Content` anchors are everything between the `Marker` anchors, including the spaces and tabs.

The problem in parsing this document is in the second iteration of the `RepeatingGroup`, which parses the second line. If we leave the `Marker` anchors in the initial phase, Conversion Agent incorrectly considers the first instance of the word `Date` to be a `Marker`. In the main phase, it fails to find `Date Palm` because the search scope is between the two `Marker` anchors, and there is no text between them.

A possible solution is to move the `Marker` for `Date` to the main phase, and to define the `Content` anchor, `Date Palm`, using an expression that searches for a tree name of one or two words. In the initial phase of the `RepeatingGroup`, Conversion Agent finds the `Marker` for `Tree`. In the main phase, it finds `Date Palm` followed by the `Marker` for `Date`.

With the new phase setting, we have changed the search scope for the tree name. The scope is now from `Tree` to the end of the iteration, and Conversion Agent finds `Date Palm` successfully.

Marking Property

Consider the following source-document structure:

```
MARKER
%%%CONTENT A
^^^CONTENT B
```

Suppose that the sequence of `Content A` and `Content B` varies among the source documents. In some documents, `Content B` precedes `Content A`.

In that case, the search criteria are:

- `Content A` and `Content B` both follow the `Marker` anchor.
- `Content A` begins with `%%%`, and `Content B` begins with `^^^`.

By default, the search scope for `Content A` is from the end of the `Marker` to the end of the document. The search scope for `Content B` is from the end of `Content A` to the end of the document. This does not work because in some source documents, `Content A` and `Content B` are reversed.

The solution is to change the search scope for `Content B`. You can do this by setting the `marking` property of `Content A`. The `marking` property specifies where Conversion Agent should place the reference points that determine the start and end of the search scope.

The default setting is `marking = full`, which means that Conversion Agent places reference points before and after each anchor. The search scope for `Content B` begins at the last reference point, which is the one following `Content A`. This leads to incorrect parsing, as we have seen.

To prevent Conversion Agent from placing reference points around `Content A`, set the `marking` property of `Content A` to `none`. As a result, the search scope for `Content B` starts at the end of the `Marker`. This allows Conversion Agent to find `Content B`, even if it precedes `Content A`.

The following table describes all four possible values of the `marking` property. The Result column assumes that you assign the `marking` value to `Content A` in the above example.

Marking Property	Explanation	Result
`full`	Conversion Agent places reference marks at the beginning and end of the current anchor. This is the default behavior.	Conversion Agent seeks the next anchor after the end of the current anchor. `Content B` follows `Content A`.
`begin position`	Conversion Agent places a reference mark only at the start of the current anchor.	Conversion Agent seeks the next anchor after the start of the current anchor. `Content B` overlaps or follows `Content A`.
`end position`	Conversion Agent places a reference mark only at the end of the current anchor.	Conversion Agent seeks the next anchor after the end of the current anchor. `Content B` follows `Content A`.
`none`	Conversion Agent does not place any reference marks at the current anchor.	Conversion Agent seeks the next anchor after the end of the preceding anchor. `Content B` follows `Marker`, without regard to `Content A`.

Note: There are a few circumstances where you must use an anchor that marks a reference point. An example is the separator of a `RepeatingGroup`. If the separator does not mark, it does nothing. Conversion Agent Studio displays a warning if you attempt to use a non-marking anchor in a location where marking is required.

Online Samples

For an online sample of the marking property, open the project `samples\Projects\Marking_Mode\Marking_Mode.cmw`. The sample uses the property to alter the search scope of a `Content` anchor.

For another example, see `samples\Projects\NonMarker\NonMarker.cmw`. This sample uses the `marking = none` option, permitting two `Content` anchors to overlap. The sample also illustrates the use of `direction = backward` to search from the end of the scope.

Adjusting the Search Criteria

Conversion Agent can search for anchors according to a large number of search criteria, for example:

- According to the delimiter locations, which Conversion Agent learns from the example source
- According to a positional offset, in other words, the number of characters from a reference point
- By searching for particular text
- By searching for a pattern or regular expression
- By searching for a specified data type
- By searching for an attribute value

You can combine these search criteria in almost any way. For example, you might specify that a `Content` anchor begins two tabs after a `Marker` anchor, and that it is 10 characters long. If you do this, you are using a delimiter criterion to define the beginning of the `Content` anchor, and an offset criterion to define the end.

The components that perform these searches are called searcher components. For more information, see “Searcher Component Reference” on page 94.

Using XSD Data Types to Narrow the Search Criteria

By default, in addition to the other search criteria, Conversion Agent searches for a `Content` anchor according to the XSD type of its data holder.

For example, suppose that the search scope of a `Content` anchor is the following string.

```
The students' grades were 81, 56, and 95, respectively.
```

Further suppose that you define no other search criteria for the anchor. If you map the anchor to a data holder that has a type of `xs:string`, the anchor retrieves the entire string.

If the data holder has a type of `xs:integer`, Conversion Agent searches for the first substring that matches the data type. Assuming that you configure the anchor with `direction = forward`, the anchor retrieves the integer `81`. If `direction = backward`, the anchor retrieves `95`.

Now suppose the data holder has a type of `xs:integer`, and the schema restricts the data holder to values less than `60`. Conversion Agent searches for an integer that conforms to the restriction and retrieves `56`.

XSD Types in Combination with Other Search Criteria

You can combine an XSD-type criterion with other search criteria. In the above example, suppose you configure the `Content` anchor to search for the following regular expression:

```
[",.*,"]
```

The expression searches for two commas, separated by any characters other than a newline. The search finds the substring

```
, 56,
```

If the XSD type of the data holder is `xs:integer`, the anchor retrieves `56`.

List Types

A data holder can have an XSD list type, which is a space-separated list. Conversion Agent filters the text retrieved by the `Content` anchor to match the XSD types of the list items.

Suppose that the schema defines an attribute called `grades`, which is a list of `xs:integer` items. If you map the above `Content` anchor to `grades`, the anchor returns a list of the integers in the string:

```
81 56 95
```

If the `grades` attribute belongs to an element called `Students`, the XML output is:

```
<Students grades="81 56 95" />
```

If you define the `Content` anchor with `direction = backward`, the list is reversed:

```
<Students grades="95 56 81" />
```

Decimal Type

If a data holder has the `xs:decimal` type, Conversion Agent assumes that the decimal separator is a period. If your locale setting uses a comma as the decimal separator, an `xs:decimal` search might fail.

Type Search with Closing Marker

If a `Content` anchor has a `closing_marker` property but does not have an `opening_marker`, Conversion Agent returns the substring closest to the `closing_marker` that matches the XSD type of the data holder.

In the above example, if you define the word `respectively` as the `closing_marker`, and the data holder has a type of `xs:integer`, the anchor retrieves `95`.

Online Sample

For an online example of searching by an XSD type, open the project `samples\Projects\Pattern\Pattern.cmw`. The sample is a parser containing a single `Content` anchor that is mapped to an XML element. The XSD schema uses an `xs:pattern` to restrict the element to certain character sequences. The anchor outputs the portion of the source document that matches the pattern.

Disabling the Data-Type Search

You can disable the data-type search by selecting the `disable_XSD_type_search` property of the `Content` anchor. If you do that, the anchor searches according to the other criteria, without regard to the XSD type of the data holder.

If the result does not have the proper type, it cannot be stored in the data holder and the anchor fails. You can use transformers to convert the result to the proper type and prevent the failure. For more information, see "Transformers" on page 101.

For example, suppose that the source document contains a date in the `dd-mm-yyyy` format, and you want to store the date in an `xs:date` data holder. You can handle this situation in the following way:

1. Define a `Content` anchor that retrieves the `dd-mm-yyyy` data, ignoring the mismatch with the `xs:date` type.
2. Configure the anchor with a `DateFormatICU` transformer that converts the result to `xs:date`.

Anchors that Contain Nested Anchors

An interesting question is how a parser searches for an anchor that has nested anchors, such as a `Group` anchor.

Conversion Agent does not search for a `Group`, and then search for the nested anchors. Rather, it searches for the nested anchors. The extent of the `Group` is defined by the nested anchors that Conversion Agent finds.

For example, suppose a parser has the following sequence of anchors. We assume that the anchors have default `phase`, `marking`, and `optional` properties.

```
Marker A
Group
  Marker B
  Content C
  Marker D
Marker E
```

Conversion Agent searches first for `Marker A` and `Marker E`. The search scope of the `Group` is the region between `Marker A` and `Marker E`.

Then, within the search scope of the `Group`, Conversion Agent searches for `Marker B` and `Marker D`. The region between these `Marker` anchors is the search scope for `Content C`.

Within the latter search scope, Conversion Agent searches for `Content C`.

You can view these relationships in the example pane of Conversion Agent Studio. The example pane highlights the nested anchors, helping you visualize the extent of the `Group`.

Anchor Quick Reference

The following table briefly describes the anchors that Conversion Agent supports. We have attempted to categorize the anchors according to their main characteristics or purpose. Within each category, the anchors are listed in alphabetical order.

Simple Anchors

The anchors in this category are used to define simple text elements in a document.

Anchor	Description
`Content`	Retrieves text from a specified location in a source document and stores the text in a data holder
`Marker`	Defines a reference point in the source text. The parser uses the reference point to search for other anchors.

Grouping Anchors

These anchors group a set of nested anchors together.

Anchor	Description
`DelimitedSections`	Defines sections of a document that are delimited by a separator.
`EnclosedGroup`	Defines a bounded segment of the source document.
`Group`	Binds a set of anchors together for processing as a unit.
`RepeatingGroup`	Parses a repetitive section of a document.

Other Anchors

Anchor	Description
`Alternatives`	Specifies alternative anchors that may exist at a particular location in a source document.
`EmbeddedParser`	Activates a secondary parser that runs on a segment of the source document.
`FindReplaceAnchor`	Marks text for replacement. Used with the `TransformByParser` transformer.
`HtmlForm`	Defines an HTML form. The anchor submits the form to a web server and runs a secondary parser on the server response.

Anchor Component Reference

This section describes the anchor components that are available in Conversion Agent.

Alternatives

The `Alternatives` anchor allows you to define a set of alternative, nested anchors. You can define a criterion for the alternative that the parser should accept. Only the accepted anchor affects the parser output. The other anchors, whether failed or successful, have no effect on the parser output.

Example

Suppose you are parsing a document in which a date can appear in either of the following patterns:

```
21/10/03
October 21, 2003
```

To process this content, you can define an `Alternatives` anchor that contains two `Content` anchors that store their output in different XML elements. Each XML element is constrained to accept one of the date patterns. The `Alternatives` anchor is configured with `selector = ScriptOrder`.

When the parser runs the `Alternatives` anchor, it tests the first `Content` anchor. If the date matches the pattern of the first anchor, the first `Content` anchor succeeds. If the date does not match the pattern, the first `Content` anchor fails, and the `Alternatives` anchor tests the second `Content` anchor. In this way, the parser can process both date patterns.

How to Define

Add an `Alternatives` anchor by editing the IntelliScript. Nested within the `Alternatives` anchor, add the alternative anchors.

Table 8-1. Basic Properties

Property	Description
`selector`	The criterion for deciding which alternative to accept. The options are: - `ScriptOrder`. Conversion Agent tests the nested anchors in the sequence that they are defined in the IntelliScript. It accepts the first nested anchor that succeeds. If all the nested anchors fail, the `Alternatives` anchor fails. - `DocumentOrder`. Conversion Agent tests all the nested anchors. It accepts either the `first` or `last` successful nested anchor, according to the locations of the anchors in the source document. If all the nested anchors fail, the `Alternatives` anchor fails. - `NameSwitch`. Conversion Agent searches for the nested anchor whose `name` property is specified in a data holder. It ignores the other nested anchors. If the named nested anchor fails, the `Alternatives` anchor fails.

Table 8-2. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Anchor Properties” on page 72.
`remark`	
`disabled`	
`optional`	
`marking`	
`phase`	

Using Alternatives to Select a Secondary Parser

You can use an `Alternatives` anchor to control which of several secondary parsers processes a document. The main parser can use this feature to process source documents of multiple types.

For example, suppose that the home page of a newspaper web site has links to articles. Following each link, the article is labeled `News`, `Business`, or `Sports`. You want to parse the articles, using a different parser for each type, like this:

```
<a href="PrincessWeds.html">Norwegian Princess Weds</a> - News
<a href="BanksMerge.html">Local Banks to Merge</a> - Business
<a href="HomeTeamWins.html">Bears Trounce Antelopes</a> - Sports
```

You can support this situation in the following way:

1. The main parser retrieves the filename of an article and stores it in a variable.
2. The main parser contains an `Alternatives` anchor that is configured with the `DocumentOrder` option.
3. The `Alternatives` anchor contains nested `Group` anchors.
4. Each `Group` anchor is configured with a `Marker` anchor and a `RunParser` action, as follows:
   - The first `Group` contains a `Marker` that searches for the string `News`. The `Group` is configured with a `RunParser` action that runs a secondary parser called `NewsParser`.
   - The second `Group` contains a `Marker` that searches for `Business` and runs `BusinessParser`.
   - The third `Group` contains a `Marker` that searches for the `Sports` and runs `SportsParser`.

The `Alternatives` anchor tests all three `Group` anchors. It accepts the `Group` containing the first `Marker` that occurs after the filename. The `Group` runs the appropriate parser on the file.

Online Sample

For an online sample of this anchor, open the project `samples\Projects\Alternatives\Alternatives.cmw`. The sample uses `Alternatives` anchors to parse different name and date formats that may exist in a source document.

Content

A `Content` anchor retrieves text from the source document. It stores the retrieved text in a data holder.

How to Define

You can create a `Content` anchor by working in either the example source or the IntelliScript. For more information, see "Defining Anchors" on page 69.

Table 8-3. Basic Properties

Property	Description
`opening_marker`	A searcher component labeling the start of a region, in which Conversion Agent should search for the Content anchor. Defining this property is similar to defining a `Group` containing a `Marker` followed by a `Content` anchor. The possible property values are `NewlineSearch`, `PatternSearch`, `OffsetSearch`, and `TextSearch`. For example, a `NewlineSearch` means that Conversion Agent should search for the anchor after a newline character. A `TextSearch` means to search after a specified text string. For more information, see the "Searcher Component Reference" on page 94.
`closing_marker`	A searcher component labeling the end of a region, in which Conversion Agent should search for the `Content` anchor. Defining this property is similar to defining a `Group` containing a `Content` anchor followed by a `Marker`. Defining both `opening_marker` and `closing_marker` is similar to defining a `Group` containing a `Marker Content Marker` sequence. The property values are the same as for `opening_marker`.
`value`	Specifies a searcher component that searches for the text retrieved by the `Content` anchor. The search is between `opening_marker` and `closing_marker`. If `opening_marker` is not defined, the search is between the surrounding reference points. For more information, see "How a Parser Searches for Anchors" on page 72. The options are: - `Empty`. The `Content` anchor retrieves the entire search scope. - `AttributeSearch`. The `Content` anchor retrieves the value from an expression of the type `AttributeName=...`. This is useful, for example, to retrieve attribute values from an XML or HTML source document. - `LearnByExample`. The parser learns what text to retrieve according to the parser format and the example source. For example, if the parser has a tab-delimited format, it counts the number of tabs from the start of the search scope to the example text. It retrieves the text between the corresponding tabs in the source document. - `PatternSearch`. The `Content` anchor retrieves the first text that matches a specified regular expression. - `TypeSearch`. The `Content` anchor retrieves the first text that matches a specified XSD data type. For more information about these options, see the "Searcher Component Reference" on page 94. In addition to the searcher components, Conversion Agent uses the XSD type of the `data_holder` as a search criterion. For more information, see "Using XSD Data Types to Narrow the Search Criteria" on page 76.
`data_holder`	A data holder where the anchor should store the retrieved text.

Table 8-4. Advanced Properties

Property	Description
allow_empty_values	If selected, the Content anchor can be empty. The data_holder is assigned an empty value. This can occur, for example, if the anchor is configured with value = LearnByExample and there is nothing between the delimiters. It can also occur if there is nothing between the opening_marker and the closing_marker. If allow_empty_values is not selected in these situations, the anchor fails.
disable_XSD_type_search	If not selected, the anchor searches for data within its search scope that matches the XSD type of the data holder. If selected, the anchor searches without regard to the XSD type. If the result, following the application of any transformers, does not have the proper type, the data cannot be stored in the data holder and the anchor fails. For more information, see "Using XSD Data Types to Narrow the Search Criteria" on page 76.
ignore_default_transformers	If selected, the anchor does not apply the default transformers to the content. For more information, see "Transformers" on page 101.
transformers	A sequence of transformers that Conversion Agent should apply to the retrieved text. For more information, "Transformers" on page 101.
name	For more information about these properties, see "Standard Anchor Properties" on page 72.
remark	
disabled	
optional	
on_fail	
direction	
marking	
phase	

Search Direction

The direction property has multiple effects in a Content anchor. If direction = backward:

- Conversion Agent searches backward from the end of the search scope for the opening_marker and closing_marker. Opening_marker still precedes closing_marker.
- The searcher component searches backward from the end of the search scope.
- If the searcher component is LearnByExample, it counts the delimiters backward from the end of the search scope.

Online Sample

For an online sample of Content anchors, open the project samples\Projects\Content\Content.cmw. The sample illustrates several uses of the opening_marker, closing_marker, and value properties to configure Content anchors.

DelimitedSections

The DelimitedSections anchor parses sectioned data that is delimited by a separator.

Within the DelimitedSections, nest other anchors. Each nested anchor is responsible for parsing a single section.

Example

An employee resume form contains several sections, each of which is preceded by a line of hyphens:

```
---------------------------
Jane Palmer
Employee ID 123456
---------------------------
Professional Experience
...
---------------------------
Education
...
```

You can define the sectioned region as a `DelimitedSections` anchor, with the line of hyphens as the `separator`. Because the line of hyphens precedes each section, define the `separator_position` as `before`.

Within the `DelimitedSections` anchor, nest three `Group` anchors. The first `Group` parses the `Jane Palmer` section, the second `Group` parses the `Professional Experience` section, and so forth.

Optional Sections

In the above example, suppose that the second section, `Professional Experience`, is missing from some source documents. Its separator, the line of hyphens, is always present.

```
---------------------------
Jane Palmer
Employee ID 123456
---------------------------
---------------------------
Education
...
```

To handle this situation, configure the `DelimitedSections` in the following way:

- In the second `Group` anchor, select the `optional` property. This means that if the `Group` fails, it does not cause the `DelimitedSections` to fail.
- In the `DelimitedSections` anchor, set `using_placeholders = always`. This means that the anchor looks for the separator of the optional section, even if the section itself is missing.

Now suppose that if the `Professional Experience` section is missing, its separator is also missing.

```
---------------------------
Jane Palmer
Employee ID 123456
---------------------------
Education
...
```

In this case, configure the `DelimitedSections` as follows:

- In the second `Group` anchor, select the `optional` property.
- In the `DelimitedSections` anchor, set `using_placeholders = never`. This means that the anchor should not look for the separator of a missing section.

How to Define

Add a `DelimitedSections` anchor by editing the IntelliScript. Nested with the `DelimitedSections` anchor, add a sequence of anchors that parse the sections.

Table 8-5. Basic Properties

Property	Description
`separator`	An anchor that delimits the sections. The anchor is typically a `Marker`.

Table 8-5. Basic Properties

Property	Description
`separator_position`	Position of the `separator` relative to the sections. The options are `before`, `after`, `between`, and `around`.
`using_placeholders`	Specifies whether the `DelimitedSections` should look for the separator of an optional section that is missing from the source document. The options are `always`, `never`, and `when necessary`.

The following table illustrates the possible values of the `separator_position` property. The examples assume that the `separator` is a vertical-line character (`|`).

separator_position	Explanation	Example
`before`	There is a separator before each section, including the first section.	`|1|2|3|4`
`after`	There is a separator after each section, including the last section.	`1|2|3|4|`
`between`	There is a separator between the successive sections, but not before the first section and not after the last section.	`1|2|3|4`
`around`	There are separators before and after each section, including the first and last sections.	`|1|2|3|4|`

The following table illustrates the possible values of the `using_placeholders` properties. The examples assume that the `separator_position` is `before` and that sections `2` and `4` are missing.

using_placeholders	Explanation	Example
`always`	The separator of a missing section always exists.	`|1||3|`
`never`	The separator of a missing section never exists.	`|1|3`
`when necessary`	The separator of a missing internal section always exists. The separator of a missing terminal section never exists.	`|1||3`

Table 8-6. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Anchor Properties” on page 72.
`remark`	
`disabled`	
`optional`	
`on_fail`	
`marking`	
`phase`	

Online Sample

For an online sample of this anchor, open the project `samples\Projects\DelimitedSections\DelimitedSections.cmw`. The sample illustrates a `DelimitedSections` anchor that parses sections separated by a `|` symbol. Each section is parsed by a single `Content` anchor.

EmbeddedParser

The `EmbeddedParser` anchor uses a secondary parser to parse its search scope.

It is permitted for a parser to call itself recursively.

Example

A document is tab-delimited, except for one section that is comma-delimited.

To parse the document, you can define a main parser that uses the `TabDelimited` format. Define another parser that uses the `CommaDelimited` format. Use an `EmbeddedParser` anchor to run the second parser within the execution of the first parser.

How to Define

You can define an `EmbeddedParser` by editing the IntelliScript.

Table 8-7. Basic Properties

Property	Description
`parser`	The name of the secondary parser, which must be defined in the same project.
`schema_connections`	Connects the output of the secondary parser to the output of the main parser. The property contains a list of `Connect` subcomponents that define the relation between data holders in the output of the two parsers. For more information, see “Connect” on page 99.

Table 8-8. Advanced Properties

Property	Description
`source_transformers`	A sequence of transformers that the parser applies to the search scope before the secondary parser processes it.
`name`	For more information about these properties, see “Standard Anchor Properties” on page 72.
`remark`	
`disabled`	
`optional`	
`on_fail`	
`marking`	
`phase`	

Online Sample

For an online sample of this anchor, open the project `samples\Projects\EmbeddedParser\EmbeddedParser.cmw`. The sample uses a main parser to determine the location of an address. It then runs an `EmbeddedParser` to parse the address.

EnclosedGroup

The `EnclosedGroup` anchor defines a bounded region that contains nested anchors.

The boundaries are specified by opening and closing anchors. In the case of nested boundaries, such as parentheses or HTML tags, the `EnclosedGroup` finds the matching boundaries.

An `EnclosedGroup` is similar to a `Content` anchor with an `opening_marker` and `closing_marker`. However:

- The `Content` anchor retrieves the entire content between the opening and closing, without further parsing.
- The `EnclosedGroup` allows you to further parse the content between the opening and closing.

Example

You can define an HTML table as an `EnclosedGroup`, with the `<table>` and `</table>` tags as the opening and closing. The nested anchors parse the content of the table.

Suppose the <table> element contains a nested <table> element. In other words, a table is nested within a table cell. The EnclosedGroup anchor matches the parent <table> tag with the parent </table> tag. It does not match the parent <table> tag with the nested </table> tag, which would be a misidentification of the table.

How to Define

You can define an EnclosedGroup anchor by editing the IntelliScript. Add the nested anchors that parse the content.

Table 8-9. Basic Properties

Property	Description
opening	The opening anchor of the EnclosedGroup, typically a Marker anchor.
closing	The closing anchor of the EnclosedGroup, typically a Marker anchor.

Table 8-10. Advanced Properties

Property	Description
source target	These properties are useful in situations where the anchor must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.
name	For more information about these properties, see “Standard Anchor Properties” on page 72.
remark	
disabled	
optional	
on_fail	
marking	
phase	
no_initial_phase	

FindReplaceAnchor

This anchor is intended for use within a parser that is activated by the TransformByParser transformer. The anchor marks text in the source, and it specifies a replacement for the text. When the parsing is done, the TransformByParser transformer uses the markings to modify the text.

FindReplaceAnchor identifies the text to replace in the following way:

- If FindReplaceAnchor does not contain any nested anchors, it replaces the complete text within its search scope. For example, if FindReplaceAnchor is between two Marker anchors, it marks the text between them.
- If FindReplaceAnchor contains nested anchors, it replaces the text spanned by the nested anchors. For example, if FindReplaceAnchor contain a Marker, it replaces the Marker. If it contains two Marker anchors, it replaces the segment from the first Marker to the second, including the Marker anchors themselves.

You can configure the anchor with a static replacement string or with a string that the parser retrieves dynamically from the source document.

For more information, see “TransformByParser” on page 127.

Example

You have a text document, to which you want to add line numbers. You can add the line numbers by the following approach:

1. Create a parser, and add a `RepeatingGroup` to it.
2. Within the `RepeatingGroup`, add a `FindReplaceAnchor`.
3. Within the `FindReplaceAnchor`, add a `Marker` anchor, and set its `search` property to `NewlineSearch`.

   This causes the `FindReplaceAnchor` to mark every newline in the document.
4. Configure the `RepeatingGroup` to store its `current_iteration` in a variable. Set the `replace_with` property of the `FindReplaceAnchor` to the variable.
5. At the global level of the IntelliScript, define a `TransformByParser` transformer. Set its `parser` property to the parser.
6. Set the `TransformByParser` as the startup component of the transformation.

   The transformer outputs a modified version of the original file, containing line numbers.

How to Define

You can define a `FindReplaceAnchor` by editing the IntelliScript. If required, add nested anchors marking a substring to be replaced.

Table 8-11. Basic Properties

Property	Description
`replace_with`	Type the replacement string, or browse to a data holder that contains the text.

Table 8-12. Advanced Properties

Property	Description
`on_partial_match`	If the `FindReplaceAnchor` does not find all its nested, non-optional anchors, and `on_partial_match` has a value of `fail`, the `FindReplaceAnchor` fails. If `on_partial_match` has a value of `skip`, the `FindReplaceAnchor` removes the area spanned by the successful nested anchors from its search scope and tries to find all the nested anchors again. It iterates this procedure until it finds the anchors, as long as there is a partial match.
`source` `target`	These properties are useful in situations where the anchor must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.
`name`	For more information about these properties, see “Standard Anchor Properties” on page 72.
`remark`	
`disabled`	
`optional`	
`on_fail`	
`marking`	
`phase`	
`no_initial_phase`	

Group

The `Group` anchor binds a sequence of anchors and actions together. It allows you to apply properties to all the nested components, together.

For example, a `Group` allows you to define operations that Conversion Agent should perform on a set of anchors or to control the phase of the nested anchors.

How to Define

You can define a Group by editing the IntelliScript. Add nested anchors, and optionally actions, that parse the content of the Group.

Optional Group

You can use the optional property of a Group to prevent Conversion Agent from attempting to retrieve text from a missing section of a document.

For example, to parse the source

```
First name: Ron
```

you might define First name: as a Marker and Ron as Content. If some source documents do not contain the first-name data, you can put the Marker and Content in a Group and make it optional. If First name: is not found, the Group immediately fails. The parser does not search for the Content anchor.

There is a difference between making the Group optional and making its nested anchors optional. If you make both the Marker and Content optional, instead of the Group, Conversion Agent ignores the Marker failure and searches for the Content. This might result in retrieving irrelevant text.

Table 8-13. Advanced Properties

Property	Description
absent	If selected, the Group succeeds only if one of its nested, non-optional anchors or actions fails. You can use this feature to test for the absence of nested anchors.
on_partial_match	If the Group does not find all its nested, non-optional anchors, and on_partial_match has a value of fail, the Group fails. If on_partial_match has a value of skip, the Group removes the area spanned by the successful nested anchors from its search scope and tries to find all the nested anchors again. It iterates this procedure until it finds the nested anchors, as long as there is a partial match.
search_order	The order in which to process the nested anchors. The options are: - top-down. The nested anchors are processed in the sequence that is defined in the IntelliScript. - bottom-up. The nested anchors are processed in reverse order. This is useful if data from a later anchor affects how you process an earlier anchor.
source target	These properties are useful in situations where the anchor must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.
name	For more information about these properties, see “Standard Anchor Properties” on page 72.
remark	
disabled	
optional	
on_fail	
marking	
phase	
no_initial_phase	

Online Sample

For an online sample of this anchor, open the project samples\Projects\persistent_search\persistent_search.cmw.

The sample illustrates a Group that is configured with the on_partial_match = skip property. The Group contains two Marker anchors:

- The first `Marker` searches for the text `A`.
- The second `Marker` searches for a string containing any number of `*` characters. It has the `adjacent` property, which means that it must be adjacent to the first `Marker`.

On the first pass, the `Group` finds an `A` character at the beginning of the source document. It does not find the second `Marker` adjacent to the `A` character, however.

The `Group` reduces its search scope by eliminating the first `A` character, and searches again for the two adjacent `Marker` anchors. It continues this procedure until it successfully finds a string `A*`, which contains the adjacent `Marker` anchors.

You can observe the behavior in the event log. The log records that the `Group` fails on the first two trials and succeeds on the third.

Try experimenting with the `on_partial_match` and `adjacent` settings. You can see the effect in the color coding of the example source.

You can also try running the sample, although the result file is empty because the parser does not contain `Content` anchors. If you set `on_partial_match = fail`, you can observe in the event log that the parser fails, because the `Group` cannot find the adjacent anchors.

HtmlForm

`HtmlForm` is an anchor that marks a `<form>` element in an HTML source document. It submits the form to a web server specified in the `action` attribute of the form. It then activates a secondary parser that parses the server response.

Within the `field_filters` property of `HtmlForm`, you can modify the fields and values of the form. The `HtmlForm` anchor collects the possible values of the form fields, combines them, and submits all the combinations. You can distribute the submissions over multiple computers.

`HtmlForm` appends the parsed output of the web-server responses to the main parser output.

How to Define

You can define an `HtmlForm` by editing the IntelliScript. The following are some tips that can help you configure the anchor with a minimum of effort:

1. Prepare and run the anchor without any filters.
2. In the example pane, confirm that the `HtmlForm` anchor highlights the correct form.
3. Examine the `Results\_HtmlForm.xml` file, which contains the form data that was submitted. Confirm that the anchor included all the fields.
4. Add filters or adjust the fields as required.

Table 8-14. Basic Properties

Property	Description
`next_parser`	The name of the secondary parser that parses the server response.
`field_filters`	Adds fields and their values to the form data that the anchor submits. Specify a sequence of `AddField`, `ModifyField`, and `RemoveField` subcomponents that generate the desired fields. Be sure to use the same field names as in the original HTML form. For more information, see the “Anchor Subcomponent Reference” on page 98.
`click`	Specifies the HTML element that a simulated user clicks to submit the form. The options are `ImageClick` and `SubmitClick`. For more information, see the “Anchor Subcomponent Reference” on page 98.

Table 8-15. Advanced Properties

Property	Description
part_to_submit	Select the portion of the possible field-value combinations to submit. The options are SubmitAll, SegmentIndex, and SegmentSize. For more information, see the.“Anchor Subcomponent Reference” on page 98.
retries	The number of retries, if the anchor cannot connect to the web server on the first attempt.
seconds_to_wait	The interval in seconds between retries.
js_function	The name of a JavaScript function that exists in the source document. The anchor calls the function before it submits the form.
js_params	A list of data holders containing parameters of js_function. The parameters must be in the same order as in the function declaration.
name	For more information about these properties, see “Standard Anchor Properties” on page 72.
remark	
disabled	
optional	

Marker

A Marker defines a location in a source document. It is used as a reference point, from which Conversion Agent searches for the succeeding anchors.

By default, the phase property of a Marker is initial, which means that Conversion Agent scans a document for Marker anchors before it searches for Content anchors. For more information, see “How a Parser Searches for Anchors” on page 72.

How to Define

You can define a Marker by the select-and-click method or by editing the IntelliScript. For more information, see “Defining Anchors” on page 69.

Table 8-16. Basic Properties

Property	Description
search	Defines the search criteria for the Marker. The search criteria determine where the Marker is located within the search scope. For example, a NewlineSearch locates the Marker at a newline character. A TextSearch locates the Marker at a specified string. For more information, see “How a Parser Searches for Anchors” on page 72. The value of this property is one of the following searcher components.: - NewlineSearch. Searches for a newline character. - TextSearch. Searches for a predefined text string or for a text string that is stored in a data holder. - PatternSearch. Searches for a string that matches a specified regular expression. - OffsetSearch. Skips a predefined number of characters following the preceding reference point, or a number of characters that is stored in a data holder. The Marker is the point following the skipped characters. - TypeSearch. Searches for a string that conforms to a specified XSD data type. For more information, see the “Searcher Component Reference” on page 94.

Table 8-17. Advanced Properties

Property	Description
adjacent	If selected, the Marker must be adjacent to the anchor at the beginning of its search scope. If direction = backward, it must be adjacent to the anchor at the end of its search scope. If not selected, Conversion Agent can skip over text until if finds the Marker.
absent	If selected, the Marker is a test that the specified text or pattern is absent from the document. If Conversion Agent finds the Marker, the Marker fails.
count	The occurrence number to find. For example, to set the Marker at the second newline following the preceding anchor, set search = NewlineSearch and count = 2.
name	For more information about these properties, see “Standard Anchor Properties” on page 72.
remark	
disabled	
optional	
on_fail	
direction	
marking	
phase	

Online Sample

In the Online Samples folder, open Projects\Markers\Markers.cmw. The sample demonstrates Marker anchors that search for:

- A predefined text string
- A newline character
- An offset
- A data type
- A regular expression

If you run the parser, note that the result file is empty because the configuration does not have any Content anchors.

RepeatingGroup

The RepeatingGroup anchor parses a repetitive region of a source document.

The repeating units are called iterations. The iterations are typically delimited by a separator. The RepeatingGroup contains a sequence of nested anchors and actions that parse each iteration.

The RepeatingGroup anchor treats all iterations in the same way. To parse a semi-repetitive region containing sections that require differing treatment, you can use a DelimitedSections anchor, instead.

How to Define

You can define a RepeatingGroup by editing the IntelliScript. Add the nested anchors, and optionally actions, that parse each iteration of the RepeatingGroup.

Search for Iterations

By default, a RepeatingGroup searches for iterations from the beginning to the end of its search scope. For more information, see “How a Parser Searches for Anchors” on page 72.

Optionally, you can set the iteration_order property for a reverse search.

In each iteration:

- If the RepeatingGroup is configured with a separator, it searches for the next separator. Then, it searches for the anchors lying between a pair of separators.
- If the RepeatingGroup is not configured with a separator, it searches only for the anchors.

End of a RepeatingGroup

You can signal the end of a RepeatingGroup in ways such as the following:

- The RepeatingGroup can continue until the end of the document.
- You can insert a Marker after the RepeatingGroup. By default, the Marker is in an earlier search phase than the RepeatingGroup. This causes the parser to search for the Marker first and use it to limit the search scope of the RepeatingGroup. For more information, see "Adjusting the Search Phase" on page 74.
- You can set the count property, limiting the search to a maximum number of iterations.
- If the RepeatingGroup does not have a separator, it ends when the parser cannot find any more iterations.

Success or Failure of a RepeatingGroup

If a RepeatingGroup cannot find the non-optional anchors in an iteration, the iteration fails.

When an iteration fails, the RepeatingGroup can either end, fail, or skip the failed iteration. The behavior is as follows:

- If the RepeatingGroup does not have a separator, the RepeatingGroup ends. Provided that there was at least one successful iteration prior to the failed iteration, the RepeatingGroup succeeds.
- If the RepeatingGroup has a separator, and the skip_failed_iterations property is not selected, the RepeatingGroup fails.
- If the RepeatingGroup has a separator, and the skip_failed_iterations property is selected, Conversion Agent skips over the failed iteration and proceeds with the next iteration. Provided that at least one iteration succeeds, the RepeatingGroup succeeds.

Event Log of a RepeatingGroup

The Conversion Agent event log records events for every iteration of a RepeatingGroup.

If the skip_failed_iterations property is selected, the RepeatingGroup might generate an optional failure event (icon) following the successful iterations. A failure event (icon) might be nested within the optional failure. These events occur because the RepeatingGroup cannot find additional iterations to parse. The events are normal and not a cause for concern.

For example, if the source document contains two iterations, the log may display an optional failure and a failure because the RepeatingGroup cannot find a third instance of its separator.

Table 8-18. Basic Properties

Property	Description
separator	An anchor, typically a Marker, that delimits the iterations. If you leave this property empty, the RepeatingGroup does not look for a delimiter between the iterations. Instead, it assumes that an iteration is finished when it has found all the nested anchors. It then starts to parse the next iteration from the top of the nested anchor sequence. You can build a complex separator by inserting a Group in the separator property instead of a Marker.
separator_position	Position of the separator relative to the iterations of the RepeatingGroup. The options are before, after, between, and around.

The following table illustrates the possible values of the `separator_position` property. The examples assume that the `separator` is a vertical-line character (|).

Separator_position	Explanation	Example
`before`	There is a separator before each iteration, including the first iteration.	`|1|2|3`
`after`	There is a separator after each iteration, including the last iteration.	`1|2|3|`
`between`	There is a separator between the successive iterations, but not before the first iteration and not after the last iteration.	`1|2|3`
`around`	There are separators before and after each iteration, including the first and last iterations.	`|1|2|3|`

Table 8-19. Advanced Properties

Property	Description
`skip_failed_iterations`	This option has an effect only if the `RepeatingGroup` has a `separator`. By default, this option is selected. This means that the `RepeatingGroup` skips over a failed iteration and proceeds with the next iteration. Provided that at least one iteration succeeds, the `RepeatingGroup` succeeds. If you deselect the option, the `RepeatingGroup` fails if any iteration fails.
`search_order`	The order in which to process the nested anchors within each iteration. The options are: - `top-down`. The nested anchors are processed in the sequence that is defined in the IntelliScript. - `bottom-up`. The nested anchors are processed in reverse order. This is useful if data from a later anchor affects how you process an earlier anchor.
`iteration_order`	The order in which to process the iterations. The options are the same as for `search_order`, but apply to the iterations rather than to the anchors within an iteration.
`count`	The number of iterations to run. Enter a number, or browse to a data holder that contains the number. If blank, the iterations continue until the search scope is exhausted. If `count = 0`, the `RepeatingGroup` does not search for iterations. In this case, the `RepeatingGroup` succeeds, but it does not produce any output.
`current_iteration`	A data holder where the `RepeatingGroup` outputs the number of the current iteration.
`on_partial_match`	This option controls the behavior if, in a particular iteration, the `RepeatingGroup` finds some but not all of its nested, non-optional anchors. In such a case, if `on_partial_match` has a value of `fail`, the iteration fails. If `on_partial_match` has a value of `skip`, the `RepeatingGroup` removes the area spanned by the successful nested anchors from its search scope and tries to find all the nested anchors again. The removal-retry procedure is repeated until the iteration succeeds, or until there is no longer a partial match. In the latter case, the iteration fails.
`source` `target`	These properties are useful in situations where the anchor must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.
`on_iteration_fail`	If an iteration fails, writes an entry in the user log. User the `on_fail` property to write an entry if the entire `RepeatingGroup` fails. Use `on_iteration_fail` to write an entry if a single iteration fails. For more information, see “Failure Handling” on page 226.
`name`	For more information about these properties, see “Standard Anchor Properties” on page 72.
`remark`	
`disabled`	
`optional`	

Table 8-19. Advanced Properties

Property	Description
on_fail	
no_initial_phase	
phase	
marking	

Online Samples

For an online example of this anchor, open the project samples\Projects\Dynamic_And_RepeatingGroup\Dynamic_And_RepeatingGroup.cmw. The sample uses a RepeatingGroup to iterate over the lines of a document.

Some lines of the source document contain a parenthesized footnote reference, such as "(1)". The RepeatingGroup contains a Group, whose purpose is to parse the footnote and insert its content in the output.

The Group contains a Content anchor that retrieves the footnote reference and stores it in a variable. The Group then activates a RunParser action that activates a secondary parser. The secondary parser finds the footnote referenced by the variable, parses it, and inserts the result in the output.

Searcher Component Reference

This section documents the searcher components that find text in a source document. The searcher components are used in many locations throughout Conversion Agent, for example:

- To define the location of anchors. For more information, see “Anchor Component Reference” on page 79.
- To define delimiter characters or strings. For more information, see “Formats” on page 39.
- To define the find_what string of a Replace transformer. For more information, see “Transformers” on page 101.

AttributeSearch

This component searches a source document for a specified attribute. The attribute must occur in an expression of the type:

```
AttributeName = value
```

or

```
AttributeName = "value"
```

The component retrieves the value.

The component is a possible setting of the value property, which belongs to the Content anchor. For more information, see “Content” on page 81.

Example

An HTML document contains the element:

```
<img src='MyPicture.gif'>
```

You can use AttributeSearch to retrieve the value of the src attribute. It returns the text MyPicture.gif.

Supported Attribute Syntax

AttributeSearch supports attribute strings containing an equals sign. Optionally, the equals sign can be surrounded by spaces. The attribute can be surrounded by double quotes, single quotes, or no quotes.

For example, suppose that AttributeSearch is configured to search for an attribute called time. In all of the following examples, it returns the same value, 12:55:33.

```
time = 12:55:33
time=12:55:33
time = '12:55:33'
time='12:55:33'
time = "12:55:33"
time="12:55:33"
```

Table 8-20. Basic Properties

Property	Description
att	The attribute name.

Table 8-21. Advanced Properties

Property	Description
match_case	If selected, AttributeSearch considers the attribute name to be case sensitive.

Online Sample

For an online sample of this component, open the project samples\Projects\Content\Content.cmw. The sample illustrates the use of an AttributeSearch to parse a text document that has a variable = value structure.

LearnByExample

This component learns how to search for text by examining the text location in the example source document. It uses the parser format to interpret the source document.

For example, if the parser has a tab-delimited format, LearnByExample counts the number of tabs from the search start to the example text. It searches for text in the source document that lies at the same number of tabs from the start of the search scope.

The component is a possible settings of the value property, which belongs to the Content anchor. For more information, see “Content” on page 81.

If the Content anchor is configured with direction = backward, the component counts the delimiters from the end of the search scope.

Table 8-22. Basic Properties

Property	Description
example	The text in the example source document at the anchor location.

NewlineSearch

This component searches for a newline, a linefeed character, a carriage return character, or both.

Anchors can use NewlineSearch to find newline markers. A Delimiter component can use NewlineSearch to find newline delimiters.

OffsetSearch

This component defines the number of characters between a reference point and an anchor. For example, it can define the number of characters between the end of a `Marker` and the start of a `Content` anchor.

Table 8-23. Basic Properties

Property	Description
`offset`	The number of characters between the reference point and the anchor. In some locations where `OffsetSearch` is used, such as in a `Marker` anchor, Conversion Agent Studio displays a browse button next to the `offset` property. You can enter a value or browse to a data holder containing the value.

Table 8-24. Advanced Properties

Property	Description
`allow_smaller_offset`	If the offset is beyond the search scope, this property allows a smaller offset. This is useful, for example, to permit a truncated field size at the end of a document.

PatternSearch

This component searches for a string that conforms to a regular expression.

Regular expressions are a way to define a text search criterion, similar to a wildcard search, but with greatly enhanced syntax. For more information about the regular expression syntax that Conversion Agent supports, see “RegularExpression” on page 121.

Anchors can use `PatternSearch` to find markers or content. The `Delimiter` component can use `PatternSearch` to find delimiters. The `Replace` transformer can use `PatternSearch` to find the text to be replaced.

Example

Suppose you want to define the string `%%%`, containing one or more `%` symbols, as a delimiter. Within the `Delimiter` component, you can use `PatternSearch` with the following regular expression:

```
%+
```

In another example, suppose you want to define a comma and a semicolon as alternative delimiters, at the same level of the delimiter hierarchy. You can use the following regular expression:

```
[,;]
```

Table 8-25. Basic Properties

Property	Description
`pattern`	The regular expression.

Table 8-26. Advanced Properties

Property	Description
`escape_sequence`	A prefix in the source document, such as a backslash character `\`, that causes the search component to ignore an instance of the pattern.

SegmentSearch

This component searches for opening and closing markers in a text string. It returns the segment from the opening marker to the closing marker, including the markers themselves.

The component is used in the `Replace` transformer to find text that is to be replaced.

Table 8-27. Basic Properties

Property	Description
`Opening`	The search criterion for the opening marker. The options are searcher components: `TextSearch`, `PatternSearch`, `NewlineSearch`, or `OffsetSearch`.
`Closing`	The search criterion for the closing marker.

TextSearch

This component searches for an explicit string.

Anchors can use `TextSearch` to find markers. The `Delimiter` component can use `TextSearch` to find delimiters. The `Replace` transformer can use `TextSearch` to find text that is to be replaced.

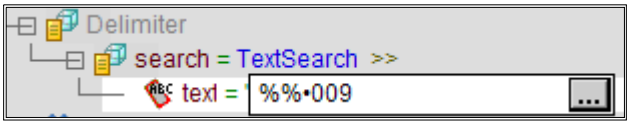
Example

To define the string percent-percent-tab as a delimiter, create a `Delimiter` component and set its `search` property to `TextSearch`. In the `TextSearch/text` property, type:

```
%%
```

Then press `Ctrl+a`, and type the ASCII code of a tab character:

```
009
```



Specifying a Search String Dynamically

In some of the locations where `TextSearch` is used, such as in a `Delimiter` component or a `Marker` anchor, Conversion Agent Studio displays a browse button to the right of the text box. Clicking the button display a Schema view. In the Schema view, you can select a data holder that contains the search text.

For example, suppose that you want to find repeated instances of the first word in a document. You can define a `Content` anchor that retrieves the first word and stores it in a variable. You can then define `Marker` anchors that use `TextSearch` to find other instances of the word that you stored in the variable.

Table 8-28. Basic Properties

Properties	Description
`text`	The string to find. In locations where dynamic search is supported, you can browse to a data holder containing the string. To type control characters, press Ctrl+a and enter their ASCII codes. The IntelliScript displays a tab as «. Other special characters appear as ASCII codes prefixed with a dot.

Table 8-29. Advanced Properties

Properties	Description
match_case	If selected, text is required to match the text property exactly, with the same upper and lower-case letter.
escape_sequence	A prefix in the source document, such as a backslash character \, that causes the search to ignore an instance of the string. In locations where dynamic search is supported, you can browse to a data holder containing the escape sequence.

Online Sample

For an online sample of this component, open the project samples\Projects\Dynamic_And_RepeatingGroup\Dynamic_And_RepeatingGroup.cmw.

In the GetRemarkParser component of this sample, a Marker anchor uses a dynamically defined TextSearch to find a footnote at the end of the source document. For more information about this sample, see "RepeatingGroup" on page 91.

TypeSearch

This component searches for an anchor of a specified XSD data type.

The component is a possible settings of the value property, which belongs to the Content anchor. For more information, "Content" on page 81.

Table 8-30. Basic Properties

Property	Description
val_type	The XSD data type.

Anchor Subcomponent Reference

This section describes subcomponents that you can assign as the values of certain anchor properties.

AddField

This is an option of the field_filters property of HtmlForm. It adds a field to be submitted with an HTML form.

Optionally, you can define multiple values. The HtmlForm anchor submits all possible combinations of the values with the values of other fields.

Table 8-31. Basic Properties

Property	Description
field_name	Name of the field.
filter	The way to generate the field values. The options are: UseDataHolder. Assigns a value that is contained in a data holder. To assign multiple values, use a multiple-occurrence data holder. UseValues. Assigns one or more explicit values. The ExcludeValues option is not in use.

Connect

This component specifies a correspondence between two data holders. The two data holders must have the same XSD data type.

`Connect` is used in the `EmbeddedParser` anchor to specify where a secondary parser should store its result in the output of the main parser. It is used in `EmbeddedSerializer` to specify how the input data holders of a secondary serializer are related to the input data holders of the main serializer. It is used in `EmbeddedMapper` for a similar purpose, on both the input and output data holders.

Example

A secondary parser outputs an XML element called `ID`. You want the main parser to store this result in a variable called `VarID`. You can connect `ID` to `VarID`.

For an additional example, see “EmbeddedSerializer” on page 174.

Table 8-32. Basic Properties

Property	Description
`data_holder`	A data holder that is referenced in the main parser or serializer.
`embedded_data_holder`	A data holder that is referenced in the secondary parser or serializer.

ImageClick

This subcomponent submits a form by simulating a user who clicks an image in the HTML form. The subcomponent is a possible value of the `click` property in the `HtmlForm` anchor.

The `pixel_x` and `pixel_y` properties are useful if the image has an area map. The properties indicate the location in the image where the user clicked.

Table 8-33. Basic Properties

Property	Description
`image_name`	The `name` attribute of the image that is specified in the HTML code.
`pixel_x`	The x-coordinate where the user clicked, measured in pixels from the left edge.
`pixel_y`	The y-coordinate where the user clicked, measured in pixels from the top edge.

ModifyField

This is an option of the `field_filters` property of `HtmlForm`. It modifies the value of a field that is defined in the HTML code of a form.

Optionally, you can define multiple values. The `HtmlForm` anchor submits all possible combinations of the values with the values of other fields.

Table 8-34. Basic Properties

Property	Description
`field_name`	Name of the field.
`filter`	The way to generate the field values. The options are: - `ExcludeValues`. Removes previously defined values. - `UseDataHolder`. Assigns a value that is contained in a data holder. To assign multiple values, use a multiple-occurrence data holder. - `UseValues`. Assigns one or more explicit values.

RemoveField

This is an option of the `field_filters` property of `HtmlForm`. It removes a field that is defined in the HTML code of a form.

Table 8-35. Basic Properties

Property	Description
`field_name`	Name of the field.

SegmentIndex

This is an option of the `part_to_submit` property of `HtmlForm`. It is used to distribute the form submissions between several computers.

`SegmentIndex` divides the set of field-value combinations into a specified number of portions, and it specifies which portion to submit. On another computer, you can configure a `SegmentIndex` that submits a different portion.

Table 8-36. Basic Properties

Property	Description
`parts`	The number of portions into which the combinations should be divided.
`selected_part`	The portion to submit. 1 means the first portion, and so forth.

SegmentSize

This is an option of the `part_to_submit` property of `HtmlForm`. It is used to distribute the form submissions between several computers.

`SegmentSize` divides the set of field-value combinations into portions of a specified size, and it specifies which portion to submit. On another computer, you can configure a `SegmentSize` that submits a different portion.

Table 8-37. Basic Properties

Property	Description
`part_size`	The number of combinations in each portion, by default, 2.
`selected_part`	The portion to submit. 1 means the first portion, and so forth.

SubmitAll

This is an option of the `part_to_submit` property of `HtmlForm`. It submits all combinations of the field values from the same computer.

SubmitClick

This component submits a form by simulating a user who clicks a submit button.

The component is a possible value of the `click` property of the `HtmlForm` anchor.

Table 8-38. Basic Properties

Property	Description
`submit_name`	The name of the submit button.

CHAPTER 9

Transformers

This chapter includes the following topics:

Overview

Transformers are components that modify data.

You can use transformers within components such as anchors, serialization anchors, and actions. The transformers modify the output of the components. For example, if you use a transformer within a `Content` anchor, it modifies the data that the anchor extracts from the source document.

You can also use transformers as document processors or as stand-alone, runnable components. In those cases, the transformers modify the complete content of a document.

You can use the transformers supplied with Conversion Agent, or you can define custom transformers.

This chapter explains how to use transformers and provides detailed information on the transformers available in Conversion Agent.

Defining Transformers

You can define transformers in the following locations of the IntelliScript:

- In the `transformers` property of an anchor or a serialization anchor
- In the `default_transformers` property of a format or of a serializer
- In the `ProcessByTransformers` document processor
- In the `transformers` property of certain actions
- At the global level, as a standalone, runnable component that modifies a source document.

The following sections explain the use of transformers in each of these locations.

Using Transformers in Anchors

You can use transformers in an anchor that creates XML output, such as `Content`. In the IntelliScript, nest the transformer components within the `transformers` property of the anchor.

The input of a transformer is the raw output of the anchor, before the anchor inserts the output in a data holder.

For example, suppose you are parsing the following source document:

```
First name: Ron
Last name: Lehrer
```

You want to create XML output in `ALL CAPS`, like this:

```
<Person>
  <FirstName>RON</FirstName>
  <LastName>LEHRER</LastName>
</Person>
```

To do this, you can configure the `Content` anchors, which retrieve the strings `Ron` and `Lehrer`, with the `ChangeCase` transformer.

Sequences of Transformers

You can configure an anchor with a sequence of transformers. Each transformer modifies the output of the preceding transformer.

In the `Ron Lehrer` example, suppose you want the following output:

```
<Person>
  <FirstName>- RON -</FirstName>
  <LastName>- LEHRER -</LastName>
</Person>
```

To do this, you might configure the `Content` anchors with the `ChangeCase` and `AddString` transformers. The transformers change the case and add the hyphens, in sequence.

Default Transformers

Very often, you want the same transformers to run on all the `Content` anchors in a parser. You can configure the format component of the parser with default transformers. This saves you the trouble of adding the same transformers to every anchor in the parser.

To do this, nest the transformers in the `default_transformers` property of the format. For more information, see “Formats” on page 39.

Many of the predefined format components include default transformers. For example, the `HtmlFormat` component has default transformers that remove HTML tags from the output and convert HTML entities to plain text. You can change the default transformers by editing the `default_transformers` property.

If an anchor has its own transformers, they run after the default transformers.

You can cancel the default transformers for particular anchors. To do this, set the `ignore_default_transformers` property of the anchor.

Using Transformers as Document Processors

You can run a transformer or a sequence of transformers as a document processor.

For example, you might run the `RemoveTags` transformer as a processor on an HTML document. The transformer removes the HTML tags before a parser starts to search for anchors in the document.

To do this, configure the parser format component with the `ProcessByTransformers` document processor, and nest the transformers within the component.

Using Transformers in Serialization Anchors

You can use transformers in serialization anchors that write to the output document, such as `ContentSerializer`. The transformers modify the data before the serializer writes it to the document.

For example, a `ContentSerializer` might write the content of a data holder called `DoctorName` to an output document. You might configure the `ContentSerializer` with an `AddString` transformer that adds the prefix `"Dr. "` to the content. Suppose the XML input has the following form:

```
<DoctorName>Albert Schweitzer</DoctorName>
```

The transformer modifies the content, resulting in the following output:

```
Dr. Albert Schweitzer
```

You can add transformers to the `default_transformers` property of a serializer. The transformers that you add here run in all the `ContentSerializer` serialization anchors before they write to the output document.

Using Transformers in Actions

Certain actions, such as `SetValue` and `Map`, let you apply transformers to their output. For more information, see "Actions" on page 133.

Using Global Transformers as Runnable Components

You can define a transformer at the global (top) level of the IntelliScript. You can then run the transformer as a standalone component that modifies a source document. The transformer runs as the startup component, not within a parser or a serializer.

For more information about defining global components, see *Using Conversion Agent Studio in Eclipse*.

To run a globally-defined transformer in the Studio:

1. Set the transformer as the startup component.
2. Click Run> Run.

   You are prompted to select the source document that the transformer should process.

   The Events view appears, for you can review the events.

   The output file is stored in the `Results` folder of the project. It has a name such as `Transformation of` *`filename`*`.txt`, where *`filename`* is the source file. You can open the file in any suitable application.

Standard Transformer Properties

In this section, we review certain properties that are found in many transformers. Many transformers have additional properties. For more information about the properties, see the "Transformer Component Reference" on page 106.

Property	Definition
`name`	A name that you assign to the transformer. Conversion Agent displays the name in the event log. This can help you find an event that was caused by the particular transformer.
`remark`	A comment describing the transformer.

Property	Definition
`disabled`	If selected, Conversion Agent ignores the transformer. This is useful for testing and debugging, or for making minor modifications in a project without deleting the existing transformers.
`optional`	This property means that if the transformer fails, its parent component does not fail. If you deselect the `optional` property and the transformer fails, it causes the parent component to fail. For more information, see “Failure Handling” on page 226.

Transformer Quick Reference

Transformer	Description
AbsURL	Converts a relative path or URL to an absolute path.
AddEmptyTagsTransformer	An XML-to-XML transformer that adds empty elements if elements are missing from the XML.
AddString	Adds strings before and/or after the input text.
Base64Decode	Converts the base64 MIME encoding to a binary string.
Base64Encode	Converts a binary string to the base64 MIME encoding.
BidiConvert	Reverses strings in languages that are written from right to left.
BigEndianUniToUni	Converts big-endian Unicode to little-endian.
CDATADecode	Decodes a CDATA section of an XML document.
CDATAEncode	Encodes a CDATA section of an XML document.
ChangeCase	Changes the text to upper case or lower case.
CreateGuid	Generates a GUID identifier.
CreateUUID	Generates a UUID identifier.
DateFormatICU	Formats a date or time.
Dos96HebToAscii	Converts Hebrew text from the MS DOS code page to the Windows code page.
EbcdicToAscii	Converts EBCDIC to ASCII text.
EncodeAsUrl	Encodes spaces and special characters, as required in a URL.
Encoder	Converts text from one code page to another.
ExternalTransformer	Runs a custom transformer that is implemented as a DLL.
FormatNumber	Formats a number by adding a sign, decimal point, leading and trailing zeros, and a unit.
FromFloat	Converts a floating point number from binary to an ASCII string.
FromInteger	Converts an integer from binary to an ASCII string.
FromPackDecimal	Converts a number from packed decimals to an ASCII string.
FromSignedDecimal	Converts a number from signed decimals to an ASCII string.
hebrewBidi	Reverses Hebrew text from RTL to LTR.
HebrewDosToWindowsTransformer	Converts from the Hebrew MS-DOS to Windows code page.
HebrewEBCDICOldCodeToWindows	Converts Hebrew text from EBCDIC to the Windows-1255 code page.
hebUniToAscii	Converts Hebrew text from Unicode UTF-16 to the Windows-1255 code page.
hebUtf8ToAscii	Converts Hebrew text from UTF-8 to the Windows-1255 code page.
HtmlEntitiesToASCII	Converts HTML entities to plain text.
HtmlProcessor	Normalizes whitespace in an HTML document.
InjectFP	Inserts a decimal point in a number.
InjectString	Inserts a string into text.
JavaTransformer	Runs a custom transformer that is implemented in Java.
LookupTransformer	Looks up a value in a table.
NormalizeClosingTags	Changes <tag /> to <tag></tag> in XML input.

Transformer	Description
ODBCLookup	Replaces the text with data retrieved from a database.
RegularExpression	Modifies the text by using a regular expression.
RemoveMarginSpace	Trims leading and trailing space characters.
RemoveRtfFormatting	Removes all RTF formatting characters within the text.
RemoveTags	Removes HTML tags.
Replace	Replaces or deletes specified text.
Resize	Pads text to a specified size.
ReverseTransformer	Reverses a string.
RtfProcessor	Normalizes RTF code.
RtfToASCII	Converts RTF input to plain text.
SubString	Returns a substring of the input.
ToFloat	Converts a floating point number from an ASCII string to binary.
ToInteger	Converts an integer from an ASCII string to binary.
ToPackDecimal	Converts a number from an ASCII string to packed decimals.
ToSignedDecimal	Converts a number from an ASCII string to signed decimals.
TransformationStartTime	Outputs the date and/or time at which the transformation started running.
TransformByParser	Runs a parser on the input text, replacing segments of the text.
TransformByProcessor	Runs a document processor on the input text, converting it to a new document format.
TransformByService	Runs a Conversion Agent service on the input.
TransformerPipeline	Applies a sequence of transformers.
WestEuroUniToAscii	Converts text in Western European languages from Unicode UTF-16 to the Windows-1252 code page.
XSLTTransformer	Applies an XSLT transformation to XML input text.

Transformer Component Reference

This section documents the transformers that are available in Conversion Agent.

AbsURL

This transformer converts a relative file path or URL to an absolute path.

For example, if the input is `test.html` and the base URL is `http://www.example.com`, the output is `http://www.example.com/test.html`.

If the input is an absolute path, the transformer does not alter it.

Table 9-1. Basic Properties

Property	Description
AbsURL	The base path or URL.

Table 9-2. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	
optional	

AddEmptyTagsTransformer

This is an XML to XML transformer. The transformer checks if all the elements defined in the XSD schema exist in the XML input. If not, it adds empty elements to the XML.

Table 9-3. Basic Properties

Property	Description
root_element	The root element of the XML. Select from a Schema view.

Table 9-4. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	

AddString

This transformer adds strings before and/or after the input text.

Table 9-5. Basic Properties

Property	Description
pre	The string to add before the text.
post	The string to add after the text.

Table 9-6. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	
optional	

Online Sample

For an online sample, open `samples\Projects\Transformers_Example\Transformers_Example.cmw`. The first `Content` anchor in the parser is configured with an `AddString` transformer.

Base64Decode

This transformer converts the base64 MIME encoding to a binary string.

Table 9-7. Advanced Properties

Property	Description
`tolerance`	This property controls how the transformer processes whitespace characters or non-base64 sections of its input. The default is `ignore_white_spaces`. Alternatively, you can choose `ignore_none` or `ignore_non_base64`.
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

Base64Encode

This transformer converts a binary string to the base64 MIME encoding. This is useful, for example, when you want to save binary data in XML.

Table 9-8. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

BidiConvert

This transformer reverses strings that are written in right-to-left (RTL) languages, such as Hebrew and Arabic.

The input must be in the RTL format. The output is LTR.

Table 9-9. Advanced Properties

Property	Description
`disabled`	For more information about this property, see “Standard Transformer Properties” on page 103.

BigEndianUniToUni

This transformer converts big-endian Unicode to little-endian.

The transformer is supported for backwards compatibility with projects that have been upgraded from previous Conversion Agent versions. It is not available for use in new projects. Instead, set the byte order in the project properties. For more information, see “Encoding Properties” on page 214.

CDATADecode

This transformer decodes a `CDATA` section of an XML document. For example, it converts

```
<![CDATA[100 < 200]]>
```

to

```
100 < 200
```

Note: If you write the result to XML, Conversion Agent re-encodes it using the standard XML encoding:

```
100 &lt; 200
```

Table 9-10. Advanced Properties

Property	Description
disabled	For more information about these properties, see “Standard Transformer Properties” on page 103.
optional	

CDATAEncode

This transformer encodes a `CDATA` section of an XML document. For example, it converts

```
100 < 200
```

to

```
<![CDATA[100 < 200]]>
```

Table 9-11. Advanced Properties

Property	Description
disabled	For more information about these properties, see “Standard Transformer Properties” on page 103.
optional	

ChangeCase

The `ChangeCase` transformer changes text to upper case, lower case, or first-letter-capitalized.

The transformer works on English characters. It may fail on some non-English characters. For example, it does not convert the lower-case German ß to the upper-case SS.

The transformer requires that the working encoding be a code page, not UTF-8. For more information, see “Encoding Properties” on page 214.

Table 9-12. Basic Properties

Property	Description
case_type	The output case: `all_caps`, `all_lower`, or `first_cap`.

Table 9-13. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	

Online Sample

For an online sample, open `samples\Projects\Transformers_Example\Transformers_Example.cmw`. The third `Content` anchor in the parser is configured with a `ChangeCase` transformer.

CreateGuid

This transformer generates a GUID identifier. The GUID is guaranteed to be unique on every generation.

The transformer ignores its input. The GUID is not related to the input in any way.

The GUIDs may have a non-standard format on UNIX platforms. For a fully UNIX-compatible transformer, use `CreateUUID` instead.

CreateUUID

This transformer generates a UUID identifier. The UUID is guaranteed to be unique on every generation and is compatible with both Windows and UNIX platforms.

The transformer ignores its input. The GUID is not related to the input in any way.

Table 9-14. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

DateFormatICU

This transformer formats a date or time.

Example

Suppose you configure a `DateFormat` transformer with:

```
input_format = "d/M/yy"
output_format = "MM/dd/yyyy"
```

If the input is

```
13/3/05
```

the output is

```
03/13/2005
```

Supported Formats

The transformer uses the ICU conventions to represent the date and time format. The following table lists the symbols that you can use in the format patterns. For more information, see:

http://icu.sourceforge.net/apiref/icu4c/classSimpleDateFormat.htm1

Pattern Symbol	Meaning	Type	Examples
`G`	Era designator	Text	`AD`
`y`	Year	Number	`1996`
`u`	Extended year	Number	`-200, meaning 201 BC`
`M`	Month in year	Text or number	`July` `07`
`d`	Day in month	Number	`10`

Pattern Symbol	Meaning	Type	Examples
h	Hour in AM/PM (1-12)	Number	12
H	Hour in day (0-23)	Number	0
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Fractional second	Number	978
E	Day of week	Text	Tuesday
e	Day of week (local 1-7)	Number	2
D	Day in year	Number	189
F	Day of week in month	Number	2, meaning the 2nd Wednesday in July
w	Week in year	Number	27
W	Week in month	Number	2
a	AM/PM marker	Text	PM
k	Hour in day (1-24)	Number	24
K	Hour in AM/PM (0-11)	Number	0
z	Time zone	Time	Pacific Standard Time
Z	Time zone (RFC 822)	Number	-0800
v	Time zone (generic)	Text	Pacific Time
g	Julian day	Number	2451334
A	Milliseconds in day	Number	69540000
' '	The text within single quotes is interpreted as a literal string	Text	'Today is 'dd/MM/yyyy generates output such as Today is 15/03/2005
''	Literal single quote	Text	'o''clock' generates the output o'clock

The count of pattern symbols further determines the format:

- For text: Four or more pattern symbols means to use the full form. Fewer than four means to use a short or abbreviated form if it exists. For example, if EEEE produces Monday, EEE produces Mon.
- For numbers: The number of pattern symbols is the minimum number of digits. Shorter numbers are zero-padded. For example, if m produces 6, mm produces 06.
- For years: The two-digit year is yy, and the four-digit year is yyyy. For example, if yy produces 05, yyyy produces 2005.
- For months: If M produces 1, then MM produces 01, MMM produces Jan, and MMMM produces January.

All non-alphabetic characters are interpreted as literals, even if they are not enclosed in single quotes. For example, dd/MM/yyyy HH:mm produces 15/03/2005 13:15, containing the /, space, and : characters.

Table 9-15. Basic Properties

Property	Description
input_format	The format of the input date, for example, d/M/yy. You can type the format, or browse to a data holder that contains the format.
output_format	The format of the output date, for example, MM/dd/yyyy. You can type the format, or browse to a data holder that contains the format.

Table 9-16. Advanced Properties

Property	Description
disabled	For more information about these properties, see "Standard Transformer Properties" on page 103.
optional	

Note: If you open a project that was created in a previous Conversion Agent version, you might observe that it uses the older `DateFormat` processor. This component is supplied for backwards compatibility. In new projects, use `DateFormatICU`.

Dos96HebToAscii

This transformer converts Hebrew text from the MS DOS code page to the Windows-1255 code page.

EbcdicToAscii

This transformer converts EBCDIC to ASCII text.

EncodeAsUrl

This transformer encodes spaces and special characters as required in a URL. The characters are encoded as hexadecimal preceded by a `%` symbol.

Note: The parentheses characters are not encoded. They are displayed as `(` and `)`.

For example, the transformer converts

```
http://www.example.com?name=Ron Lehrer
```

to

```
http://www.example.com?name=Ron%20Lehrer
```

Table 9-17. Advanced Properties

Property	Description
name	For more information about these properties, see "Standard Transformer Properties" on page 103.
remark	
disabled	

Online Sample

For an online sample, open `samples\Projects\Transformers_Example\Transformers_Example.cmw`. The fourth `Content` anchor in the parser is configured with an `EncodeAsUrl` transformer.

Encoder

This transformer converts text from one code page to another.

Table 9-18. Basic Properties

Property	Description
input_code_page	The input code page.
output_code_page	The output code page.

Table 9-19. Advanced Properties

Property	Description
add_prefix	Adds a Byte Order Mark (BOM) when the output encoding is UTF-8 or UTF-16.
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	
optional	

ExternalTransformer

This component allows you to run a custom transformer that is implemented as a C++ DLL.

Note: This component is supported for backwards compatibility with existing custom transformers. For more information about custom transformers and other external components, see the *Conversion Agent Engine Developer Guide*.

The following instructions are for the Microsoft Visual C++ compiler, running on a Microsoft Windows platform.

To create a custom C++ transformer:

1. Copy the following file from the Conversion Agent installation folder:

   ```
   samples\SDK\ExternalTransformer\ExternalTransformerExample.c
   ```

2. Using the Visual C++ compiler, create a Win32 dynamic-link library project, and insert the C file into the project.

3. Edit the following function, which performs the transformation:

   ```
   __declspec(dllexport) int transform(const char* in, char** out)
   ```

   In the sample implementation, the function reverses the text. Replace the sample code with your implementation.

4. Implement the following function, which releases a buffer:

   ```
   __declspec(dllexport) void release_buf(char* buf)
   ```

5. Compile the DLL.

6. Store the DLL in the `externLibs\user` subfolder of the Conversion Agent installation folder.

7. Define an `ExternalTransformer` that references the DLL.

8. Optionally, add the `ExternalTransformer` to the component list that Conversion Agent Studio displays. For more information about customizing the component list, see *Using Conversion Agent Studio in Eclipse*.

Table 9-20. Basic Properties

Property	Description
import_dll	Browse to the DLL in the `externLibs\user` folder.

Table 9-21. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	

Table 9-21. Advanced Properties

Property	Description
disabled	
optional	

FormatNumber

This transformer formats a number by adding a sign, decimal point, leading or trailing zeros, and unit.

Table 9-22. Basic Properties

Property	Description
sign	Adds a plus or minus sign at the beginning or end of the number. The options are: - un_signed. Deletes a sign if present. - leading_sign. - trailing_sign. - negative sign only. - as in source. Does not change the input sign.
insert_decimal_point	Sets the decimal point symbol. The options are none, point, and comma.
unit_type	Adds a unit after the number. Select a unit such as meter, cm, mm, or inch. If you do not want to add a unit, select undefined.
size_of_integer_part	Pads the integer part with leading zeros to the indicated size.
number_of_decimals	Pads the decimal part with trailing zeros to the indicated size.

Table 9-23. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	
optional	

FromFloat

This transformer converts a floating point number from binary to an ASCII string representation.

Table 9-24. Advanced Properties

Property	Description
size	Size of the number: single_precision_32_bit or double_precision_64_bit.
name	For more information about these properties, see “Standard Transformer Properties” on page 103
remark	
disabled	
optional	

FromInteger

This transformer converts an integer from binary to an ASCII string representation, in decimal, octal, or hexadecimal.

Table 9-25. Basic Properties

Property	Description
`size`	Size in bytes of the binary representation. The supported values are 1 to 8.

Table 9-26. Advanced Properties

Property	Description
`signed`	If selected, the transformer adds a sign to the number.
`to_base`	The base of the output: `decimal, octal, hexadecimal, lowercase hexadecimal`.
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

FromPackDecimal

This transformer converts a number from packed decimals to an ASCII string representation.

Table 9-27. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

FromSignedDecimal

This transformer converts a number from signed decimals to an ASCII string representation.

Advanced Properties

Property	Description
insert_sign_symbol	Adds a plus or minus sign before or after the number. The options are no, before, and after.
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	
optional	

hebrewBidi

This transformer reverses Hebrew text from RTL to LTR. It is similar to BidiConvert, but it uses a different algorithm for the reversal, which may produce slightly different results.

HebrewDosToWindowsTransformer

This transformer converts Hebrew documents from the MS DOS Hebrew code page to the Windows Hebrew code page.

HebrewEBCDICOldCodeToWindows

This transformer converts Hebrew text from EBCDIC to the Windows-1255 code page.

hebUniToAscii

This transformer converts Hebrew text from Unicode UTF-16 to the Windows-1255 code page.

hebUtf8ToAscii

This transformer converts Hebrew text from Unicode UTF-8 to the Windows-1255 code page.

HtmlEntitiesToASCII

This transformer converts HTML entities to plain text. For example, it converts © or © to a copyright symbol (©).

Supported Entities

The transformer supports the ISO 8859-1 (Latin-1) entities that are defined in the HTML 4.0 reference, http://www.w3.org/TR/1998/REC-html40-19980424/sgml/entities.html. The supported entities include:

- &, <, >, and " (& < > ", respectively)
- Numeric character codes &#0; to ÿ
- Entities for Latin-1 characters: = non-breaking space, © = copyright, etc.

The transformer does not support extended characters, that is, codes greater than 255 or non-Latin-1 characters.

Output Encoding for Upper-ASCII Characters

If the transformer output contains upper-ASCII characters, be sure to select an output encoding that supports the characters, such as Windows-1252 or UTF-8.

Include an encoding attribute in the XML processing instruction. Otherwise, the Studio might not be able to display the characters.

For more information, see “Encoding Properties” on page 214.

Table 9-28. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	

HtmlProcessor

This transformer normalizes whitespace according to HTML conventions. It converts any sequence of tabs, line breaks, and space characters to a single space character.

You can use this transformer to normalize whitespace in any type of text. It is not restricted to HTML text.

The component can also be used as a format preprocessor. For more information, see “Format Preprocessor Component Reference” on page 48.

InjectFP

This transformer inserts a decimal point at a specified location in a number. For example, the transformer can convert 12345 to 123.45.

Table 9-29. Basic Properties

Property	Description
`digits_after_decimal_point`	The number of digits after the decimal point.

Table 9-30. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

InjectString

The `InjectString` transformer inserts a string into text.

Table 9-31. Basic Properties

Property	Description
`injection_place`	The location in the text to insert the string. 0 means to insert the string before the text.
`string_to_inject`	The string to insert.

Table 9-32. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	

JavaTransformer

This component allows you to run a custom transformer that is implemented in Java.

Note: This component is supported for backwards compatibility with existing custom transformers. For more information about custom transformers and other external components, see the *Conversion Agent Engine Developer Guide*.

To create a custom Java transformer:

1. Create a new Java project and package, for example, named `MyTransformer`
2. Create a class, for example, named `TransformerTest`.
3. In the class, define a method having the following syntax. The method can have any name.

   ```
   public static byte[] Transform(byte[] in)
   ```

4. Create a jar file containing the class.
5. Store the jar file in the `externLibs\user` subfolder of the Conversion Agent installation folder.
6. Define a `JavaTransformer` that references the class and method.
7. Optionally, add the `JavaTransformer` to the component list that Conversion Agent Studio displays. For more information about customizing the component list, see *Using Conversion Agent Studio in Eclipse*.

Example

The following example is a transformer that changes text to upper case.

```
package MyTransformer;
public class TransformerTest
{
  public static byte[] Transform(byte[] in)
  {
    String str = new String(in);
    String ret = str.toUpperCase();
    return ret.getBytes();
  }
}
```

Table 9-33. Basic Properties

Property	Description
java_class	The path of the Java class, for example, `MyTransformer/TransformerTest`.
method	The method to run, for example, `Transform`.

Table 9-34. Advanced Properties

Property	Description
disabled	For more information about these properties, see “Standard Transformer Properties” on page 103.
optional	

LookupTransformer

This transformer looks up a value in a table. For example, you can configure a `LookupTransformer` to look up values in the following table:

Key	Value
1	George Washington
2	John Adams
3	Thomas Jefferson
4	James Madison

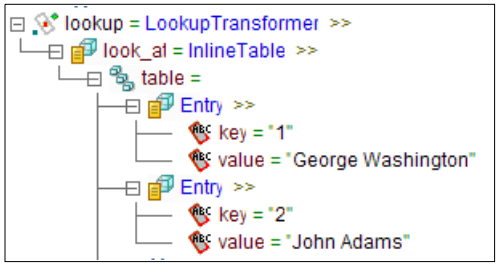
If the input of the transformer is `3`, the output is `Thomas Jefferson`.

There are three ways to define a lookup table:

- **Inline lookup table.** The table is fully defined in the IntelliScript.
- **XML lookup table stored in a file.** The table is defined in an XML file that the LookupTransformer retrieves.
- **XML lookup table generated dynamically.** The transformation prepares an XML lookup table at runtime and stores it in a data holder.

Defining an Inline Table

To define an inline table, configure the key-value pairs in the IntelliScript, as in the following example.



Storing an XML Lookup Table in a File

Prepare an XML file conforming with the schema `lookupTableDefinition.xsd`. The schema is stored in the `doc` subdirectory of the Conversion Agent installation directory. The following XML document is an example:

```
<?xml version="1.0" encoding="windows-1252" ?>
<lt:LookupTable xmlns:lt="http://www.Itemfield.com/Engine/V4/lookupTable"
  matchCase="false">
  <lt:Entry key="1" value="George Washington" />
  <lt:Entry key="2" value="John Adams" />
</lt:LookupTable>
```

Creating an XML Lookup Table Dynamically

A transformation can create an XML lookup table at runtime. For example, the transformation might run a secondary parser that generates the XML structure. The structure is identical to that of an XML lookup table stored in file.

The transformation must store the XML string in a data holder of type `xs:string`. It can do this by running a `WriteValue` action configured to write the XML structure to an `OutputDataHolder`.

For more information about configuring a secondary parser that generates a dynamic lookup table, see “AdditionalInputPort” on page 16.

Table 9-35. Basic Properties

Property	Description
look_at	Under this property, you define the type of lookup table used by the transformer. Select one of the following values: - InlineTable - XMLLookupTable - DynamicTable If you use the same lookup table repeatedly, consider defining an InlineTable or an XMLLookupTable at the global level of the IntelliScript. You can then reference the table by name in the look_at property.

Table 9-36. Advanced Properties

Property	Description
disabled	For more information about these properties, see “Standard Transformer Properties” on page 103.
optional	
name	
remark	

NormalizeClosingTags

For XML input, this transformer removes shorthand closing tags from empty elements. It changes <tag/> to <tag></tag>.

The transformer does not correct incorrect XML. It converts well-formed XML from one style of closing tag to another.

Table 9-37. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	

ODBCLookup

The ODBCLookup transformer uses the input text to query a database. It replaces the text with the query result.

Table 9-38. Basic Properties

Property	Description
db_connection	The database connection. The value is an ODBC_Text_Connection subcomponent, which specifies a DSN, user name, and any other required connection parameters.

Table 9-39. Advanced Properties

Property	Description
query	A SQL SELECT or EXEC query that retrieves the data from the database. Use? to represent the input text, for example: SELECT Name FROM Employees WHERE Id=? The query must retrieve a single field, which is the transformer output.
retry	The number of retries if the first connection attempt fails.
disabled	For more information about these properties, see “Standard Transformer Properties” on page 103.
optional	
name	
remark	

RegularExpression

The RegularExpression transformer performs a pattern search on the input text. It replaces instances of the pattern with a specified string.

For example, suppose that a Content anchor retrieves the text:

```
transformer
```

You configure the anchor with a RegularExpression transformer that searches for the pattern t.+s. The pattern means the letter t, followed by one or more of characters, followed by the letter s. You configure the transformer to replace the pattern with the character X.

The pattern matches the substring trans of the input. The transformer replaces the substring and outputs:

```
Xformer
```

Table 9-40. Basic Properties

Property	Description
exp	A regular expression for the search criterion.
replacement	The replacement text. Leave blank to delete the found text.

Table 9-41. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	
optional	

Regular Expressions

A regular expression is a string that uses a standard syntax to define a search pattern. The syntax is similar to a wildcard search, but with greatly enhanced search capabilities.

Consider, for example, the following text:

```
Peter Piper picked a peck of pickled peppers
```

The following are illustrations of regular expressions and the strings that they retrieve from the above text.

Regular expression	Retrieved string	Explanation
`Peter`	`Peter`	Retrieves the literal string.
`Pip.+cked`	`Piper picked`	Retrieves a string beginning with `Pip`, followed by at least one character, followed by `cked`.
`[Pp]i[a-k]{3}`	`picke`	Retrieves a string beginning with `P` or `p`, followed by `i`, followed by exactly 3 characters between `a` and `k`. Finds `picke` because the string matches these criteria. Does not find `Piper` or `pickl` because the letters `p` and `l` do not lie between `a` and `k`.

The following table lists special characters that you can use in regular expressions. The table is not comprehensive. There are other syntax combinations that have special meanings in regular expressions.

Character		Meaning	Example
`*`	(asterisk)	Match zero or more instances of the preceding character	`ab*c` matches `ac`, `abc`, or `abbbc`.
`?`	(question)	Match zero or one instance of the preceding character	`ab?c` matches `ac` or `abc`.
`+`	(plus)	Match one or more instances of the preceding character	`a+` matches `a` or `aaaa`.
`{}`	(braces)	Matches the specified number of instances of the preceding character	`ab{2}c` matches `abbc`.
`-`	(minus)	Use inside `[]` to represent a range of characters	`[A-Za-z]` matches any character in the English alphabet.
`[]`	(brackets)	Match any of a set of characters	`a[bst]c` matches `abc`, `asc`, or `atc`.
`.`	(period)	Match any character	`a.c` matches `abc`, `a c`, or `a1c`.
`^`	(caret)	Match the start of the input text	`^P.` matches `Pe` but not `Pi` in the Peter Piper example.
`$`	(dollar)	Match the end of the input text	`r.$` matches `rs` in the Peter Piper example.
`|`	(bar)	Match either of two expressions	`abc|ded` matches `abc` or `def`.
`()`	(parentheses)	Grouping	`A(abc|def)` matches `Aabc` or `Adef`.
`\`	(backslash)	Escapes one of the other special characters, treating it as a literal character	`\.` matches a literal period, rather than any character.

Conversion Agent uses the Regex++ implementation of regular expressions, copyright © 1998-2003 by Dr. John Maddock, Version 3.12, 18 April 2000. For detailed information about the regular expression syntax supported by this implementation, see http://www.boost.org/libs/regex/doc/index.html.

For general information about regular expressions, see http://en.wikipedia.org/wiki/Regular_expression and http://www.regular-expressions.info/index.html.

Preserving Portions of the Original Text

In the `exp` property, you can enclose portions of the regular expression in parentheses. In the `replacement` property, you can use:

- `$0` to identify the entire text that matches the regular expression
- `$1` to identify the substring that matches the first parenthesized portion of the regular expression
- `$2`, `$3`, and so forth, to identify the substrings that match the second, third, etc. parenthesized portions

For example, suppose you set:

```
exp = abc([0-9]+)(def)
replacement = $1
```

This replaces `abc5624def` with `5624`.

Alternatively, suppose you set:

```
exp = abc([0-9]+)(def)
replacement = $2ZYX$1
```

This replaces `abc5624def` with `defZYX5624`.

RemoveMarginSpace

This transformer deletes leading and trailing space characters from the text.

Table 9-42. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	

RemoveRtfFormatting

The transformer removes RTF formatting instructions from the text.

Table 9-43. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	

RemoveTags

This transformer removes HTML tags from the input text.

It replaces the tags at internal locations in the text with a separator string, such as a space character. It does not insert the separator string at the beginning or end of the text. Adjacent multiple tags are transformed into a single separator.

Table 9-44. Basic Properties

Property	Description
`replace_with`	The separator string.

Table 9-45. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

Replace

This transformer finds and replaces strings in the input text. Leaving the `replace_with` property empty deletes the found text.

Table 9-46. Basic Properties

Property	Description
`find_what`	The text to find. The value is one of the following searcher components: - `NewlineSearch`. Finds a newline character. - `PatternSearch`. Finds text that matches a regular expression. - `SegmentSearch`. Finds a segment from a specified opening marker to a closing marker. - `TextSearch`. Finds a specified string. For more information, see the "Searcher Component Reference" on page 94.
`replace_with`	The replacement string.

Table 9-47. Advanced Properties

Property	Description
`occurrence`	Specifies which occurrences to replace: `all`, `first`, or `last`.
`name`	For more information about these properties, see "Standard Transformer Properties" on page 103.
`remark`	
`disabled`	
`optional`	

Online Sample

For an online sample, open `samples\Projects\Transformers_Example\Transformers_Example.cmw`. The second and fifth `Content` anchors in the parser are configured with `Replace` transformers.

Resize

This transformer fits the input text to a specified size. It pads or truncates the text as required.

Table 9-48. Basic Properties

Property	Description
`size`	The desired size. Type an integer, or click the browse button and select a data holder that contains an integer.
`padding_character`	The padding character, such as a space character. Type the character, or click the browse button and select a data holder that contains a character.
`align`	The text alignment within the resized string. The options are: - `left`. Padding or trimming is on the right. - `right`. Padding or trimming is on the left.

ReverseTransformer

This transformer reverses a string. For example, it transforms `1234` to `4321`.

RtfProcessor

This transformer normalizes RTF code. It is also available as a format preprocessor. For more information, see "Format Preprocessor Component Reference" on page 48.

RtfToASCII

This transformer converts RTF input to plain text. It removes RTF control words from the text.

Table 9-49. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	

SubString

This transformer returns a substring of the input, starting and ending at specified locations.

Table 9-50. Basic Properties

Property	Description
`begin`	The start location. 0 means to start at the beginning of the input.
`end`	The end location.

Table 9-51. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	

ToFloat

This transformer converts a floating point number from an ASCII string representation to binary.

Table 9-52. Advanced Properties

Property	Description
`size`	Size of the number: `single_precision_32_bit` or `double_precision_64_bit`.
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

ToInteger

This transformer converts an integer from an ASCII string representation to a binary integer. The string input can be a decimal, octal, or hexadecimal representation.

Table 9-53. Basic Properties

Property	Description
`size`	Size in bytes of the binary representation. The supported values are 1 to 8.

Table 9-54. Advanced Properties

Property	Description
signed	If selected, the input has a plus or minus sign.
from_base	The base of the input: decimal, octal, hexadecimal, lowercase hexadecimal.
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	
optional	

ToPackDecimal

This transformer converts a number from an ASCII string representation to packed decimals.

Table 9-55. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	
optional	

ToSignedDecimal

This transformer converts a number from an ASCII string representation to signed decimals.

Table 9-56. Advanced Properties

Property	Description
insert_sign_ symbol	Adds a sign symbol, plus or minus, before or after the number. The options are no, before, and after.
name	For more information about these properties, see “Standard Transformer Properties” on page 103.
remark	
disabled	
optional	
unsigned	

TransformationStartTime

This transformer outputs the date and/or time at which the transformation started running.

The transformer ignores its input. It copies the date and time from the `VarSystem` variable, and it formats the output according to your specification.

Table 9-57. Basic Properties

Property	Description
`format`	The format of the date and time. You can type the format or browse to a data holder that contains the format. For more information about the supported formats, see the `DateFormatICU` transformer.

Table 9-58. Advanced Properties

Property	Description
`disabled`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`optional`	

TransformByParser

This transformer runs a parser on its input text. The parser must contain `FindReplaceAnchor`components that mark segments of the text for replacement. When the parser completes execution, the transformer performs the replacements.

The transformer output is the modified text. Conversion Agent ignores any XML output that the parser generates.

For more information about this transformer, see “FindReplaceAnchor” on page 86.

Table 9-59. Basic Properties

Property	Description
`parser`	The name of the parser.

Table 9-60. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

Online Sample

For an online sample, open `samples\Projects\TransformByParser\TransformByParser.cmw`. The sample uses `TransformByParser` to replace every instance of the string `~NL~` with a carriage return followed by a linefeed.

To run the TransformByParser sample:

1. Set `MyTransformByParser` as the startup component.
2. Run the transformer.
3. At the prompt, select the source file `Report.edi`.

   The transformer stores its output in `Results\Transformation of Report.edi`. You can compare the output with the source in Notepad.

TransformByProcessor

This transformer runs a document processor on its input. The output of the transformer is the output of the document processor. For more information, see “Document Processors” on page 21.

For example, you can use the transformer to convert an Excel document to text by invoking the `ExcelToTxt` document processor. The input of the transformer must in a valid Excel format.

Table 9-61. Basic Properties

Property	Description
`processor`	The name of the document processor.

Table 9-62. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

TransformByService

This transformer runs a Conversion Agent service on its input. The output of the transformer is the output of the service. For more information, see “Deploying Conversion Agent Services” on page 231.

For example, if you use the transformer to invoke a parser service, the output of the transformer is an XML string.

Table 9-63. Basic Properties

Property	Description
`service_name`	The name of the service.

Table 9-64. Advanced Properties

Property	Description
`disable_automatic_encoding`	If selected, Conversion Agent does not apply the input and output encodings that are defined in the service. For more information, see “Encoding Properties” on page 214.
`parameters`	A list of initial values that Conversion Agent should assign to variables defined in the service. In each element of the list, specify the name of a variable and its value. For more information, see “Initializing Variables at Runtime” on page 63.
`disabled`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`optional`	

TransformerPipeline

This transformer applies a sequence of nested transformers to its input.

Table 9-65. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

WestEuroUniToAscii

This transformer converts text in western-European languages from Unicode UTF-16 to the Windows-1252 code page.

The transformer is supported for backwards compatibility with projects that have been upgraded from previous Conversion Agent versions. It is not available for use in new projects. Instead, set the encoding in the project properties. For more information, see “Encoding Properties” on page 214.

XSLTTransformer

This transformer applies an XSLT transformation to XML input text.

For example, you might use a parser to extract data from an XML document. A `Content` anchor retrieves a complete, well-formed branch of the XML tree. You can configure the `Content` anchor with an `XSLTTransformer` that runs an XSLT transformation on the branch.

Table 9-66. Advanced Properties

Property	Description
`xslt_file`	The path and filename of the XSLT file.
`name`	For more information about these properties, see “Standard Transformer Properties” on page 103.
`remark`	
`disabled`	
`optional`	

Transformer Subcomponent Reference

This section describes subcomponents that you can assign as the values of certain transformer properties.

InlineTable

This component lets you define a lookup table in the IntelliScript. The table is used by the `LookupTransformer`. For example, you might specify:

Key	Value
1	George Washington
2	John Adams
3	Thomas Jefferson
4	James Madison

If the input of the transformer is `3`, the outputs is `Thomas Jefferson`.

Table 9-67. Basic Properties

Property	Description
`table`	Under this property, enter a sequence of `entry` components. In each `entry`, specify `key` and `value` strings.

Table 9-68. Advanced Properties

Property	Description
`match_case`	If selected, the `key` string is considered to be case-sensitive.

ODBC_Text_Connection

The subcomponent defines a database connection. It is used, for example, in the `ODBCLookup` transformer.

Before using this component, use the operating system tools to define a DSN for the database connection.

Table 9-69. Basic Properties

Property	Description
`DSN`	The data source name of the connection.

Table 9-70. Advanced Properties

Property	Description
`username`	User name for the database connection.
`password`	Password of the user.
`timeout`	Time in seconds to wait for the database response.

XMLLookupTable

This component lets you specify an XML file that contains a lookup table. The table is used by the `LookupTransformer`.

The XML file must have the following syntax:

```
<LookupTable>
  <Entry key="..." value="..." />
  ...
</LookupTable>
```

The XML must conform to the following schema, which is located in the Conversion Agent installation folder:

```
doc\lookupTableDefinition.xsd
```

Example

The following is an XML lookup table:

```
<LookupTable>
  <Entry key="1" value="George Washington" />
  <Entry key="2" value="John Adams" />
  <Entry key="3" value="Thomas Jefferson" />
  <Entry key="4" value="James Madison" />
</LookupTable>
```

Table 9-71. Basic Properties

Property	Description
`xml_file_name`	Browse to the XML file.

Table 9-72. Advanced Properties

Property	Description
`match_case`	If selected, the `key` element is considered to be case-sensitive.

CHAPTER 10

Actions

This chapter includes the following topics:

Overview

Actions are components that perform operations on data that Conversion Agent has extracted from a source document. Some examples of the supported actions are:

- Arithmetic computations
- String concatenations
- Submitting forms to a web server
- Activating a secondary parser, serializer, or mapper
- Querying a database

You can use the out-of-the-box actions supplied with Conversion Agent, or you can define custom actions.

This chapter explains how to use actions and documents the actions that are available in Conversion Agent.

How Actions Work

An action takes its input from the data holders that are currently available. A single action can have multiple inputs.

If the action is embedded in a parser, the available data holders are the ones that the parser has generated. In a serializer, the data holders are the ones that exist in the input XML, plus any additional data holders that the serializer has generated. For a mapper, the data holders can be in either the input or the output.

The action performs operations on the input and generates output. You can configure many actions to store their output in data holders.

In most actions, the input and output data holders must have simple data types. They must not contain nested elements. A few actions work with data holders that contain nested elements, with multiple-occurrence data holders, or with other special types.

An action can have additional effects, such as writing to a file, updating a database, or submitting data to an external application.

Comparison between Actions and Transformers

Some actions perform operations that are similar to transformers, for example, modifying a string or querying a database. However, actions differ from transformers in some fundamental ways. The following table summarizes the differences.

Operation	Transformers	Actions
Input	The input of a transformer is a single string.	The input is implemented by the action. An action can have multiple inputs. The inputs can be data holders.
Output	The output of a transformer is a string.	The output is implemented by the action. For example, an action can create output data holders.
Side effects	A transformer has no side effects, other than modifying the input string.	An action can have side effects, such as updating a database.

Defining Actions

You can define actions by editing the IntelliScript. You can insert the actions under the `contains` line of components such as a `Parser`, `Serializer`, `Mapper`, `Group`, or `RepeatingGroup`. Essentially, you can insert actions in any location where you can insert anchors, serialization anchors, or mapper anchors.

The actions run in sequence with the anchors that you specify in the same location. In a parser, you can set the `phase` property of an action, which controls whether it runs in the initial, main, or final stage of the parsing procedure. For more information, see "Search Phases" on page 73.

Standard Action Properties

In this section, we review certain standard properties of actions. For more information about the properties of specific actions, see the "Action Component Reference" on page 135.

Property	Description
`name`	A name that you assign to the action. Conversion Agent displays the name in the event log. This can help you find an event that was caused by the particular action.
`remark`	A comment describing the action.
`disabled`	If selected, Conversion Agent ignores the action. This is useful for testing and debugging, or for making minor modifications in a project without deleting the existing actions.
`optional`	By default, if an action fails, its parent component fails. If you select the `optional` property, the parent component does not fail. For more information, see "Failure Handling" on page 226.
`phase`	The processing phase during which Conversion Agent should execute the action: `initial`, `main`, or `final`. This property has an effect only if the action is used in a parser.

Action Quick Reference

Action	Description
AddEventAction	Writes a message in the event log.
AppendListItems	Concatenates a list of strings stored in a multiple-occurrence data holder.
AppendValues	Concatenates strings.
CalculateValue	Performs a computation defined in a JavaScript expression.
CombineValues	Generates all possible concatenations from multiple-occurrence data holders.
CreateList	Fills a multiple-occurrence data holder with specified data.
CustomLog	Writes a custom log message.
DateAddICU	Increments a date.
DateDiffICU	Computes the difference between two dates.
DownloadFile	Downloads a file.
DownloadFileToDataHolder	Downloads the content of a file into a data holder.
DumpValues	A debugging tool for dumping extracted data.
EnsureCondition	Evaluates a JavaScript expression. If the expression is `false`, the action fails.
ExcludeItems	Deletes values from a multiple-occurrence data holder.
ExternalCOMAction	Runs a custom action that is implemented as an ActiveX DLL.
JavaScriptFunction	Runs a JavaScript function.
Map	Copies a data holder, optionally running transformers on the value.
ODBCAction	Runs a database query.
ResetVisitedPages	Resets the list of visited pages, permitting repeat visits to a page.
RunMapper	Runs a mapper.
RunParser	Runs a parser.
RunSerializer	Runs a serializer.
SetValue	Fills a data holder with predefined content.
Sort	Sorts the occurrences of a multiple-occurrence data holder.
SubmitForm	Submits an HTML form using the Post method and parses the response.
SubmitFormGet	Submits an HTML form using the Get method and parses the response.
WriteValue	Writes a value to a location such as a file, MSMQ queue, or database.
XSLTMap	Runs an XSLT transformation.

Action Component Reference

This section documents the actions that are available in Conversion Agent.

AddEventAction

This action adds a message to the event log.

Table 10-1. Basic Properties

Property	Description
`severity`	The severity level of the message. The options are `notification`, `warning`, `failure`, or `fatal error`.
`message`	The message string.

Table 10-2. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`phase`	

AppendListItems

The `AppendListItems` action concatenates the strings in a multiple-occurrence data holder.

For more information about preparing the input for this action, see “Mapping to Multiple-Occurrence Data Holders” on page 69.

Example

A source document contains the following space-separated text:

```
H E L L O
```

When you parse the document, you want to remove the spaces and store the result in an XML element called `Greeting`.

One way to do this is to create a multiple-occurrence variable called `VarLetter`. Create several `Content` anchors that retrieve the individual letters and store them in occurrences of `VarLetter`.

Then, use the `AppendListItems` action to concatenate the occurrences of `VarLetter` and store the result in the `Greeting` element. The result is:

```
<Greeting>HELLO</Greeting>
```

Table 10-3. Basic Properties

Property	Description
`input`	The multiple-occurrence data holder.
`output`	A data holder to store the output.

Table 10-4. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

Online Sample

For an online sample of this action, open the project `samples\Projects\AppendListItems\AppendListItems.cmw`. The sample uses a `RepeatingGroup` to store values in a multiple-occurrence variable. It then uses as an `AppendListItems` action to concatenate the values.

AppendValues

The `AppendValues` action concatenates strings.

Example

A parser has generated the following XML:

```
<Name>
  <First>Ron</First>
  <Last>Lehrer</Last>
<Name>
```

You can configure an `AppendValues` action that outputs:

```
<FullName>Ron Lehrer</FullName>
```

Table 10-5. Basic Properties

Property	Description
`input`	A list of data holders containing the values to be appended.
`output`	A data holder to store the output.

Table 10-6. Advanced Properties

Property	Description
`skip_unfound_values`	If selected, and one of the input data holders is missing, the action continues. If not selected, the action fails.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

CalculateValue

The `CalculateValue` action performs a computation that is defined by a JavaScript expression.

For example, you can use the action to compute a sum of numerical values or to concatenate string values.

Note: For more information about the JavaScript syntax that Conversion Agent supports, see `EnsureCondition`. For more information about how Conversion Agent handles the precision of `xs:decimal` and `xs:float` values, see “Precision of Numerical Data” on page 54.

Example

A parser has generated the following XML:

```
<ItemOrdered>
  <Name>Gizmo</Name>
  <Quantity>100</Quantity>
  <Price>25<Price>
</ItemOrdered>
```

You can use a CalculateValue action to generate the output:

```
<ItemOrdered>
  <Name>Gizmo</Name>
  <Quantity>100</Quantity>
  <Price>25<Price>
  <Total>2500</Total>
</ItemOrdered>
```

To do this, define the Name and Quantity elements as input parameters. Specify the JavaScript expression $1 * $2, and store the result in the Total element.

Table 10-7. Basic Properties

Property	Description
params	Data holders that contain the input parameters.
expression	The JavaScript expression. Use $1, $2,... $9, to represent the input parameters.
result	A data holder to store the output.

Table 10-8. Advanced Properties

Property	Description
failure_action	The behavior in the event of a failure. The options are: - Ignore. Continue the transformation. - HaltExecution. Stop the transformation.
name	For more information about these properties, see “Standard Action Properties” on page 134.
remark	
disabled	
optional	
phase	

Online Sample

For an online sample of this action, open the project samples\Projects\CalculateValue\CalculateValue.cmw. The sample retrieves three numbers from a source document and stores them in variables. It uses a CalculateValue action to compute a mathematical function of the numbers.

CombineValues

The CombineValues action generates all possible combinations from lists of strings that are stored in multiple-occurrence data holders. It concatenates the strings in each combination, generating an output list.

The input of this action must include one or more multiple-occurrence data holders. Optionally, it may also include single-occurrence data holders. For more information, see “Multiple-Occurrence Data Holders” on page 64.

The output is a multiple-occurrence data holder. Each occurrence of the data holder stores a combination.

This action is useful, for example, to prepare the VarFormData variable that is required by the SubmitForm action.

Example

In a multiple-occurrence variable called VarDay, you have stored the list Monday, Tuesday. In a multiple-occurrence variable called VarTime, you have stored morning, afternoon. In a single-occurrence variable called VarSpace, you have stored a space character.

Suppose you run CombineValues on VarDay, VarSpace, and VarTime, with an output data holder called DayTime. The output is:

```
<DayTime>Monday morning</DayTime>
<DayTime>Monday afternoon</DayTime>
<DayTime>Tuesday morning</DayTime>
<DayTime>Tuesday afternoon</DayTime>
```

Table 10-9. Basic Properties

Property	Description
input	From a Schema view, select the data holders containing the input. Typically, at least one of the inputs should be a multiple-occurrence data holder.
output	A multiple-occurrence data holder where the action stores its output.

Table 10-10. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Action Properties” on page 134.
remark	
disabled	
optional	
phase	

Online Sample

For an online sample of this action, open the project `samples\Projects\CombineValues\CombineValues.cmw`. The sample retrieves lists of days, months, and years from a source document. It uses a `CombineValues` action to generate all possible dates from the lists.

For an additional online sample, see the `SubmitForm` action.

CreateList

This action inserts data in a list. The output is a multiple-occurrence data holder containing the list. For more information, see “Multiple-Occurrence Data Holders” on page 64.

Nested in this component, enter the data values.

Example

If the input data values are

```
Jack
Jennie
Larissa
```

the action can create the output

```
<Name>
  <First>Jack</First>
  <First>Jennie</First>
  <First>Larissa</First>
```

```
</Name>
```

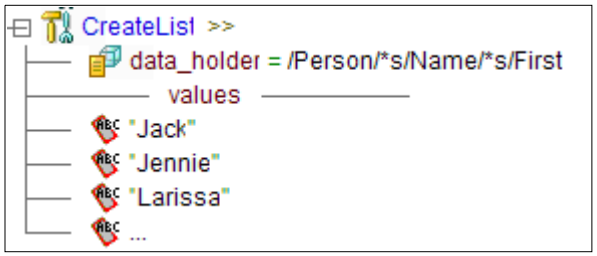


Table 10-11. Basic Properties

Property	Description
`data_holder`	The multiple-occurrence data holder where the action should store the list.

Table 10-12. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`phase`	

CustomLog

This component can be used as the value of the `on_fail` property. In the event of a failure, the `CustomLog` component runs a serializer that prepares a log message. The system writes the message to a specified location.

For more information about the `on_fail` property, see “Failure Handling” on page 226.

Table 10-13. Basic Properties

Property	Description
`run_serializer`	A serializer that prepares the log message. Define a serializer in this location, or enter the name of a globally defined serializer.

Table 10-14. Advanced Properties

Property	Description
`output`	The output location. The options include: - `MSMQOutput`. Writes to an MSMQ queue. - `OutputDataHolder`. Writes to a data holder. - `OutputFile`. Writes to a file. - `ResultFile`. Writes to the default results file of the transformation. - `OutputCOM`. Uses a custom COM component to output the data. Do not select this option directly. Instead, select the display name of the custom COM component. For more information about these options, see “Action Subcomponent Reference” on page 157. In addition, you can choose: - `OutputPort`. The name of an `AdditionalOutputPort` where the data is written. For more information, see “Ports” on page 15. - `StandardErrorLog`. Writes to the user log. For more information, see “Failure Handling” on page 226.

DateAddICU

This action increments a date.

Table 10-15. Basic Properties

Property	Description
input_format	The date format, for example, dd/MM/yy. You can type the format or browse to a data holder containing the format. If you omit the format, the system default is assumed. For more information, see “DateFormatICU” on page 110.
input_date	The date to be incremented. You can type the date or browse to a data holder containing the date.
num_of_days	The number of days to add. You can type a positive or negative integer or browse to a data holder containing the number.
output	The data holder to store the output date.

Table 10-16. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Action Properties” on page 134.
remark	
disabled	
optional	
phase	

Note: DateAddICU replaces the DateAdd component used in previous versions of Conversion Agent. Existing transformations that use the DateAdd component continue to run without change.

DateDiffICU

This action computes the difference between two dates.

Table 10-17. Basic Properties

Property	Description
date_format1 date_format2	The formats of the two dates, for example, dd/MM/yy. You can type the format, or you can browse to a data holder containing the format. If you omit the format, the system default is assumed. For more information, see “DateFormatICU” on page 110.
date1 date2	The two dates. You can type the date or browse to a data holder containing the date.
output	The data holder to store the difference, in days.

Table 10-18. Advanced Properties

Property	Description
remark	For more information about these properties, see “Standard Action Properties” on page 134.
disabled	
optional	
phase	

Note: DateDiffICU replaces the DateDiff component used in previous versions of Conversion Agent. Existing transformations that use the DateDiff component continue to run without change.

DownloadFile

This action downloads a file to the local computer. The file path or URL is specified in a data holder, which a transformation might retrieve dynamically from a source document.

Table 10-19. Basic Properties

Property	Description
`file_url`	A data holder that stores the file path or URL.
`target_path`	The folder path to store the downloaded file. If you leave the property blank, the file is stored in the `Results` folder of the project.

Table 10-20. Advanced Properties

Property	Description
`transformers`	A sequence of transformers that the action applies to the path or URL string before downloading.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

Online Sample

For an online sample of this action, open the project `samples\Projects\DownloadFile\DownloadFile.cmw`. To run the sample, you must have an Internet connection. The sample retrieves the URL of a file. It then uses the `DownloadFile` action to download the file to the `Results` folder of the project.

DownloadFileToDataHolder

This action downloads a file from a web server and stores its content in a data holder.

If the file contains symbols such as < and >, the action converts them to XML entities such as `<` and `>`.

Table 10-21. Basic Properties

Property	Description
`file_url`	A data holder that stores the URL of the file.
`output`	The data holder to store the downloaded content.

Table 10-22. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

DumpValues

This action is a debugging tool. It writes data to a `<DumpValues>...</DumpValues>` element.

Nested in the action, insert the data holders that should be dumped.

Table 10-23. Advanced Properties

Property	Description
`output`	The file in which to write the output. The options are: - `ResultFile`. The default output file of the project. - `OutputFile`. Specify a path.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`phase`	

EnsureCondition

This action evaluates a JavaScript expression. If the expression is `false`, the action fails.

Table 10-24. Basic Properties

Property	Description
`condition`	A JavaScript expression to be evaluated. In the expression, use `$1`, `$2`, ... `$9`, to refer to the `params`. For example, the following expression checks whether the first parameter has the value `Ron Lehrer`: `$1 == "Ron Lehrer"`
`params`	A list of data holders, containing parameters that you can use in the condition.

Table 10-25. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`phase`	

JavaScript Syntax

The internal Conversion Agent JavaScript processor supports standard JavaScript expressions containing the following features:

- The unary and binary operators:

  ```
  () + - * / % == != < <= > >= && ||
  ```

- The ternary `?:` operator.
- The following methods:

  ```
  charAt
  indexOf
  lastIndexOf
  length
  substring
  toString
  ```

  If you apply these methods to a literal having a simple data type, you must enclose the literal in parentheses, for example:

  ```
  123.toString();                    //Wrong
  (123).toString();                  //Right

  "Hello, World".substring(3,7);     //Wrong
  ```

```
("Hello, World").substring(3,7);   //Right
```

- The following functions:

```
Math.ceil
Math.floor
Math.max
Math.min
Math.pow
Math.sqrt
parseFloat
parseInt
```

The internal JavaScript processor does not support features such as the following:

- The unary and binary operators:

```
++ -- typeof void >> >>> << === !== ~ & | ^
```

- Assignment operators:

```
= += -= *= /= >>= >>>= <<= &= |= ^=
```

- The comma operator `(,)`.
- The values `NaN`, `null`, `infinity`, or `-0` (negative 0).
- Data types other than string, number, and boolean.
- The `Date` object.
- The `equalsIgnoreCase` function.

Expressions that use these features can nonetheless be used in some cases. If the internal JavaScript processor cannot process an expression, the system automatically uses an external JavaScript processor supplied with Conversion Agent, to interpret the expression.

Note: Information about JavaScript syntax is available in many books about web development. For a tutorial introduction, see http://www.w3schools.com.

ExcludeItems

This action deletes specified values from a multiple-occurrence data holder. For more information, see “Multiple-Occurrence Data Holders” on page 64.

Nested in the action, specify the values to exclude.

Table 10-26. Basic Properties

Property	Description
`data_holder`	The multiple-occurrence data holder.

Table 10-27. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

ExternalCOMAction

The `ExternalCOMAction` component runs a custom action. You can implement the custom action as a COM (ActiveX) DLL or as a .NET DLL with the COM interoperability feature.

Because this component uses the Microsoft COM architecture to activate the custom action, it runs only on Microsoft Windows platforms.

To create a custom COM action:

1. In Microsoft Visual Studio, create a DLL project.

   If you use Microsoft Visual Basic 6, create an ActiveX DLL project containing a class module.

   If you use Microsoft Visual Studio .NET:

   - Create a class library project that references `ICMAction.dll` in the Conversion Agent installation folder.
   - Create a class that implements the `ICMAction` interface.
   - Configure the project with the COM interop feature.

1. In the class, implement the `Run` method.

   In Visual Basic, the syntax of the method is:

   ```
   Function Run(ByVal inp As String, ByVal design_mode As Boolean) As String
   ```

   In C#, the syntax is:

   ```
   public string Run(string inp, bool design_mode)
   ```

   The `inp` parameter is the input string that the action should process. The `ExternalCOMAction` component passes the input string to the function and receives the return value as output. The function can have any desired side effects, such as interacting with a third-party system.

   The `design_mode` parameter is `True` if the action is activated within Conversion Agent Studio. If the custom action requires a long processing time or has side effects that interfere while you are designing a parser, the function can perform different operations based on the `design_mode` value.

2. Register the DLL on the Conversion Agent computer.

   If you use Visual Basic 6, use the `regsvr32` command to register the component. If you use Visual Studio .NET, use the `regasm` command.

3. Define an `ExternalCOMAction` that references the ProgID of the DLL.

4. Optionally, add the `ExternalCOMAction` to the component list that Conversion Agent Studio displays. For more information about customizing the component list, see *Using Conversion Agent Studio in Eclipse*.

Table 10-28. Basic Properties

Property	Description
`COM`	A `COMClass` component specifying the ProgID of the custom action.
`input`	A data holder storing the input of the action.
`output`	A data holder where the action should store its output.

Table 10-29. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

Detailed Instructions for C#

For the convenience of Conversion Agent users who may not be familiar Visual Studio .NET, the following is a step-by-step procedure for implementing a custom action in the C# language. The instructions for other .NET languages, such as Visual Basic .NET, are similar.

To implement the custom action in C#:

1. Open Visual Studio .NET and create a new C# Class Library project.
2. Add a reference to the file `ICMAction.dll`.

   In the Add Reference window, you can find the reference on the .NET tab, component name `ICMAction`. Alternatively, browse to `ICMAction.dll` in the Conversion Agent installation folder.
3. Add a class that implements the `ICMAction` interface.

   You can copy the following sample code. Change the namespace and class names, `CMActionExample` and `CCMActionExample`, to meaningful names for your project.

```
using System;
using System.Runtime.InteropServices; //Enables COM interop
using Itemfield.ContentMaster;        //For ICMAction interface

namespace CMActionExample
{

  //Prevents automatic creation of class interface.
  //Causes class to be exported to COM only as an implementor
  //of the ICMAction interface
  [ClassInterface(ClassInterfaceType.None)]

  public class CCMActionExample : ICMAction
    {
    public CCMActionExample()
    {
    }

    public string Run(string inp, bool design_mode)
    {
    //ToDo: Insert code here
    }
  }
}
```

4. Implement the `Run` function, inserting code that performs the desired action.

   For example, the following implementation causes the custom action to count the characters in the input and return the result.

```
public string Run(string inp, bool design_mode)
{
  Int32 res = inp.Length;
  return res.ToString();
}
```

5. In the Solution Explorer, right-click the project and edit its properties.
   - In the left pane of the properties window, expand the tree and select Configuration Properties / Build.
   - In the right pane, in the Outputs section, set the Register for COM Interop property to true.
6. Right-click the project and click Build.

   This generates the DLL file.

On the computer where you developed the .NET project, Visual Studio .NET registers the DLL when you build the project. The DLL is ready to use in the `ExternalCOMAction` component.

To run the custom action in Conversion Agent on another computer, you must install the custom DLL as follows:

To install the DLL on another computer:

1. Confirm that Microsoft .NET Framework, version 1.1 or higher, is installed on the computer.
2. Copy the custom DLL to any convenient location on the computer, such as the Conversion Agent program folder.
3. Open a command prompt, and use the `regasm` utility to register the DLL. The utility is located in the Windows folder, in the subfolder `Microsoft.NET\Framework\<version>`.

   For example, enter the following command:

   ```
   regasm <path>\YourCustomDLL.dll /codebase
   ```

   The `regasm` utility displays a message indicating that the DLL was successfully registered.

Online Sample

For an online sample of a Visual Studio .NET project that implements a custom action in the C# language, see the following location in the Conversion Agent installation folder:

```
samples\SDK\CMACTION
```

JavaScriptFunction

This action executes a JavaScript function, for example, a function located in an HTML source document. You can pass parameters to the function, and you can store the return value of the function.

Table 10-30. Basic Properties

Property	Description
`function_to_execute`	The name of the function.
`result`	A data holder in which to store the return value of the function.
`params`	A list of data holders containing the input parameters of the function. The parameters must be in the same order as in the function declaration.

Table 10-31. Advanced Properties

Property	Description
`refresh`	If selected, Conversion Agent recompiles the function for each page that a parser processes. If not selected, Conversion Agent assumes that the function is the same on all the pages, and it compiles the function only on the first page.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

Map

This action copies a value from one data holder to another.

When copying a data holder that has a simple XSD data type, the source and destination must have compatible data types. The action can apply transformers to the copied value.

If you use the action to copy a data holder that has a complex type, the source and destination must have identical internal structures and identical XSD types. The action copies the nested elements and attributes.

Table 10-32. Basic Properties

Property	Description
source	The source data holder.
target	The destination data holder.
transformers	A sequence of transformers that modify the value. Do not assign this property if the source and destination are complex XML elements.

Table 10-33. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Action Properties” on page 134.
remark	
disabled	
optional	
phase	

Online Sample

For an online sample of this action, open the project `samples\Projects\CopyValue\CopyValue.cmw`. The sample uses a `Map` action to copy a complex element that contains an attribute and nested elements.

ODBCAction

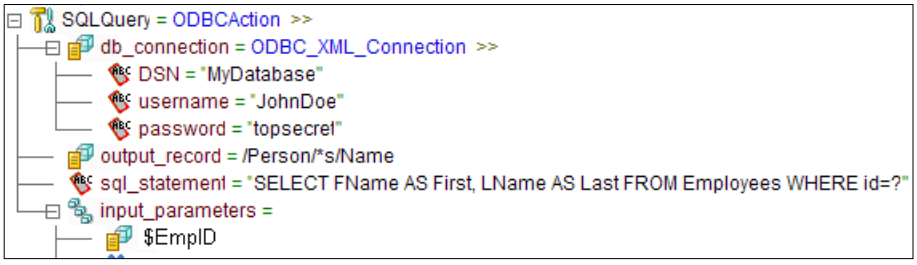
This action runs a SQL query on a database. For example, it can perform a `SELECT` query that retrieves data, or it can perform an `INSERT` or `UPDATE` query that adds data to the database.

Example

A source document contains an employee ID number. A parser retrieves the ID and stores it in a variable called `EmpID`. You want to retrieve the employee's name from a database and store the result in the following XML structure:

```
<Person>
  <Name>
    <First>...</First>
    <Last>...</Last>
  <Name>
</Person>
```

To do this, you can run an `ODBCAction`. Configure the action as follows:



In this example:

- The `db_connection` property defines the database connection.
- The `output_record` defines the data holder where the action should store the retrieved data.

- The `sql_statement` is the SQL query that retrieves the data.
- The `input_parameters` property contains the `EmpID` variable, which is the input of the action.

Table 10-34. Basic Properties

Property	Description
`db_connection`	An `ODBC_XML_Connection` component defining the ODBC provider, which is typically a database.

`sql_statement`	The SQL query, for example:
	`SELECT Name FROM Employees WHERE Id = ?`
	Use the `?` symbol to represent an input parameter. If there is more than one input parameter, each`?` symbol represents the next parameter in sequence, for example:
	`SELECT Name FROM Employees WHERE Id =? AND Gender =?`
	In this case, the two`?` symbols represent the first and second input parameters, respectively.
	The SQL syntax must be valid for the ODBC provider. Please see the provider or database documentation for details.

Table 10-35. Advanced Properties

Property	Description
`on_sql_no_data`	The behavior if the SQL query does not retrieve any data. The value can be: - `Success`. The action does not fail. - `Fail`.The action fails.
`output_record`	An XML element, defined in the XSD schema, where the action should store any data that the SQL query retrieves. The element must nested elements, at the top level of nesting, whose names are identical to the output fields of the query. If the SQL query retrieves multiple records, the schema must permit multiple occurrences of the XML element. For more information, see “Multiple-Occurrence Data Holders” on page 64.
`retry`	The number of retries if the first connection attempt fails.
`input_parameters`	A list of data holders that contain the input parameters.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

ResetVisitedPages

This action clears the list of visited pages of specified secondary parsers.

This action is used with the `reject_recurring_pages` property of a `Parser` component. `ResetVisitedPages` allows multiple visits to the same page, even if `reject_recurring_pages` is selected. You might do this, for example, if you want to post different input data to the same web page.

Table 10-36. Basic Properties

Property	Description
`parsers`	Specify the parsers to be reset.

Table 10-37. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`phase`	

RunMapper

This action runs a mapper.

For example, you can use this action in a parser to run a mapper that modifies the parser output.

Table 10-38. Basic Properties

Property	Description
`mapper`	The mapper. You can select the name of an existing `Mapper` component, or you can create a `Mapper` component at this location of the IntelliScript. For more information, see “Mappers” on page 179.
`input`	A data holder storing XML text on which to run the mapper. The data holder must have a simple data type such as `xs:string`. The value of the string can be XML text of any complexity. For more information about how to run a mapper on a data holder that has a complex type, see “EmbeddedMapper” on page 184. If you omit this property, the mapper uses the data holders available in the scope of the action. For example, if the action is nested in a parser, the mapper runs on the output of the parser. If the action is within a `Group`, it runs on the output of the `Group`.

Table 10-39. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

RunParser

This action runs a parser.

In a parser, for example, you can use this action to follow the links in an HTML file and run a secondary parser on the link destinations. In a serializer, you can use the action to parse bits of unstructured data that exist in the input.

The output of `RunParser` is appended to the output of the main component that activated it, such as a parser or serializer.

The `RunParser` action differs from the `EmbeddedParser` anchor, in that `RunParser` parses a new source, whereas `EmbeddedParser` parses a section of an existing source.

Example

An HTML file has a link to a second file. A `Content` anchor stores the file path or URL of the link destination in the `VarLinkURL` system variable. The `RunParser` action accesses the destination file and runs a secondary parser on it.

In another example, the main parser contains an `Alternatives` anchor that selects a secondary parser according to text in the source document. For more information, see the “Alternatives” on page 79.

Table 10-40. Basic Properties

Property	Description
`next_parser`	The name of the parser to run. Recursive calls to the same parser are permitted.

Table 10-41. Advanced Properties

Property	Description
`input_source_as_text`	This property specifies the type of data that the `input_source` data holder contains. If `input_source_as_text` is selected, `input_source` contains a text string that should be parsed. If not selected, `input_source` contains a file path or a URL.
`input_source`	If `input_source_as_text` is selected, `input_source` is a data holder that contains a string to be parsed. If `input_source_as_text` is not selected, `input_source` is a data holder containing the path or URL of the document to be parsed. The default value is the `VarLinkURL` system variable. If the `VarPostData` system variable contains a value, the value is posted to the URL. If `VarPostData` is empty, the action accesses the URL without posting any data.
`pre_processor`	A document processor that the parser should apply to the source.
`retries`	The number of times to retry if the request fails.
`seconds_to_wait`	The interval in seconds between retries.
`include_strings`	Strings that must be present in the `input_source` value. If a specified string is not present, the action does not access the source or activate the secondary parser. For example, if you want to follow links only within the same web site, you might add the domain name to `include_strings`.
`exclude_strings`	Strings that must not be present in the `input_source`. If a string is present, the action does not access the source or activate the secondary parser.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

Posting Data to a URL

Optionally, you can use the action to post data to a URL. This feature simulates the submission of an HTML form to a web server. The action activates a parser that processes the result returned by the web server.

To do this, you must store the data to be posted in the `VarPostData` system variable.

To prepare the data in VarPostData:

1. Save a copy of the HTML page containing the form on your local computer.
2. Edit the copy, changing the form action attribute to your email address. For example, if the form element reads `<form method="POST" action="http://example.com/MyServer.exe">`, change it to `<form method="POST" action="mailto:jdoe@example.com">`.
3. Open the copy in your browser, fill in the form, and click the submit button. This sends an email containing the form data to your address.

4. The body of the email is a string containing the form data. Assign this string to the `VarPostData` variable.

Note: Alternatively, you can use the `SubmitForm` or `SubmitFormGet` action to submit HTML form data to a URL.

RunSerializer

This action runs a serializer. The output of the serializer is stored in a data holder.

For example, a parser can use this action to run a serializer that modifies the parser output.

Table 10-42. Basic Properties

Property	Description
`serializer`	The serializer. You can select the name of an existing `Serializer` component, or you can create a `Serializer` at this location of the IntelliScript. For more information, see “Serializers” on page 163.
`input`	A data holder storing XML text on which to run the serializer. The data holder must have a simple data type such as `xs:string`. The value of the string can be XML text of any complexity. For more information about how to run a serializer on a data holder that has a complex type, see “EmbeddedSerializer” on page 174. If you omit this property, the serializer uses the data holders available in the scope of the action. For example, if the action is nested in a parser, the serializer runs on the output of the parser. If the action is within a `Group`, it runs on the output of the `Group`.
`output`	A data holder to store the serializer output.

Table 10-43. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

Online Sample

For an online sample of this action, open the project `samples\Projects\RunSerializer\RunSerializer.cmw`.

To observe how the sample works, set `MainParser` as the startup component and run it. `MainParser` contains a `RepeatingGroup` that parses pairs of names and stores them in variables. After each iteration, the `RepeatingGroup` executes a `RunSerializer` action that concatenates the variables with some predefined text. The action stores its output in an XML element that is added to the parser output.

SetValue

This action fills a data holder with predefined content.

The assignment overwrites any existing content, except for a multiple-occurrence data holder. For more information, see “Multiple-Occurrence Data Holders” on page 64.

Table 10-44. Basic Properties

Property	Description
`quote`	The content to assign.
`data_holder`	The data holder.

Table 10-45. Advanced Properties

Property	Description
`transformers`	A list of transformers that are applied to the content.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`phase`	

Sort

This action sorts the occurrences of a multiple-occurrence data holder. The output is saved to the original data holder. For more information, see “Multiple-Occurrence Data Holders” on page 64.

You can sort any multiple-occurrence data holder in the scope of the project, for example:

- The output of a parser
- The input of a serializer
- The input or output of a mapper
- A variable

If you run the action on an XML element that contains attributes or nested elements, you can use them as sort keys.

Limitation

You cannot use the `Sort` action if a `Key` is defined on the multiple-occurrence data holder. For more information, see “Locators, Keys, and Indexing” on page 187.

Table 10-46. Basic Properties

Property	Description
`recurring_element`	The multiple-occurrence data holder that should be sorted.
`by_fields`	The sort keys, in decreasing order of precedence. For each field, select the data holder and an `ascending` or `descending` sort. You can select the multiple-occurrence data holder itself, or any of its nested elements or attributes. To sort numerically, a sort key must have a numerical XSD type such as `xs:integer`.

Table 10-47. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`phase`	

SubmitForm

This action submits HTML form data to a URL and parses the response.

`SubmitForm` uses the HTTP Post method to submit the form. To use the HTTP Get method, use the `SubmitFormGet` action, instead. See also the `RunParser` action, which can submit an HTML form.

The output of `SubmitForm` is appended to the output of the main component that activated it, such as a parser or serializer.

SubmitForm is an alternative to using the HtmlForm anchor. HtmlForm is easier to use because it performs some of the data-preparation steps automatically. SubmitForm gives you greater control because it lets you configure these steps yourself.

To use the SubmitForm action:

1. Store the URL to which you want to submit the form in the VarFormAction system variable.

   The URL corresponds to the action attribute of an HTML <form> element,

2. Store the form data in the VarFormData system variable. You can determine the correct format of the data in the following way:
   - Save a copy of the HTML page containing the form on your local computer.
   - Edit the copy, changing the form action attribute to your email address. For example, if the form element reads <form method="POST" action="http":://example.com/MyServer.exe">, change it to <form method="POST" action="mailto:jdoe@example.com">.
   - Open the copy in your browser, fill in the form, and click the submit button. This sends an email containing the form data to your address.
   - The body of the email is a string containing the form data. Assign this string to the VarFormData variable.

3. Run the SubmitForm action. The action submits the data that you stored in VarFormData to the location that you stored in VarFormAction.

Submitting Multiple Copies of a Form

VarFormData is a multiple-occurrence variable. This means that you can create multiple occurrences of VarFormData, each storing a different set of post data.

If you do this, SubmitForm posts each occurrence of VarFormData independently, and it parses each of the web-server responses.

You can use the CombineValues action to prepare the VarFormData occurrences. For example, if you know the possible values of each form field, CombineValues can prepare all possible combinations of the values.

Table 10-48. Basic Properties

Property	Description
action	An OpenURL component specifying how to parse the web-server response. For more information, see “OpenURL” on page 158.

Table 10-49. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Action Properties” on page 134.
remark	
disabled	
optional	
phase	

Online Sample

For an online sample of this action, open the project samples\Projects\SubmitForm\SubmitForm.cmw. The sample works in the following way:

- The main parser, Flower_form_parser, retrieves options from the HTML order form of an online florist. The options include several flower types and price ranges.

- The parser uses a `CombineValues` action to prepare all possible combinations of the flower-type and price-range options.
- The parser uses a `SubmitForm` action to post the combinations to an web application.
- The `SubmitForm` action activates a secondary parser that parses the responses from the web application. The parsing output is added to the output of the main parser.

Note: You cannot run this sample because the web application does not exist.

SubmitFormGet

This action submits HTML form data to a URL and parses the response.

`SubmitFormGet` is identical to `SubmitForm`, except that it uses the HTTP Get method instead of Post. For more information, see "SubmitForm" on page 153.

This action writes the value of a data holder to a location such as a file, a database, or an MSMQ queue.

If the data holder is an XML element, the action writes both the element and any nested elements and attributes.

Table 10-50. Basic Properties

Property	Description
`input`	The data holder to write.
`output`	The output location. The options are: - `MSMQOutput`. Writes to an MSMQ queue. - `OutputDataHolder`. Writes to a data holder. - `OutputFile`. Writes to a file. - `ResultFile`. Writes to the default results file of the transformation. - `OutputCOM`. Uses a custom COM component to output the data. Do not select this option directly. Instead, select the display name of the custom COM component. - `ExternalOutputWriter`. Uses a custom component to output the data. - `ToOutputPort`. The name of an additional output port, in addition to standard output. For more information about these options, see the "Action Subcomponent Reference" on page 157.

Table 10-51. Advanced Properties

Property	Description
`no_tags`	By default, the action surrounds the value that it writes with XML tags. If you select `no_tags`, the XML tags are omitted. This is appropriate only if `input` is a simple data holder, containing no nested elements or attributes.
`transformers`	A list of transformers that modify the value before writing. The input to the transformers is the complete `input` data holder, including XML tags.
`name`	For more information about these properties, see "Standard Action Properties" on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

WriteValue

This action writes the value of a data holder to a location such as a file, a database, or an MSMQ queue.

If the data holder is an XML element, the action writes both the element and any nested elements and attribute

Table 10-52. Basic Properties

Property	Description
input	The data holder to write.
output	The output location. The options include: - MSMQOutput. Writes to an MSMQ queue. - OutputDataHolder. Writes to a data holder. - OutputFile. Writes to a file. - ResultFile. Writes to the default results file of the transformation. - OutputCOM. Uses a custom COM component to output the data. Do not select this option directly. Instead, select the display name of the custom COM component. For more information about these options, see “Action Subcomponent Reference” on page 157. In addition, you can choose: - OutputPort. The name of an AdditionalOutputPort where the data is written. For more information, see “Ports” on page 15. - StandardErrorLog. Writes to the user log. For more information, see “Failure Handling” on page 226.

Table 10-53. Advanced Properties

Property	Description
no_tags	By default, the action surrounds the value that it writes with XML tags. If you select no_tags, the XML tags are omitted. This is appropriate only if input is a simple data holder, containing no nested elements or attributes.
transformers	A list of transformers that modify the value before writing. The input to the transformers is the complete input data holder, including XML tags.
name	For more information about these properties, see “Standard Action Properties” on page 134.
remark	
disabled	
optional	
phase	

Online Samples

For an online sample of this action, open the project samples\Projects\Splitter\Splitter.cmw.

The sample demonstrates how to split a file into two files. A parser uses a RepeatingGroup to retrieve the records of an HL7 file. It uses a Map action to create unique filenames for each record, and a WriteValue action to write the records to the files. The output files, MyOutput1.txt and MyOutput2.txt, are stored in the Results folder of the project.

A practical use of the splitting technique is in sending large messages to an MSMQ queue for subsequent processing by Microsoft BizTalk Server. To avoid exceeding the MSMQ and BizTalk size limits, you can split the messages using a RepeatingGroup and WriteValue. For a sample, see samples\Projects\BizTalkSplitter\BizTalkSplitter.cmw.

Before you run the BizTalkSplitter example, use the MSMQ administration tools to create a queue where the parser will store its output. Edit the output property of the WriteValue action, inserting your queue path.

Note: Alternatively, you can use a streamer to split large inputs. For more information, see “Streamers” on page 203.

XSLTMap

This action runs an XSLT transformation.

The input and output are branches of an XML document. For example, they can be the output of a parser or the input of a serializer.

Example

Suppose that the following XML is the result of a parser:

```
<Person>
  <First>Ron</First>
  <Last>Lehrer</Last>
</Person>
```

You can use the `XSLTMap` action, with an appropriate XSLT file, to convert this to:

```
<Person Name="Lehrer, Ron" />
```

Table 10-54. Basic Properties

Property	Description
`input`	The XML element at the root of the branch to be transformed.
`output`	The XML element at the root of the branch that should store the output.
`xslt_file`	Browse to the XSLT file.

Table 10-55. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	
`phase`	

Action Subcomponent Reference

This section describes subcomponents that you can assign as the values of certain action properties.

COMClass

This subcomponent is used in `ExternalCOMAction` to define a custom COM component.

Table 10-56. Basic Properties

Property	Description
`ProgID`	The ProgID of the COM component. If you developed the custom action in Visual Basic 6, the ProgID typically has the form `dll_name.class`. If you used Visual Studio .NET with the COM interoperability option, the ProgID has the form `namespace.class`. For example, if you developed a .NET namespace with the name `CMActionExample`, and the class name is `CCMActionExample`, the ProgID is `CMActionExample.CCMActionExample`.

Table 10-57. Advanced Properties

Property	Description
`thread_safe`	Deselect this option if the COM component is incompatible with multithreading. This causes Conversion Agent to synchronize calls to the component.

MSMQOutput

This subcomponent specifies that a stream should be written to an MSMQ message and sent to a queue.

The subcomponent is used in the `WriteValue` action to specify the output location.

Table 10-58. Basic Properties

Property	Description
`output_id`	The MSMQ queue identifier, such as path. You can type the identifier or browse to a data holder that contains the identifier.

Table 10-59. Advanced Properties

Property	Description
`append`	This property is not in use.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	

ODBC_XML_Connection

The subcomponent defines a database connection. It is used, for example, in an `ODBCAction`.

Before using this component, use the operating system tools to define a DSN for the database connection.

Table 10-60. Advanced Properties

Property	Description
`DSN`	The data source name of the connection.
`username`	User name for the database connection.
`password`	Password of the user.
`timeout`	Time in seconds to wait for the database response.

OpenURL

This subcomponent is used within the `SubmitForm` and `SubmitFormGet` actions to specify how to parse a web-server response.

Table 10-61. Basic Properties

Property	Description
`next_parser`	The name of a parser to run on the web-server response.

Table 10-62. Advanced Properties

Property	Description
`retries`	The number of retries if the first request fails.
`seconds_to_wait`	The interval in seconds between retries.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	
`disabled`	
`optional`	

OutputCOM

The `OutputCOM` option of the `WriteValue` action allows you to use a custom COM component to create output from Conversion Agent. The component can perform any desired operations, such as modifying the data, writing the data to multiple locations, or interacting with an information system.

Because `OutputCOM` uses the Microsoft COM technology, it operates only on Microsoft Windows systems.

The `OutputCOM` component works differently from most other Conversion Agent components. It is a template for a custom component, and not a component that you can use directly. You cannot configure an action with the `OutputCOM` option and nest a custom component within `OutputCOM`. Instead, you must program a custom COM component, add it to the drop-down list, and select its name in the `output` property of `WriteValue`. The following paragraphs explain the procedure.

Programming the Custom COM Component

To create the custom COM component, program an ActiveX DLL containing the following function:

```
Public Function process_output( _
    ByVal output_id As String, _
    ByVal outContent As String, _
    ByVal mode As String) _
    As String
```

The action passes the following parameters to the function:

Parameter	Description
`output_id`	An identifier for the desired output location. The value of this parameter is the value of the `output_id` property of the `OutputCOM` component.
`outContent`	The content that the action is outputting.
`mode`	If the `append` property of the component is not selected in the IntelliScript, `mode = "CREATE"`. If the `append` property is selected, `mode = "APPEND"`.

The function can perform any desired operations. Conversion Agent ignores the return value of the function.

Install and register the DLL on the Conversion Agent computer.

Adding the Custom COM Component to the Drop-Down List

You must add the custom component to the drop-down list that Conversion Agent displays under the action.

To add the component to the drop-down list:

1. In Notepad, create a text file.
2. Type a line such as the following in the file:

   ```
   profile DisplayName ofPT OutputCOMT("MyProject.MyClass")
   ```

   Here, *`DisplayName`* is the name that you would like to display in the drop-down list, and *`MyProject.MyClass`* is the ProgID of the component.
3. Save the file with an extension of `*.tgp`, for example, `MyClass.tgp`, in the program subdirectory `ConversionAgent\AutoInclude\User`.

Using the Custom COM Component

To use the custom COM component:

1. Configure an action.
2. In the `output` property of the action, select the *`DisplayName`* that you configured above.
3. Assign the `output_id` and `append` properties of the `DisplayName` component.

You can then run the transformation. The action activates the custom component.

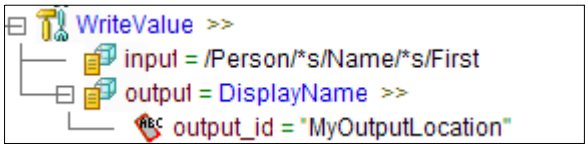


Table 10-63. Basic Properties

Property	Description
`output_id`	An identifier for the location where the custom COM component should write its output. You can use this parameter, for example, to pass the name of an output file to the custom component. You can type an identifier, or you can browse to a data holder that contains the identifier.

Table 10-64. Advanced Properties

Property	Description
`append`	If selected, the custom COM component should append its output to the existing content of the output location rather than overwriting it.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	

OutputDataHolder

This subcomponent specifies how to write a stream to a data holder. The subcomponent is used in the `WriteValue` action to specify the output location.

Table 10-65. Basic Properties

Property	Description
`data_holder`	The data holder.

Table 10-66. Advanced Properties

Property	Description
`transformers`	A sequence of transformers that modify the stream before writing.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	

OutputFile

This subcomponent specifies that a stream should be written to a file.

The subcomponent is used in the `DumpValues` and `WriteValue` actions to specify the output location.

Table 10-67. Basic Properties

Property	Description
`file`	The filename, optionally including a path. You can type the name, or you can browse to a data holder that contains the name. The path can be absolute or relative. In the latter case, Conversion Agent resolves the path relative to the output folder of the transformation. If you run the transformation within Conversion Agent Studio, the path is relative to the project `Results` folder.

Table 10-68. Advanced Properties

Property	Description
`append`	If selected, the data is appended to the existing content of the file, rather than overwriting it.
`name`	For more information about these properties, see “Standard Action Properties” on page 134.
`remark`	

ResultFile

This subcomponent specifies that a stream should be written to the normal output file of a project.

The subcomponent is used in the `DumpValues` and `WriteValue` actions to specify the output location.

CHAPTER 11

Serializers

This chapter includes the following topics:

Creating a Serializer

Serialization is the opposite of parsing. A parser converts a source document from any format to an XML file. A serializer converts an XML file to an output document in any format. For example, the output of a serializer can be a text document, an HTML document, or even another XML document.

You can create a serializer by any of the following methods:

- By inverting the configuration of an existing parser
- By using the New Parser wizard
- By editing the IntelliScript and inserting a `Serializer` component

You can combine these methods. For example, you can invert a parser and edit the IntelliScript of the resulting serializer.

It is usually easier to create a serializer than a parser. This is because the XML input is completely structured. The structure makes it easy to identify the required data and write it, in a sequential procedure, to the output. A parser, in contrast, may need to process unstructured or semi-structured input, a task that is often more complex than serialization.

The main components that are nested in a serializer are called serialization anchors. The function of the serialization anchors is to identify the XML data and write it to the output. Serialization anchors are analogous to the anchors that are used in a parser, except that they work in the opposite direction.

Creating a Serializer by Inverting a Parser

You can create serializer by inverting the configuration of an existing parser. For example, if you have a parser that transformers tab-delimited text to XML, you can create a serializer that transformers the XML to tab-delimited text.

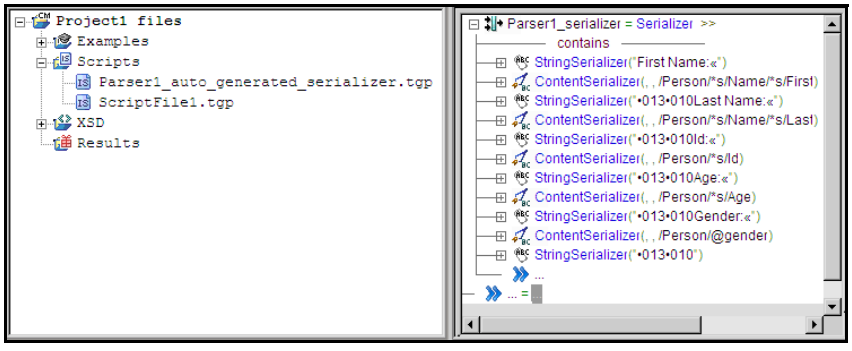
To create a serializer automatically from a parser:

1. In Conversion Agent Studio, open an existing parser in an IntelliScript editor.
2. Right-click the parser and click Create Serializer.

   Conversion Agent notifies you that the serializer has been created, and it displays the serializer in the IntelliScript.

   The name of the serializer is derived from that of the parser, with the suffix `_serializer`. For example, if you create a serializer from `Parser1`, the serializer is called `Parser1_serializer`.

   Conversion Agent stores the serializer in a new TGP script file, which has a name such as `Parser1_auto_generated_serializer.tgp`. Use the Conversion Agent Explorer or the Component view to open the new serializer file in an IntelliScript editor.



3. Test the serializer. For more information, see “Running a Serializer” on page 168.

   If necessary, edit the IntelliScript. For more information, see “Troubleshooting an Auto-Generated Serializer” on page 166.

Online Sample

For an example of an auto-generated serializer, open `samples\Projects\Serialization\TabDelimited\TabDelimited.cmw`.

To run the sample:

1. Set `MyHL7Parser` as the startup component, and run it. This generates an output file `Results\output.xml`.
2. Now set `MyHL7Parser_serializer` as the startup component, and run it. At the prompt, browse to `Results\output.xml` as the input. The original input file is regenerated.

A variant of this project is in `samples\Projects\Serialization\HL7\HL7.cmw`. You can generate the serializer yourself and try the above experiment.

Controlling How the Create Serializer Command Works

When you run the Create Serializer command, Conversion Agent converts the `Content` anchors of the parser to `ContentSerializer` serialization anchors.

By default, the command converts all other text in the example source to `StringSerializer` serialization anchors. Assuming that the other text is boilerplate content, this means that the output of the serializer contains all the boilerplate that was in the original example source.

For example, suppose the parser runs on tab-delimited source documents having the following structure:

```
Name (first and last):<tab>Ron Lehrer
```

Assume that the anchors are defined in the following way:

Source text	Anchor
`Name`	`Marker`
`(first and last):<tab>`	Not marked as an anchor
`Ron Lehrer`	`Content`

The XML output of the parser is:

```
<FullName>Ron Lehrer<FullName>
```

Now, generate a serializer from this parser, and run the serializer on the following input:

```
<FullName>Larissa Chan<FullName>
```

The output of the serializer is:

```
Name (first and last):<tab>Larissa Chan
```

Serialization Mode

The example source might contain text that you don't want in the serializer output. In that case, you can modify the behavior of the Create Serializer command in a way that does not generate the `StringSerializer` serialization anchors.

To do this, set the `serialization_mode` property of the `Parser` component. The possible values of the `serialization_mode` are explained in the following table.

Value	Description
`Full`	The Create Serializer command copies the non-XML text to the serializer configuration. This is the default behavior.
`Outline`	The Create Serializer command copies only the delimiters of the non-XML text to the serializer configuration. Under the `Outline` option, you can select the `use_markers` option. This causes the Create Serializer command to copy the content of the `Marker` anchors but only the delimiters of other non-XML text.

The following table illustrates the results of the `serialization_mode` settings.

serialization_mode	Behavior	Sample Serializer Output
`outline` With `use_markers` not selected	The Create Serializer command converts: - `Content` anchors to ContentSerializer serialization anchors - The delimiters of other text in the example source to `StringSerializer` serialization anchors	`<tab>Larissa Chan`
`outline` With `use_markers` selected	The Create Serializer command converts: - `Content` anchors to `ContentSerializer` serialization anchors - The complete text of `Marker` anchors to `StringSerializer` serialization anchors - The delimiters of other text in the example source to `StringSerializer` serialization anchors	`Name<tab>Larissa Chan`
`full`	The Create Serializer command converts: - `Content` anchors to `ContentSerializer` serialization anchors - All other text in the example source to `StringSerializer` serialization anchors	`Name (first and last):<tab>Larissa Chan`

Troubleshooting an Auto-Generated Serializer

Often, you can use an automatically generated serializer directly. If required, you can edit the auto-generated serializer to correct any limitations or problems that you find in it.

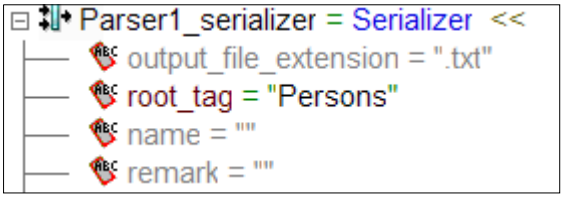
The following paragraphs list some typical circumstances under which you need to edit the serializer, and the suggested editing steps.

Root Tag

On the XML Generation tab of the project properties, there is an option to Add XML Root Element. The effect of this option is to nest the parser output in a specified root element. For more information, see “XML Generation Properties” on page 219.

If this option is selected, and you try to run an auto-generated serializer on the parser output, it cannot find the input XML elements because of the nesting.

The solution is to set the `root_tag` property of the serializer to the same value as in the project properties. The serializer then finds its input nested under the root.



Variables

If the parser uses a variable to store intermediate results, an auto-generated serializer may fail. To solve the problem, review the serializer logic, and remove the variable if necessary.

Additional Components

The Create Serializer command inverts the anchors of a parser. It does not invert components such as document processors, transformers, or actions.

For example, suppose that a parser uses a `PdfToTxt_3_01` document processor to convert PDF source documents to text. The parser contains anchors that transform the text to XML.

The auto-generated serializer transforms the XML back to text. It does not convert the text to PDF. You can obtain PDF output by submitting the text to the Adobe Acrobat Distiller or to any other PDF generation utility.

In another example, suppose that a parser uses an `AddString` transformer to add a prefix to the output of a `Content` anchor. The auto-generated serializer does not remove the prefix. If you need to remove it, you can edit the serializer and insert a component such as a `Replace` transformer.

Creating a Serializer by Using the New Serializer Wizard

You can use the New Serializer wizard to create a serializer.

To create a new project that contains a serializer:

1. Click File > New > Project.
2. Under the Conversion Agent category, select a Serializer Project and click Next.
3. Follow the wizard prompts to enter the serializer options.

   When you finish, the Conversion Agent Explorer view displays the new project containing the serializer. The Component view displays the serializer.

To create a new serializer in an existing project:

1. Click File > New > Serializer.
2. Follow the wizard prompts to enter the serializer options.

   When you finish, the Conversion Agent Explorer view displays a new TGP script file defining the serializer. The Component view displays the serializer.

The following table describes the wizard options.

Option	Description
Serializer name	A name for the serializer.
Script name	A name for a TGP script file where the wizard stores the serializer definition.
Schema file path	The name of an XSD schema defining the XML syntax of the serializer input.

To complete the serializer configuration:

1. Display the serializer in an IntelliScript editor.
2. Under the `contains` line, add a sequence of serialization anchors and actions.
3. Run and test the serializer, and modify the IntelliScript as required.

   For more information, see “Running a Serializer” on page 168.

Creating a Serializer by Editing the IntelliScript

Like all other Conversion Agent components, you can create a serializer by editing the IntelliScript directly.

To create a serializer in the IntelliScript:

1. At the top level of the IntelliScript, select the three dots (. . .) symbol. Press Enter and type a name for the serializer.
2. To the right of the name, press Enter. Select a `Serializer` component from the list.
3. Expand the tree under the `Serializer` component. Assign its properties as required.
4. If necessary, add an XSD schema defining the XML syntax of the serializer input.

   For more information, see “Data Holders” on page 51.
5. Under the `contains` line, add a sequence of nested serialization anchors and actions.

   For more information, see “Serialization Anchors” on page 168 and “Actions” on page 133.
6. Run and test the serializer and modify the IntelliScript as required.

   For more information, see “Running a Serializer” on page 168.

Online Sample

For an example of a serializer that we created by editing the IntelliScript, open the project `samples\Projects\ManualSerializer\ManualSerializer.cmw`. You can run the serializer on the input file `Example XML of Person.xml`.

Creating a Serializer within a RunSerializer Action

In addition to defining a serializer at the global level, it is possible to define a serializer within a `RunSerializer` action.

Running a Serializer

To run a serializer in Conversion Agent Studio:

1. Set the serializer as the startup component.
2. Click Run > Run.
3. Browse to the input XML file.

   If you created the serializer from a parser, a convenient test file is the parser output. Browse to the output file, by default `Results\output.xml`, in the project folder.

   Alternatively, you can set the `example_source` property of the serializer. This lets you test a serializer repeatedly on the same input, without needing to browse to the file each time.
4. When the execution is complete, Conversion Agent Studio displays the Events view. Examine the events for any failures or warnings.
5. To view the serialization results, open the output file, located in the `Results` folder of the project.

Serialization Anchors

The main components that you can use in a serializer are called serialization anchors. These are analogous to the anchors that are used in a parser, except that they work in the opposite direction. Anchors read data from locations in the source document and write the data to XML. Serialization anchors read XML data and write the data to locations in the output document.

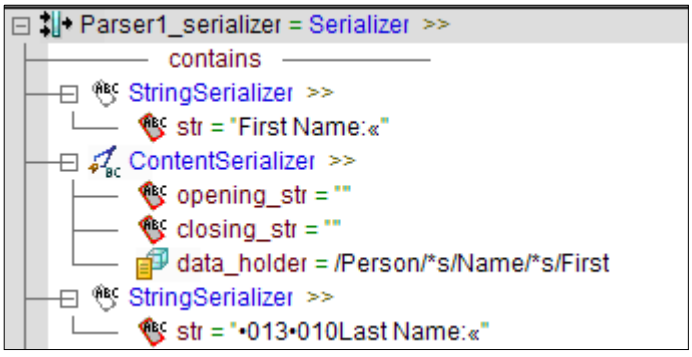
Please note that a serialization anchor is not an anchor, despite their similar names. You cannot use anchors in a serializer, and you cannot use serialization anchors in a parser.

The most important serialization anchors are `ContentSerializer` and `StringSerializer`:

- A `ContentSerializer` writes the content of a specified data holder to the output document. It is the inverse of a `Content` anchor, which reads content from a source document.
- A `StringSerializer` writes a predefined string to the output. It is the inverse of a `Marker` anchor, which finds a predefined string in a source document.

Example of Serialization Anchors

The following example illustrates three serialization anchors.



The first `StringSerializer` instructs the serializer to write the following text in the output document:

```
First Name:<tab>
```

The `ContentSerializer` writes the value of the `Person/Name/First` element to the output.

The second `StringSerializer` writes the string:

```
<newline>Last Name:<tab>
```

Note: The IntelliScript represents the newline and tab using ASCII codes and «, respectively. For more information about entering special characters in the IntelliScript Editor, see *Using Conversion Agent Studio in Eclipse*.

Now, assume that you run the serializer on the following XML:

```
<Person gender="M">
  <Name>
    <First>Ron</First>
    <Last>Lehrer</Last>
  </Name>
  <Id>547329876</Id>
  <Age>27</Age>
</Person>
```

From the illustrated serialization anchors, the output is:

```
First Name<tab>Ron<newline>Last Name<tab>
```

The display of this text is:

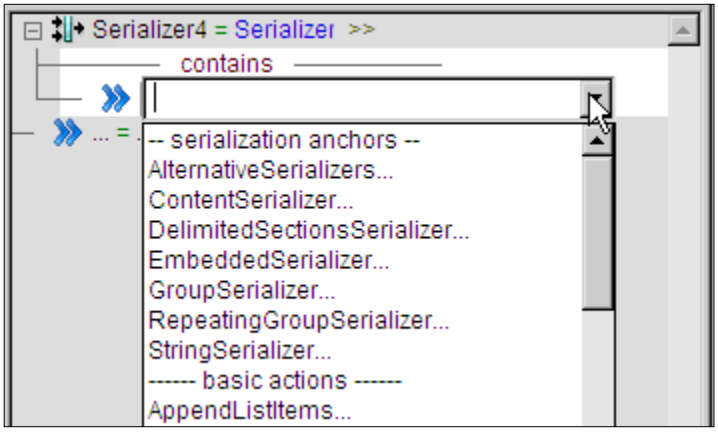
```
First Name:      Ron
Last Name:
```

The serializer contains additional serialization anchors, which are not shown in the above illustration. The complete output of the serializer is:

```
First Name:      Ron
Last Name:       Lehrer
Id:      547329876
Age:     27
Gender: M
```

Defining Serialization Anchors

To define serialization anchors, edit the IntelliScript under a `Serializer` component.



Sequence of Serialization Anchors

A serializer executes the serialization anchors in the sequence of their definitions.

Serialization anchors write data sequentially, always appending it to the end of the output document. You can alter the order by changing the sequence in the serializer configuration.

You can intersperse actions with the serialization anchors. The actions are executed as part of the sequence.

Standard Serializer Properties

In this section, we review certain standard properties that are found in the `Serializer` component and in many serialization anchors. For more information about the properties of specific components, see the “Serializer Component Reference” on page 170 and “Serialization Anchor Component Reference” on page 171.

Property	Description
`name`	A name that you assign to the component. Conversion Agent includes the name in the event log. This can help you find an event that was caused by the particular component.
`remark`	A comment describing the component.
`disabled`	If selected, Conversion Agent ignores the component. This is useful for testing and debugging, or for making minor modifications in a project without deleting the existing components.
`optional`	By default, if a component fails, its parent component fails. If you select the `optional` property, the parent component does not fail. For more information, see “Failure Handling” on page 226.
`on_fail`	If the component fails, writes an entry in the user log. For more information, see “Failure Handling” on page 226.

Serializer Quick Reference

The main serialization component is:

Component	Description
`Serializer`	Converts XML to an output document format.

Within a `Serializer` component, you can nest the following serialization anchors:

Serialization Anchor	Description
`AlternativeSerializers`	Specifies alternative serialization anchors that may be appropriate, depending on the structure of the XML.
`ContentSerializer`	Serializes XML data and writes it to the output document.
`DelimitedSectionsSerializer`	Serializes sections of data, writing a separator string between them.
`EmbeddedSerializer`	Runs a secondary serializer.
`GroupSerializer`	Binds a set of serialization anchors together for processing as a unit.
`RepeatingGroupSerializer`	Creates a repetitive structure in the output document.
`StringSerializer`	Writes a specified string to the output document.

Serializer Component Reference

This section documents the top-level `Serializer` component. For more information about serialization anchors, see “Serialization Anchor Component Reference” on page 171.

Serializer

A `Serializer` converts XML documents to output documents in any format.

Table 11-1. Advanced Properties

Property	Description
`validate_source_document`	The level of source XML validation that the serializer performs. The options are: - `Partial`. Permits some deviations from the schema. - `Strict`. Enforces the schema strictly. For more information, see “Role of XSD in Serialization and Mapping” on page 60.
`example_source`	A sample XML source document. When you run the serializer in Conversion Agent Studio, it operates on the sample document. The value of the property is an input port. For more information, see “Ports” on page 15. If you leave the `example_source` property blank, Conversion Agent prompts you for a source document when you run the serializer. Nested within the `example_source`, you can assign a preprocessor that converts the source documents to a format that the serializer can accept. For example, the `example_source` might be a Microsoft Excel workbook configured with the `ExcelToXml` preprocessor. For more information, see “Document Processors” on page 21.
`output_file_extension`	The file extension of the generated output file, including the leading period, for example: `.txt`
`root_tag`	The name of a root XML element that is not in the XSD schema of the project. For example, if the top-level element of the schema is `Person`, but the XML input nests `Person` in an element called `InputWrapper`, enter `root_tag = InputWrapper`.
`default_transformers`	A list of transformers that the `Serializer` applies to all serialized data.
`source` `target`	These properties are useful in situations where the serializer must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.
`name`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`remark`	
`on_fail`	

Serialization Anchor Component Reference

This section describes the serialization anchors that you can use in a `Serializer`.

AlternativeSerializers

This serialization anchor lets you define a set of alternative, nested serialization anchors. You can define a criterion for the alternative that the serializer should accept. Only the accepted alternative affects the serializer output. The other serialization anchors, whether failed or successful, have no effect on the serializer output.

Example

The input XML might contain a `Product` element or a `Service` element, but not both. You want to serialize whichever element is in the input.

Define an `AlternativeSerializers` serialization anchor, and set its `selector` property to `ScriptOrder`.

Within the `AlternativeSerializers`, nest two `ContentSerializer` serialization anchors. Configure one of them to process the `Product` element and the other to process `Service`.

Table 11-2. Basic Properties

Property	Description
`selector`	The criterion for deciding which alternative to accept. The options are: - `ScriptOrder`. Conversion Agent tests the nested serialization anchors in the sequence that they are defined in the IntelliScript. It accepts the first one that succeeds. If all the nested serialization anchors fail, the `AlternativeSerializers` component fails. - `NameSwitch`. Conversion Agent searches for the nested serialization anchor whose `name` property is specified in a data holder. It ignores the other nested serialization anchors. If the named serialization anchor fails, the `AlternativeSerializers` component fails.

Table 11-3. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`remark`	
`disabled`	
`optional`	
`on_fail`	

ContentSerializer

This serialization anchor writes the serialized data to the output document.

Table 11-4. Basic Properties

Property	Description
`opening_str`	A string that the anchor should write before the `data_holder`.
`closing_str`	A string that the anchor should write after the `data_holder`.
`data_holder`	The data holder containing the data.

Table 11-5. Advanced Properties

Property	Description
`allow_empty_values`	If selected, the `data_holder` can be empty. If not selected, and the `data_holder` is empty, the `ContentSerializer` fails.
`ignore_default_transformers`	If selected, the default transformers of the `Serializer` are not applied to the serialized data.
`transformers`	A list of transformers that are applied to the serialized data.
`name`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`remark`	
`disabled`	

Table 11-5. Advanced Properties

Property	Description
optional	
on_fail	

DelimitedSectionsSerializer

This serialization anchor processes sections of data. Between each section of the output, the DelimitedSectionsSerializer writes a separator string.

Within the DelimitedSectionsSerializer, nest other serialization anchors. Each nested serialization anchor is responsible for outputting a single section.

Example

The XML input contains an employee resume. You wish to write the data to an output text document in the following format:

```
----------------------------
Jane Palmer
Employee ID 123456
----------------------------
Professional Experience
...
----------------------------
Education
...
```

Define a DelimitedSectionsSerializer with the line of hyphens as its separator. Because you want a line of hyphens before each section, set separator_position = before.

Within the DelimitedSectionsSerializer, nest three GroupSerializer components. The first GroupSerializer writes the Jane Palmer section, the second writes the Professional Experience section, and so forth.

Optional Sections

In the above example, suppose that the second section, Professional Experience, is missing from some input XML documents. You nonetheless want to write its separator to the output, like this:

```
----------------------------
Jane Palmer
Employee ID 123456
----------------------------
----------------------------
Education
...
```

To support this situation, configure the DelimitedSectionsSerializer in the following way:

- In the second GroupSerializer, select the optional property. This means that if the GroupSerializer fails, it should not cause the DelimitedSectionsSerializer to fail.
- In the DelimitedSectionsSerializer, set using_placeholders = always. This means to write the separator of an optional section, even if the section itself is missing.

Alternatively, suppose that if the Professional Experience section is missing, you do not want to write its separator:

```
----------------------------
Jane Palmer
Employee ID 123456
----------------------------
Education
...
```

In this case, configure the `DelimitedSectionsSerializer` as follows:

- In the second `GroupSerializer`, select the `optional` property.
- In the `DelimitedSectionsSerializer`, set `using_placeholders = never`. This means not to write the separator of a missing section.

Table 11-6. Basic Properties

Property	Description
`separator`	The separator string.
`separator_position`	Position of the `separator` relative to the sections. The options are `before`, `after`, `between`, and `around`.
`using_placeholders`	This property specifies whether the `DelimitedSectionsSerializer` should write the separator of an optional section that is missing from the XML input. The options are `always`, `never`, and `when necessary`.

The following table illustrates the possible values of the `separator_position` property. The examples assume that the `separator` is a vertical-line character (`|`).

separator_position	Explanation	Example
`before`	Write a separator before each section, including the first sections.	`|1|2|3|4`
`after`	Write a separator after each section, including the first sections.	`1|2|3|4|`
`between`	Write a separator between the successive sections, but not before the first section and not after the last section.	`1|2|3|4`
`around`	Write separators before and after each section, including the first sections.	`|1|2|3|4|`

The following table illustrates the possible values of the `using_placeholders` property. The examples assume that the `separator_position` is `before` and that sections `2` and `4` are missing.

using_placeholders	Explanation	Example
`always`	Always write the separator of a missing section.	`|1||3|`
`never`	Never write the separator of a missing section.	`|1|3`
`when necessary`	Always write the separator of a missing internal section. Never write the separator of a missing terminal section.	`|1||3`

Table 11-7. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`remark`	
`disabled`	
`optional`	
`on_fail`	

EmbeddedSerializer

This serialization anchor activates a secondary `Serializer`, which writes its output in the same output document.

Example

The XML input is a family tree. The input contains `Person` elements, which are recursively nested as shown:

```
<Person>       <!-- Parent -->
  ...
  <Person>     <!-- Child -->
    ...
    <Person>   <!-- Grandchild -->
      ...
    </Person>
  </Person>
</Person>
```

A `Serializer` can use an `EmbeddedSerializer` component to call itself recursively, until all levels of nesting are exhausted.

Table 11-8. Basic Properties

Property	Description
`serializer`	The name of the secondary serializer. The serializer must be defined at the global level of the IntelliScript.
`schema_connections`	Connects the data holders that are referenced in the secondary serializer to the data holders that are referenced in the main serializer. The property contains a list of `Connect` subcomponents that define the correspondence. For more information, see “Connect” on page 99. If all the data holders in the main and secondary serializers are identical, you can omit this property. If there are any differences between the data holders, you must connect the data holders explicitly, even the ones that are identical. In the recursive example described above, `Person` should be connected to `Person/Person`. This instructs the secondary instance of the serializer to process a nested level of the input. It is sufficient to connect just the parent element (`Person`), and not the nested elements `(Person/*s/Name, Person/*s/BirthDate`, etc.), provided that the two `Person` elements have the same XSD type.

Table 11-9. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`remark`	
`disabled`	
`optional`	
`on_fail`	

GroupSerializer

The `GroupSerializer` serialization anchor binds its nested serialization anchors together. You can set properties of the `GroupSerializer` that affect the members of the group.

Table 11-10. Basic Properties

Property	Description
`source` `target`	These properties are useful in situations where the serialization anchor must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.

Table 11-11. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Serializer Properties” on page 170.
remark	
disabled	
optional	
on_fail	

RepeatingGroupSerializer

This serialization anchor writes a repetitive structure to the output document.

A RepeatingGroupSerializer is useful if the XML data contains a multiple-occurrence data holder. It iterates over the occurrences of the data holder and outputs the data. For more information, see “Multiple-Occurrence Data Holders” on page 64.

Within the RepeatingGroupSerializer, nest serialization anchors that process and output each occurrence of the data holder. Optionally, you can define a separator that the RepeatingGroupSerializer writes to the output between the iterations.

Example

The XML input contains the following structure:

```
<Persons>
  <Person>
    <Name>John</Name>
    <Age>35</Age>
  </Person>
  <Person>
    <Name>Larissa</Name>
    <Age>42</Age>
  </Person>
  ...
</Persons>
```

A RepeatingGroupSerializer, using a newline character as a separator, can output this data to:

```
John      35
Larissa   42
```

You can iterate over several multiple-occurrence data holders in parallel. For example, you can iterate over a list of men and a list of women, and output a list of married couples. To do this, insert a ContentSerializer within the repeating group for each data holder.

Table 11-12. Basic Properties

Property	Description
separator	A serialization anchor, typically a StringSerializer, that outputs the separator. Leave this property empty if you do not want to output a separator.
separator_ position	Position of the separator relative to the iterations. The options are before, after, between, and around.

The following table illustrates the possible values of the `separator_position` property. The examples assume that the `separator` is a vertical-line character (|).

separator_position	Explanation	Example
before	Write a separator before each iteration, including the first iteration.	|1|2|3
after	Write a separator after each iteration, including the last one.	1|2|3|
between	Write a separator between the successive iterations, not before the first iteration and not after the last iteration.	1|2|3
around	Write separators before and after each iteration, including the first and last iterations.	|1|2|3|

Table 11-13. Advanced Properties

Property	Description
count	The number of iterations to run. Enter a number, or click the browse button and select a data holder that contains the number. If blank, the iterations continue until the input is exhausted.
current_iteration	A data holder, where the `RepeatingGroupSerializer` should output the number of the current iteration. You can use a `ContentSerializer` to write the number to the output.
source target	These properties are useful in situations where the serialization anchor must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.
on_iteration_fail	If an iteration fails, writes an entry in the user log. User the `on_fail` property to write an entry if the entire `RepeatingGroupSerializer` fails. Use `on_iteration_fail` to write an entry if a single iteration fails. For more information, see “Failure Handling” on page 226.
name	For more information about these properties, see “Standard Serializer Properties” on page 170.
remark	
disabled	
optional	
on_fail	

StringSerializer

This serialization anchor writes a predefined string to the output document.

Table 11-14. Basic Properties

Property	Description
str	The string to write.

Table 11-15. Advanced Properties

Property	Description
name	For more information about these properties, see “Standard Serializer Properties” on page 170.
remark	
disabled	
on_fail	

CHAPTER 12

Mappers

This chapter includes the following topics:

Creating a Mapper

Mappers are components that convert an XML source document to another XML structure or schema.

A mapper processes the XML input like a serializer, and it generates the XML output like a parser. Because both the input and the output are fully structured XML, the configuration is straightforward.

The principles of mapper operation are similar to those of a serializer. For more information, see "Serializers" on page 163.

Within a mapper, you can nest mapper anchors and actions. Mapper anchors are analogous to anchors, which are used in parsers, and to serialization anchors, which are used in serializers.

This chapter explains how to configure the mapper and mapper anchor components.

To create a mapper:

1. Add XSD input and output schemas to the project.

   It is permitted to use either the same schema or different schemas for the input and the output. For more information, see "Data Holders" on page 51.

2. At the top level of the IntelliScript, add a `Mapper` component.

3. Assign the `source` and `target` properties of the `Mapper` to the input and output elements of the `Mapper`, respectively.

   For more information, see "Locators, Keys, and Indexing" on page 187

4. Edit the other properties of the `Mapper` as required.

5. Nest a sequence of actions and mapper anchors within the `Mapper`.

   For more information, see “Components Nested within a Mapper” on page 180.

6. Test the mapper and modify the IntelliScript if required.

   For more information, see “Running a Mapper” on page 181.

Creating a Mapper within a RunMapper Action

In addition to defining a mapper at the global level, it is possible to define a mapper within a `RunMapper` action. For more information, see “Actions” on page 133.

Components Nested within a Mapper

Within a `Mapper`, you can nest the following components:

- Any number of `Map` actions. The actions retrieve a data holder from the output and write the content to the output.
- Optionally, any number of mapper anchors. For more information, see the “Mapper Anchor Component Reference” on page 183.

The `Map` actions and the mapper anchors can be in any sequence. You can also insert other actions in the sequence.

Notice that a mapper uses `Map` actions rather than mapper anchors to write to the output XML. This may seem a little different from parsers and serializers, where the output is created by anchors and serialization anchors, respectively. Actually, this is just a terminology issue. The `Map` action could have been defined as a mapper anchor. It is defined as an action because it is useful in other circumstances, unrelated to mappers.

Mapper Example

To illustrate the mapper configuration, we present a simple example.

Source XML

The input of the mapper is an XML document containing a list of personal names and their associated ID numbers.

```
<Persons>
  <Person ID="10">Bob</Person>
  <Person ID="17">Larissa</Person>
  <Person ID="13">Marie</Person>
</Persons>
```

Output XML

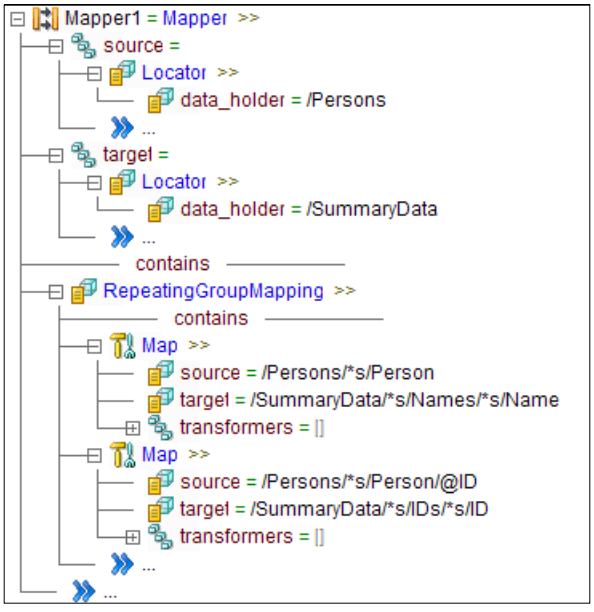
The desired output of the mapper is an XML list of the names and ID numbers, with no association between them.

```
<SummaryData>
  <Names>
    <Name>Bob</Name>
    <Name>Larissa</Name>
    <Name>Marie</Name>
  </Names>
```

```
    <IDs>
      <ID>10</ID>
      <ID>17</ID>
      <ID>13</ID>
    </IDs>
  </SummaryData>
```

Mapper Configuration

The following mapper configuration performs the desired transformation:



The `RepeatingGroupMapping` iterates over the `Person` elements of the input. It uses `Map` actions to write the data to the `Name` and `ID` elements of the output.

Running a Mapper

To run a mapper in Conversion Agent Studio:

1. Set the mapper as the startup component.
2. Click Run > Run.

   You are prompted to open the input XML file.

   Alternatively, you can set the `example_source` property of the mapper. This lets you test a mapper repeatedly on the same input, without needing to browse to the file each time.
3. When the execution is complete, Conversion Agent Studio displays the Events view. Examine the events for any failures or warnings.
4. View the mapping results by opening the `output.xml` file, located in the `Results` folder of the project.

Standard Mapper Properties

In this section, we review certain standard properties that are found in the `Mapper` component and in many mapper anchors. For more information about the properties of specific components, see the “Mapper Component Reference” on page 182 and the “Mapper Anchor Component Reference” on page 183.

Property	Description
`name`	A name that you assign to the component. Conversion Agent includes the name in the event log. This can help you find an event that was caused by the particular component.
`remark`	A comment describing the component.
`disabled`	If selected, Conversion Agent ignores the component. This is useful for testing and debugging, or for making minor modifications in a project without deleting the existing components.
`optional`	By default, if a component fails, its parent component fails. If you select the `optional` property, the parent component does not fail. For more information, see “Failure Handling” on page 226.
`on_fail`	If the component fails, writes an entry in the user log. For more information, see “Failure Handling” on page 226.

Mapper Quick Reference

The main mapping component is:

Component	Description
`Mapper`	Converts XML to XML.

Within a `Mapper` component, you can nest the following mapper anchors:

Mapper Anchor	Description
`AlternativeMappings`	Defines a set of nested mappings, one of which is valid for the current XML context.
`EmbeddedMapper`	Activates a secondary mapper.
`GroupMapping`	Binds its nested mapper anchors and actions together.
`RepeatingGroupMapping`	Maps repetitive XML structures.

Mapper Component Reference

This section documents the top-level `Mapper` component. For more information about mapper anchors, see the “Mapper Anchor Component Reference” on page 183.

Mapper

A `Mapper` performs XML to XML transformations. It converts a source XML document to an output document that has a different XML structure.

You must use the `source` and `target` properties to identify the root elements of the XML documents. For example, if the document element of the source is `Persons`, and the document element of the output is `SummaryData`, set the `source` and `target` as follows:

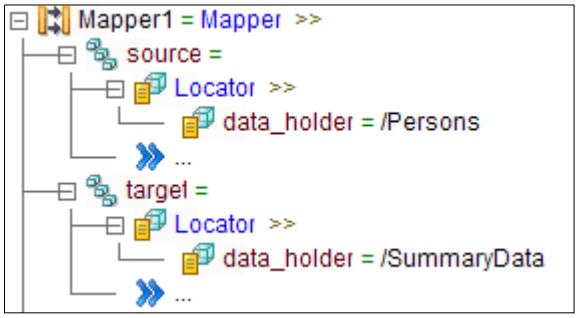


Table 12-1. Basic Properties

Property	Description
`source`	Under this property, insert a `Locator` component, and select the root of the source XML from a Schema view. For more information about this property, see “Locators, Keys, and Indexing” on page 187.
`target`	Under this property, insert a `Locator` component, and select the root of the output XML from a Schema view. For more information about this property, see “Locators, Keys, and Indexing” on page 187.

Table 12-2. Advanced Properties

Property	Description
`validate_source_document`	The level of source XML validation that the mapper performs. The options are: - `Partial`. Permits some deviations from the schema. - `Strict`. Enforces the schema strictly. For more information, see “Role of XSD in Serialization and Mapping” on page 60.
`example_source`	A sample XML source document. When you run the mapper in Conversion Agent Studio, it operates on the sample document. The value of the property is an input port. For more information, see “Ports” on page 15. If you leave the `example_source` property blank, Conversion Agent prompts you for a source document when you run the mapper. Nested within the `example_source`, you can assign a preprocessor that converts the source documents to a format that the mapper can accept. For example, the `example_source` might be a Microsoft Excel workbook configured with the `ExcelToXml` preprocessor. For more information, see “Document Processors” on page 21.
`root_tag`	The name of a root XML element that is not in the XSD schema of the input. For example, if the top-level element of the schema is `Person`, but the XML input nests `Person` in an element called `InputWrapper`, enter `root_tag = InputWrapper`.
`name`	For more information about these properties, see “Standard Mapper Properties” on page 182.
`remark`	
`on_fail`	

Mapper Anchor Component Reference

This section describes the mapper anchors.

AlternativeMappings

This mapper anchor lets you define a set of alternative, nested mapper anchors. You can define a criterion for the alternative that the mapper should accept. Only the accepted alternative affects the mapper output. The other mapper anchors, whether failed or successful, have no effect on the mapper output.

Example

The input XML may contain a `Product` element or a `Service` element, but not both. You wish to process whichever element is in the input.

Define an `AlternativeMappings` mapper anchor, and set its `selector` property to `ScriptOrder`.

Within the `AlternativeMappings`, nest two `Map` actions. Configure one of them to process the `Product` element and the other to process `Service`.

Table 12-3. Basic Properties

Property	Description
`selector`	The criterion for deciding which alternative to accept. The options are: - `ScriptOrder`. Conversion Agent tests the nested mapper anchors in the sequence that they are defined in the IntelliScript. It accepts the first one that succeeds. If all the nested mapper anchors fail, the `AlternativeMappings` component fails. - `NameSwitch`. Conversion Agent searches for the nested mapper anchor whose `name` property is specified in a data holder. It ignores the other nested mapper anchors. If the named mapper anchor fails, the `AlternativeMappings` component fails.

Table 12-4. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`remark`	
`disabled`	
`optional`	
`on_fail`	

EmbeddedMapper

This mapper anchor activates a secondary `Mapper`, which stores its output in the same output document.

Example

The XML input is a family tree. The input contains `Person` elements, which are recursively nested as shown:

```
<Person>       <!-- Parent -->
  ...
  <Person>     <!-- Child -->
    ...
    <Person>   <!-- Grandchild -->
      ...
    </Person>
  </Person>
</Person>
```

A `Mapper` can use an `EmbeddedMapper` component to call itself recursively, until all levels of nesting are exhausted.

Table 12-5. Basic Properties

Property	Description
`mapper`	The name of the secondary mapper.
`schema_connections`	Connects the data holders that are referenced in the secondary mapper to the data holders that are referenced in the main mapper. The property contains a list of `Connect` subcomponents that define the correspondence. For more information, see “Connect” on page 99. If all the data holders in the main and secondary mappers are identical, you can omit this property. If there are any differences between the data holders, you must connect the data holders explicitly, even the ones that are identical. In the recursive example described above, `Person` should be connected to `Person/Person`. This instructs the secondary instance of the mapper to process a nested level of the input. It is sufficient to connect just the parent element (`Person`), and not the nested elements (`Person/*s/Name, Person/*s/BirthDate`, etc.), provided that the two `Person` elements have the same XSD type.

Table 12-6. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`remark`	
`disabled`	
`optional`	
`on_fail`	

GroupMapping

The `GroupMapping` mapper anchor binds its nested mapper anchors and actions together. You can set properties of the `GroupMapping` that affect the members of the group.

Table 12-7. Basic Properties

Property	Description
`source` `target`	These properties are useful in situations where the mapper anchor must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.

Table 12-8. Advanced Properties

Property	Description
`name`	For more information about these properties, see “Standard Mapper Properties” on page 182.
`remark`	
`disabled`	
`optional`	
`on_fail`	

RepeatingGroupMapping

This mapper anchor processes a repetitive structure in the input or output.

A `RepeatingGroupMapping` is useful if the XML input and/or output contains a multiple-occurrence data holder. It iterates over occurrences of the data holders. For more information, see “Multiple-Occurrence Data Holders” on page 64.

Within the `RepeatingGroupMapping`, nest mapper anchors and actions that process each occurrence of the data holder.

Example

For more information, including an example of a `RepeatingGroupMapping`, see the “Mapper Example” on page 180.

Table 12-9. Advanced Properties

Property	Description
`count`	The number of iterations to run. Enter a number, or click the browse button and select a data holder that contains the number. If blank, the iterations continue until the input is exhausted.
`current_iteration`	A data holder, where the `RepeatingGroupMapping` should output the number of the current iteration.
`source` `target`	These properties are useful in situations where the mapper anchor must select specific occurrences of data holders. For more information, see “Locators, Keys, and Indexing” on page 187.
`name`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`on_iteration_fail`	If an iteration fails, writes an entry in the user log. User the `on_fail` property to write an entry if the entire `RepeatingGroupMapping` fails. Use `on_iteration_fail` to write an entry if a single iteration fails. For more information, see “Failure Handling” on page 226.
`name`	For more information about these properties, see “Standard Mapper Properties” on page 182.
`remark`	
`disabled`	
`optional`	
`on_fail`	

CHAPTER 13

Locators, Keys, and Indexing

This chapter includes the following topics:

Overview

In designing a transformation, a frequent issue is how to locate the data holders that you want to process. If the same data holders can occur multiple times in an XML structure, there can be ambiguities in identifying the occurrences. This chapter explains how to use the `Locator` and `Key` components to resolve the ambiguities.

The components described in this chapter let you identify the occurrences of multiple-occurrence data holders in three ways:

- Sequentially. Each iteration of a component processes the next occurrence of the data holder.
- By occurrence number. For example, a component can select the third occurrence of a data holder.
- By a key such as an attribute or a nested element. The key uniquely identifies the occurrence of the data holder.

The sequential approach is the default. It is subject to some complexities that you can control by using the `Locator` component.

The occurrence number and key approaches are collectively known as indexing. The term is analogous to the index of a book, where you use a page number or a subject key to identify the location of information. You can implement the indexing by using components called `LocatorByOccurrence`, `LocatorByKey`, and `Key`.

You can use the locator and key components in parsers, serializers, or mappers. You can use the components to identify the occurrences of data holders in the input, the output, or both.

The locator components are nested in the `source` and `target` properties of various transformation components. The meaning and usage of the `source` and `target` properties is explained below.

Example of Locators

To understand the issues involved in identifying data holders, consider the following example. The example illustrates the use of:

- The `target` property
- The `Locator` component

We will explain the broad outline of the example here. In the following sections of the chapter, we will go back and explain how the `target` and the `Locator` work in detail.

Input and Output

Suppose that the output schema of a parser supports the following structure:

```
<Report>
  <Company>
    <Employee>John</Employee>
    <Employee>Leslie</Employee>
    <Employee>Pedro</Employee>
  </Company>
  <Company>
    <Employee>Marie</Employee>
    <Employee>Larry</Employee>
    <Employee>Frances</Employee>
  </Company>
</Report>
```

The source document that the parser processes is a list containing a single employee per company:

```
John
Marie
```

The output of the parser should be:

```
<Report>
  <Company>
    <Employee>John</Employee>
  </Company>
  <Company>
    <Employee>Marie</Employee>
  </Company>
</Report>
```

Incorrect Solution

Suppose you use the following `RepeatingGroup` to parse the source document:

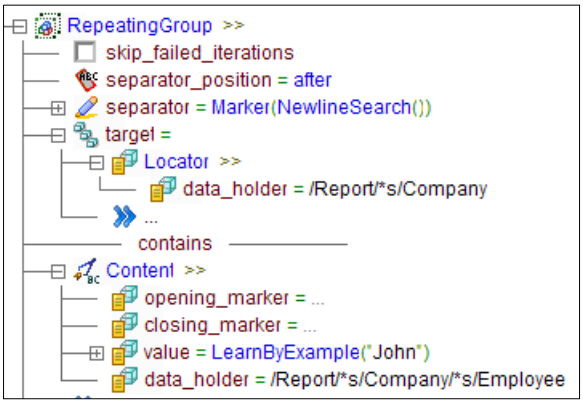
The output is incorrect:

```
- <Report>
  - <Company>
      <Employee>John</Employee>
      <Employee>Marie</Employee>
    </Company>
  </Report>
```

The problem is that both `Company` and `Employee` are multiple-occurrence elements. The `RepeatingGroup` creates multiple `Employee` elements correctly, but it does not know that each `Employee` element should be nested in a separate `Company` element.

Correct Solution

To resolve the ambiguity, you can assign the `target` property of the `RepeatingGroup`.



The `target` identifies the data holder that the `RepeatingGroup` should create. The `target` contains a `Locator` component pointing to the `Company` element. This means that each iteration of the `RepeatingGroup` should create a new occurrence of the `Company` element.

If you configure the `RepeatingGroup` in this way, the output is correct:

```
- <Report>
  - <Company>
      <Employee>John</Employee>
    </Company>
  - <Company>
      <Employee>Marie</Employee>
    </Company>
  </Report>
```

Example of Indexing by Key

To further introduce the data-holder identification issues, we present an example of indexing by key.

The example is a mapper that uses indexing to identify the occurrences of data holders in both its input and its output. On the input side, the indexing matches the corresponding data from different parts of an XML structure. On the output side, the indexing finds the correct location of an element in an XML structure.

The example illustrates the use of:

- The `source` and `target` properties
- The `Locator`, `Key`, and `LocatorByKey` components

In the following sections of the chapter, we will explain the detailed operation of these properties and components.

Input

The input XML is a report listing the names of parents and their children.

- For each parent, the XML lists a first name, a last name, and an ID.
- For each child, the XML lists a first name, a hobby, and the ID of the parent.

```
<Report>
  <Parents>
    <Parent id="1" firstName="John" lastName="Smith"/>
    <Parent id="2" firstName="Jane" lastName="Doe"/>
  </Parents>
  <Children>
    <Child name="Eric" hobby="Swimming" parentID="1"/>
    <Child name="Elizabeth" hobby="Biking" parentID="2"/>
    <Child name="Mary" hobby="Painting" parentID="1"/>
    <Child name="Edward" hobby="Swimming" parentID="2"/>
  </Children>
</Report>
```

Output

The desired output is a list of hobbies and the children who engage in each hobby.

```
<Hobbies>
  <Hobby name="Swimming">
    <Person firstName="Eric" lastName="Smith"/>
    <Person firstName="Edward" lastName="Doe"/>
  </Hobby>
  <Hobby name="Biking">
    <Person firstName="Elizabeth" lastName="Doe"/>
  </Hobby>
  <Hobby name="Painting">
    <Person firstName="Mary" lastName="Smith"/>
  </Hobby>
</Hobbies>
```

Outline of the Transformation Approach

The transformation uses the following approach:

1. In the input XML, the transformation identifies the corresponding `Child` and `Parent` elements as follows:

   `id` attribute of `Parent` = `parentID` attribute of `Child`

2. The transformation creates `Hobby` and `Person` elements. It identifies the `Hobby` element where it should nest each `Person` element as follows:

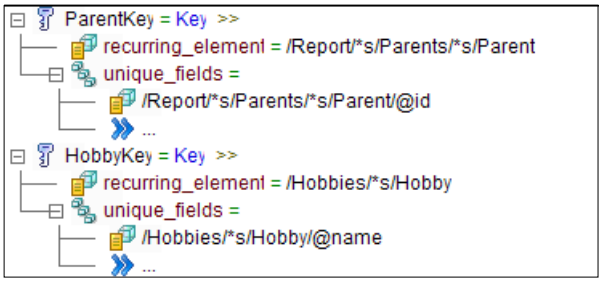
   `name` attribute of `Hobby` = `hobby` attribute of `Child`

3. The transformation writes the child's first name into the `Person` element.
4. The transformation writes the parent's last name into the `Person` element.

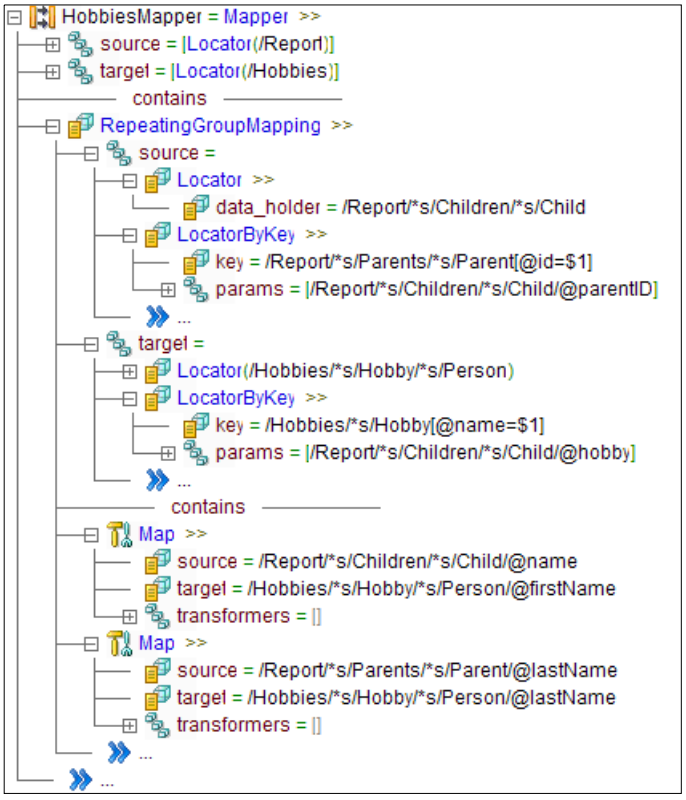
Mapper Configuration

The IntelliScript uses `Key` components to define identifiers for the data holders:

- The first `Key` specifies that the `id` attribute is a unique identifier of a `Parent` element.
- The second `Key` specifies that the `name` attribute is a unique identifier of a `Hobby` element.



The IntelliScript then defines a `Mapper` having the following configuration:



The components of the `Mapper` configuration are described below.

1. The `source` property of the `RepeatingGroupMapping` specifies that each iteration should obtain its input from two data holders:
   - From an occurrence of the `Child` element
   - From the corresponding occurrence of the `Parent` element
2. The `target` property of the `RepeatingGroupMapping` specifies that each iteration should store its output in two data holders:
   - In an occurrence of the `Person` element
   - In the corresponding occurrence of the `Hobby` element
3. The first `Map` action copies the `name` attribute of the `Child` to the `firstName` attribute of the `Person`.
4. The second `Map` action copies the `lastName` attribute of the `Parent` into the `lastName` attribute of the `Person`.

Use of Indexing

The example uses indexing by key to identify the occurrences of the `Parent` and `Hobby` data holders.

- In the `source` property of the `RepeatingGroupMapping`, the indexing identifies the occurrence of `Parent` that corresponds to a `Child`.
- In the `target` property, the indexing identifies the occurrence of `Hobby` where a `Person` should be nested.

Source and Target Properties

The `source` and `target` properties exist in components such as the following:

- In parsers:

```
Parser
Group
RepeatingGroup
EnclosedGroup
FindReplaceAnchor
```

- In serializers:

```
Serializer
GroupSerializer
RepeatingGroupSerializer
```

- In mappers:

```
Mapper
GroupMapping
RepeatingGroupMapping
```

In all these categories, the meaning and usage of the properties is identical:

- The `source` property identifies existing data holders that a transformation should use.
- The `target` property identifies data holders that may or may not already exist. If they exist, the transformation uses them. If they do not exist, the transformation creates them.

After you define the `source` and/or the `target`, the subsequent components use the identified data holders. For example, if you define the `target` of a `Group`, the anchors nested within the `Group` use the data holders that the `target` identifies.

Note: There are properties called `source` and `target` in some other components such as `Map`. These properties have a different meaning and usage from the above. For an explanation, please see the components where the properties are used.

Source Property

The `source` property identifies existing occurrences of data holders. The value of the `source` property is a list containing one or more of the following components:

Source	Description
`Locator`	Identifies a single-occurrence or multiple-occurrence data holder. In the latter case, each iteration accesses the next occurrence, in sequence.
`LocatorByKey`	Identifies an occurrence of a multiple-occurrence data holder by using a key.
`LocatorByOccurrence`	Identifies an occurrence of a multiple-occurrence data holder by number.

Default Behavior

If you do not assign the `source` property of a component, the component identifies data holders in the following way:

- If there is only one occurrence of the data holder, the component uses the existing occurrence.
- If there are multiple occurrences of the data holder, the behavior is as follows:
  - In an iterative context, such as within a `RepeatingGroupSerializer`, each iteration accesses the next occurrence of the data holder in sequence.
  - In a non-iterative context, such as a `GroupSerializer` that is not nested within an iterative component, the component accesses the first occurrence of the data holder.

Ambiguities in the Default Behavior

There can be some ambiguities in the default behavior. Ambiguities can arise, for example, in the following circumstances.

- In cases where a multiple-occurrence element is nested within another multiple-occurrence element. For more information, see “Example 1: Nested Multiple-Occurrence Data Holders” on page 193.
- In cases where the XSD schema permits alternative data holders, defined with `xs:choice`.
- In cases where the XSD schema permits a data holder to be missing, defined with `minOccurs = 0`.

In such cases, it is prudent to assign the `source` property explicitly.

Data Holder Must Exist

The `source` property identifies a data holder that already exists in the scope of the transformation. If the data holder does not exist, the component containing the `source` property fails.

For example, suppose that the source property of a `Group` contains a non-optional `LocatorByOccurrence` that points to the third occurrence of a data holder. If only two occurrences exist, the `Group` fails.

Using the Source Property for Input or Output

Typically, a component uses the `source` property to identify where it should obtain input. For example, a `GroupSerializer` can use the property to identify an occurrence that it should serialize.

It is also possible to use the property to identify where the component should store output. For example, suppose that a parser has already created 10 occurrences of an XML element. After the occurrences have been created, a `Group` anchor assigns an attribute in one occurrence of the element. The `Group` can use the `source` property to identify the occurrence.

Example 1: Nested Multiple-Occurrence Data Holders

Suppose that the input schema of a serializer supports the following structure:

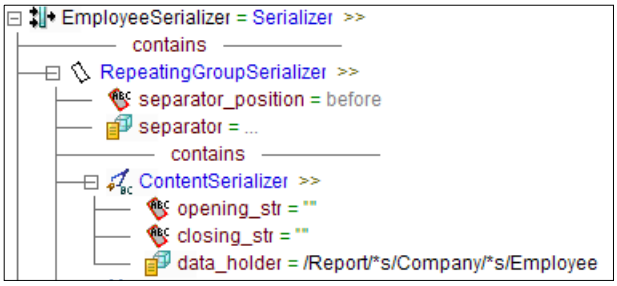
```
<Report>
  <Company>
    <Employee>John</Employee>
    <Employee>Leslie</Employee>
    <Employee>Pedro</Employee>
  </Company>
  <Company>
    <Employee>Marie</Employee>
    <Employee>Larry</Employee>
    <Employee>Frances</Employee>
  </Company>
</Report>
```

You want to iterate over all the `Employee` elements and produce the following output:

```
John
Leslie
```

```
Pedro
Marie
Larry
Frances
```

At first thought, you might create a `RepeatingGroupSerializer` and configure it to output the `Employee` data holder:

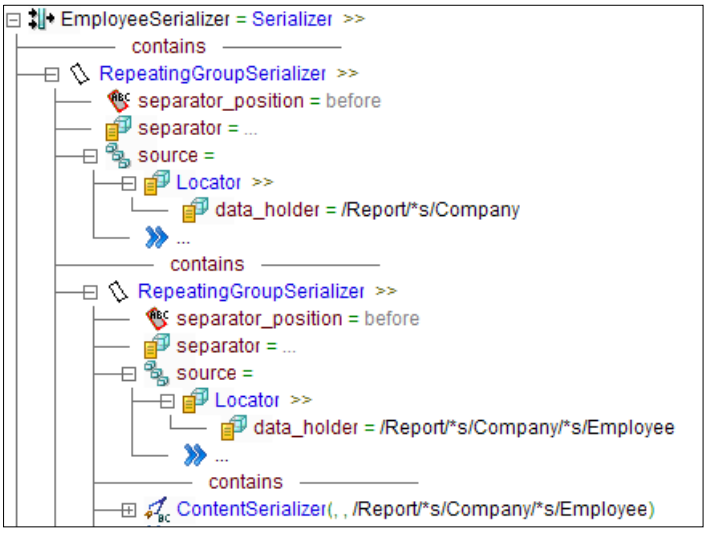


This does not work correctly! By default, each iteration selects a new instance of `Employee` within the same `Company`. The result is the output:

```
John
Leslie
Pedro
```

In other words, the `RepeatingGroupSerializer` accesses only the first `Company`.

You can solve the problem by nesting the `RepeatingGroupSerializer` inside another `RepeatingGroupSerializer`. To resolve any potential ambiguities, you can configure the `source` properties explicitly:



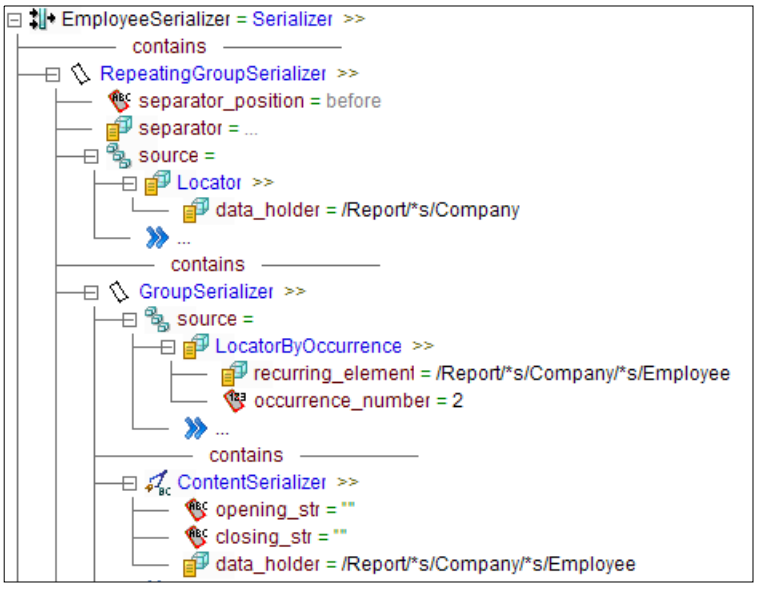
Each iteration of the outer `RepeatingGroupSerializer` processes a different occurrence of `Company`. Each iteration of the nested `RepeatingGroupSerializer` processes a different occurrence of `Employee`. The result is the desired output.

Alternatively, suppose you want to iterate only over the second `Employee` element in each `Company`. The desired output is:

```
Leslie
Larry
```

You can do this by configuring a single `RepeatingGroupSerializer`, whose source is `Company`. This causes each iteration to access the next instance of `Company`. Within the iteration, you can configure a

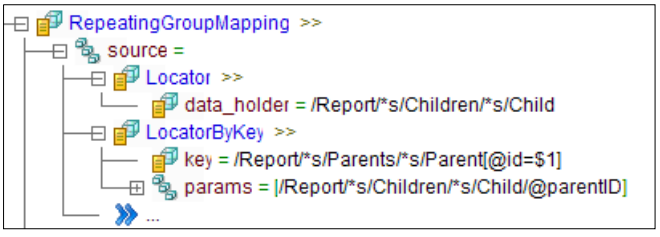
GroupSerializer, whose source property uses a LocatorByOccurrence to select the second Employee. This generates the desired output.



Example 2: Indexing

In the Example of Indexing by Key at the beginning of this chapter, we used a RepeatingGroupMapping configured as shown below. In this example, the source property identifies two data holders:

- It uses a Locator component to identify an occurrence of Child. Each iteration processes the next occurrence of Child, sequentially.
- It uses a LocatorByKey component to identify an occurrence of Parent. This causes each iteration to process the occurrence of Parent that corresponds to the occurrence of Child.



Target Property

The target property identifies an occurrence of a data holder that may or may not already exist. If the occurrence exists, the component uses it. If the occurrence does not exist, the component creates it.

The value of the target property is a list containing one or more of the following components:

Target	Description
Locator	Identifies a single-occurrence or multiple-occurrence data holder. In the latter case, each iteration creates a new occurrence.
LocatorByKey	Identifies an occurrence of a multiple-occurrence data holder by an indexing key. If the occurrence does not yet exist, it is created.
LocatorByOccurrence	Identifies an occurrence of a multiple-occurrence data holder by number. If the occurrence does not yet exist, it is created along with any needed intervening occurrences. For example, if four occurrences exist, and LocatorByOccurrence specifies the tenth occurrence, occurrences 5-9 are also created, but left empty.

Default Behavior

If you do not assign the `target` property of a component, the component identifies data holders in the following way:

- If the schema permits only a single occurrence of the data holder, Conversion Agent accesses or creates the occurrence.
- If the data holder can have multiple occurrences, the behavior is as follows:
  - In an iterative context, for example, within a `RepeatingGroup`, each iteration creates a new occurrence of the data holder.
  - In a non-iterative context, for example, a `Group` that is not nested within an iterative component, the component creates one new occurrence of the data holder.

Ambiguities in the Default Behavior

There can be some ambiguities in the default behavior. Ambiguities can arise, for example, in the following circumstances.

- In cases where a multiple-occurrence element is nested within another multiple-occurrence element. For more information, see “Example 1: Nested Multiple-Occurrence Data Holders” on page 196.
- In cases where the XSD schema permits alternative data holders, defined with `xs:choice`.
- In cases where the XSD schema permits a data holder to be missing, defined with `minOccurs = 0`.

In such cases, it is prudent to assign the `target` property explicitly.

Data Holder Can Be Created

The `target` property identifies a data holder that may or may not already exist in the scope of the transformation. If the data holder does not exist, it is created.

For example, suppose that the `target` property of a `Group` contains a `LocatorByKey`, which points to a particular occurrence of a data holder. If the occurrence already exists, the `Group` uses it. If the occurrence does not exist, the `Group` creates it.

Using the Target Property for Input or Output

Typically, a component uses the `target` property to identify where it should store output. For example, a `Group` can use the property to identify an occurrence where it should store data.

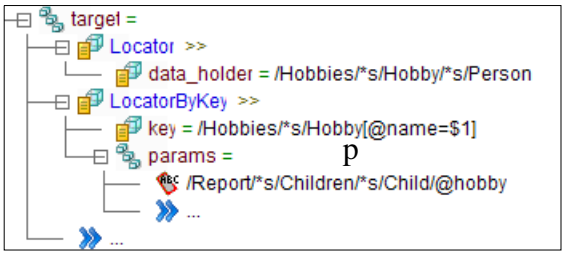
It is also possible to use the property to identify where a component should obtain input. For example, suppose that a `GroupSerializer` contains an action that computes data and stores it in a variable. The `GroupSerializer` then activates a `ContentSerializer` that writes the variable to the output. You can use the `target` property to create the occurrence of the variable that the `GroupSerializer` uses. The variable then serves as the input of the `ContentSerializer`.

Example 1: Nested Multiple-Occurrence Data Holders

The Example of Locators, at the start of this chapter, illustrates how to use the `target` property to differentiate between parent and child multiple-occurrence data holders.

Example 2: Indexing

The Example of Indexing by Key, at the start of this chapter, illustrates how to use the `target` property with indexing. The `target` property of the `RepeatingGroupMapping` is configured as follows:



The `target` property identifies two data holders:

- It uses a `Locator` component to identify an occurrence of `Person`. Each iteration creates a new occurrence of `Person`.
- It uses a `LocatorByKey` component to identify the occurrence of the `Hobby` element, where the occurrence of `Person` should be nested. If the `Hobby` element already exists, the transformation uses it. If the `Hobby` element does not yet exist, the transformation creates it.

Standard Locator and Key Properties

In this section, we review certain standard properties that are used in the locator and key components. For more information about the properties of specific components, see the "Locator and Key Component Reference" on page 198.

Property	Description
`disabled`	If selected, Conversion Agent ignores the component. This is useful for testing and debugging, or for making minor modifications in a project without deleting the existing components.
`optional`	By default, if a component fails, its parent component fails. If you select the `optional` property, the parent component does not fail.
`remark`	A comment describing the component.

Locator and Key Quick Reference

The locator and key components are:

Component	Description
`Key`	Defines a unique identifier for a data holder.
`Locator`	Identifies a single-occurrence or multiple-occurrence data holder.
`LocatorByKey`	Identifies an occurrence of a multiple-occurrence data holder by using a key.
`LocatorByOccurrence`	Identifies an occurrence of a multiple-occurrence data holder by number.

Locator and Key Component Reference

This section documents the locator and key components that are available in Conversion Agent.

Key

A `Key` defines attributes or elements that serve as a unique identifier of their parent element.

How to Define

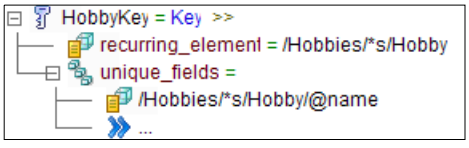
You can define a `Key` only at the global level of the IntelliScript. This allows you to reference the `Key` anywhere in the project.

Example

The Example of Indexing by Key defines a key for the `Hobby` element in the following structure:

```
<Hobbies>
  <Hobby name="Swimming">
    <Person firstName="Eric" lastName="Smith"/>
    <Person firstName="Edward" lastName="Doe"/>
  </Hobby>
  <Hobby name="Biking">
    <Person firstName="Elizabeth" lastName="Doe"/>
  </Hobby>
  <Hobby name="Painting">
    <Person firstName="Mary" lastName="Smith"/>
  </Hobby>
</Hobbies>
```

The key is the `name` attribute, which uniquely identifies each `Hobby`.



Composite Keys

Optionally, you can define a list of data holders as a composite key. To do this, nest multiple data holders under the `unique_fields` property.

Consider the following example:

```
<Persons>
  <Person ID="17" SubID="A">Bob</Person>
  <Person ID="17" SubID="B">Jane</Person>
  <Person ID="35" SubID="A">Larry</Person>
</Persons>
```

Neither the `ID` attribute nor the `SubID` attribute identifies a `Person` element uniquely. The combination of `ID` and `SubID`, however, is a unique identifier. You can define `ID` and `SubID` as a composite key.

Restrictions on the Key

The `unique_fields` must be nested within the `recurring_element`. They can be attributes of the element, they can be nested elements at any level of nesting, or they can be attributes of the nested elements.

For example, this means that `Persons/Person/SocialSecurity/@Number` can be a valid key for `Persons/Person`, because `@Number` is nested within `Persons/Person`. On the other hand, `Persons/Child` is not a valid key for `Persons/Person` because it is not correctly nested.

The `unique_fields` must identify the closest ancestor that can have multiple occurrences. For example, if both `Parent` and `Child` are multiple-occurrence elements, then `Parent/Child/@name` can be a valid key for `Parent/Child` but not for `Parent`.

The `unique_fields` must have simple data types. They cannot be structures.

Sibling and Non-Sibling Occurrences

A key uniquely identifies sibling occurrences of an element. It is permitted for non-sibling occurrences to have the same key.

Consider the following XML structure:

```
<Report>
  <Company>
    <Employee ID="1">John</Employee>
    <Employee ID="2">Leslie</Employee>
  </Company>
  <Company>
    <Employee ID="1">Marie</Employee>
    <Employee ID="2">Larry</Employee>
  </Company>
</Report>
```

The `ID` attribute can be a valid key for `Employee` because it uniquely identifies an `Employee` within a single `Company`. The duplication of `ID` values in different `Company` elements does not invalidate the key.

Keys of Reusable Elements

You can define a key on a reusable element that is defined in the XSD schema.

For example, suppose that `Persons/Person` can occur in several different contexts within the XML. If you define ID as a key for `Persons/Person`, the key is valid in any context where `Persons/Person` is used.

Enforced Uniqueness of a Key

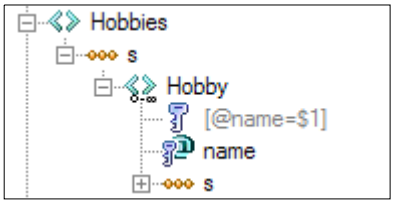
Conversion Agent enforces the uniqueness of a `Key`. This has the following consequences:

- If two or more sibling occurrences of an input element have the same key values, Conversion Agent considers each occurrence to overwrite the previous occurrences. It uses only the last occurrence that it encounters.
- If an occurrence of an input element is missing a key value, the occurrence is ignored.
- If Conversion Agent outputs a keyed element, and a sibling element having the same key value already exists, the existing occurrence is overwritten.

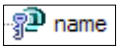
In these cases, Conversion Agent writes a warning in the event log.

Display in the Schema View

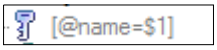
The Schema view displays a key in the following way:



The symbol



means that the `name` attribute has been defined as one of the `unique_fields`. The symbol



is called an XPath predicate. It is an XPath representation of the complete key definition.

Table 13-1. Basic Properties

Property	Description
`recurring_element`	A multiple-occurrence element whose occurrences are identified by the key.
`unique_fields`	The key.

Table 13-2. Advanced Properties

Property	Description
`disabled`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`remark`	

Locator

This component is used in the `source` and `target` properties to identify a data holder.

You can use it to identify either a single-occurrence or multiple-occurrence data holder. In the latter case, each iteration of the component that uses the `Locator` processes the next occurrence of the data holder.

Table 13-3. Basic Properties

Property	Description
`data_holder`	The data holder that the component identifies. Select from a Schema view.

Table 13-4. Advanced Properties

Property	Description
`disabled`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`optional`	
`remark`	

LocatorByKey

This component is used in the `source` and `target` properties to identify an occurrence of a multiple-occurrence data holder.

Before you use this component, you must define a `Key` at the global level of the IntelliScript. The `Key` specifies the data holders that uniquely identify the occurrence.

In the `LocatorByKey` configuration, you must specify:

- The key that you wish to use.
- The values of the key fields. You can specify the values either statically, by typing a value, or dynamically, by selecting a data holder that contains the value.

Conflicts Between Locators

In case of conflicts, a nested `LocatorByKey` overrides a parent locator.

For example, suppose that the `target` property of a `Group` contains a `LocatorByKey` pointing to the third occurrence of an element. A nested `Group` contains a `LocatorByKey` pointing to the fifth occurrence. The nested `Group` uses the fifth occurrence.

Table 13-5. Basic Properties

Property	Description
`key`	From a Schema view, select the XPath predicate representation of the key. For example, if you have defined `Hobbies/Hobby/@name` as a `Key`, then you can select `Hobbies/Hobby[@name=$1]`.
`params`	Under this property, specify the values of the parameters in the XPath predicate. (`$1`, `$2`, and so forth). Type each value, or click the Browse button and select a data holder that contains the value.

Table 13-6. Advanced Properties

Property	Description
`disabled`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`optional`	
`remark`	

LocatorByOccurrence

This component is used in the `source` property to identify an occurrence of a multiple-occurrence data holder, such as an element that can occur multiple times in an XML document or a variable that can occur multiple times.

The component identifies the occurrence by number. For example, if there are ten occurrences of a data holder, you can use `LocatorByOccurrence` to process the third occurrence. `LocatorByOccurrence` can be used to iterate over the occurrences in a repeating structure such as a `RepeatingGroup` anchor.

You can specify the occurrence number either statically, by entering a number, or dynamically, by selecting a data holder that contains the number.

Conflicts Between Locators

In case of conflicts, a nested `LocatorByOccurrence` overrides a parent locator.

For example, suppose that the `target` property of a `Group` contains a `LocatorByOccurrence` pointing to the third occurrence of an element. A nested `Group` contains a `LocatorByOccurrence` pointing to the fifth occurrence. The nested `Group` uses the fifth occurrence.

Table 13-7. Basic Properties

Property	Description
`recurring_element`	The data holder that the component identifies.
`occurrence_number`	The number of the occurrence. Type a number, or click the Browse button and select a data holder that contains the number.

Table 13-8. Advanced Properties

Property	Description
`disabled`	For more information about these properties, see “Standard Serializer Properties” on page 170.
`optional`	
`remark`	

CHAPTER 14

Streamers

This chapter includes the following topics:

How a Streamer Works

A `Streamer` is a Conversion Agent component that splits a large source document into smaller portions that a transformation can process separately. Streamers are useful in transformations that process very large inputs, such as multi-gigabyte data streams.

A streamer offers the following advantages:

- The transformation parses each source segment as soon as it is available, rather than waiting until the entire source is received.
- The transformation has reduced memory requirements.

For example, suppose that an input stream contains stock market transaction data. The stream is transmitted to a server continuously over the course of the entire trading day. A streamer enables Conversion Agent to process each transaction as soon as it arrives, rather than waiting until the end of the day.

In another example, suppose that you receive a large source file over an FTP connection. By using a streamer, Conversion Agent can start processing the file before it is completely received.

Streamers are runnable components. The `Streamer` component is defined at the top-level of the IntelliScript, and it must be set as the startup component of the transformation. It functions by splitting its input into segments and passing them to other runnable components, which can be parsers, mappers, or serializers.

Segments

A streamer identifies segments of its input. It passes the segments individually to parsers, mappers, or serializers, which transform the segment data.

A streamer assumes that the source is composed of:

- A header segment
- Any number of repeating segments
- A footer segment

For each type of segment, the streamer defines a parser, mapper, or serializer that processes the segment.

The repeating segments can be either simple or complex. A simple segment is a single unit of data. A complex segment has its own nested header, repeating segments, and footer.

Headers and footers are always simple segments.

Simple Segments

A simple segment has an opening marker that identifiers where is starts, and a closing marker that identifies where it ends. Thus, a simple segment has the following structure:

```
Opening marker
Data
Closing marker
```

The streamer passes the segment to the specified transformation component, such as a parser.

It is possible to omit some of the markers from the streamer definition. For example:

- If you omit the opening marker of the source header, the header is assumed to start at the beginning of the source.
- If you omit the closing marker, then the segment ends at the opening marker of the next segment.

Complex Segments

A complex segment has a header and footer. Between the header and footer, it can contain any number of nested simple segments, for example:

```
Header
Simple segment
Simple segment
Simple segment
Footer
```

A complex segment can contain nested complex segments, for example:

```
Header
Complex segment
Complex segment
Complex segment
Footer
```

You can also define a complex segment that is missing the header or footer, for example:

```
Simple segment
Simple segment
Simple segment
```

The nested simple segments must all be of the same type. That is, they must all be identified by the same opening and closing markers.

Example

A data stream contains stock transaction data. The stream has the following structure:

- The header begins with the string `yy-MM-dd/`, which is a date followed by a slash.
- The header contains various data, followed by the string `ENDHEAD/`.
- The repeating segments begin with the string `TRANS HH:mm nnn/`, where `HH:mm` is the time on a 24-hour clock, and `nnn` is a serial number of any length.
- The data stream ends with the string `END/`.

The following is a sample data stream conforming to this specification, where . . . represents arbitrary data that must be parsed:

```
06-12-13/...ENDHEAD/TRANS 09:30 1...TRANS 09:30 2...TRANS 09:31 03...TRANS 09:32
14...END/
```

You can parse this stream by using a streamer having the following schematic structure. Notice that the opening and closing markers are located by searching for a particular pattern or string.

Segment	Type	Opening Marker	Closing Marker
Header	Simple	`[0-9][0-9]-[0-9][0-9]-[0-9][0-9]/`	`ENDHEAD/`
Repeating	Simple	`TRANS [0-9][0-9]:[0-9][0-9] [0-9]+/`	none
Footer	Simple	`END/`	none

Header Concatenation

Optionally, you can configure a streamer to concatenate the header segment with each of the repeating segments. The streamer passes the concatenated result to a parser, mapper, or serializer.

For example, suppose that a streamer passes the repeating segment to a parser. The source has the structure

```
Header
Segment1
Segment2
Segment3
```

where `Segment1`, and so forth, are instances of the repeating segment. If you select the concatenation option, the streamer sends the following data to the parser:

```
HeaderSegment1
HeaderSegment2
HeaderSegment3
```

Output of a Streamer

A streamer generates an independent output document for each of the source segments.

Output in Studio Environment

If you run a streamer in the Conversion Agent Studio environment, it combines the individual output segments into a single output.

For example, suppose that the streamer passes each segment to a parser. The output of each parser is an XML document. The combined output is a sequence of XML documents, for example:

```
<?xml version="1.0" encoding="windows-1252"?>
<header>...</header>

<?xml version="1.0" encoding="windows-1252"?>
<repeating_segment>...</repeating_segment>

<?xml version="1.0" encoding="windows-1252"?>
<repeating_segment>...</repeating_segment>

<?xml version="1.0" encoding="windows-1252"?>
<footer>...</footer>
```

This output is not well-formed XML because it contains multiple document elements.

Wrapping Output in a Root Tag

Optionally, you can wrap the combined output of a streamer in a root tag. This is useful, for example, to convert the output in the Studio environment to well-formed XML. For example, if you specify a root tag called `MyRoot`, the output becomes:

```
<MyRoot>
  <?xml version="1.0" encoding="windows-1252"?>
```

```
  <header>...</header>

  <?xml version="1.0" encoding="windows-1252"?>
  <repeating_segment>...</repeating_segment>

  <?xml version="1.0" encoding="windows-1252"?>
  <repeating_segment>...</repeating_segment>

  <?xml version="1.0" encoding="windows-1252"?>
  <footer>...</footer>
</MyRoot>
```

Using Markers and Variables in Streamers

Within a `Streamer` component, you cannot use the regular `Marker` and `Variable` components that are used in other types of transformations. Instead, you should use the `MarkerStreamer` component to define the opening and closing markers of simple segments. You can use the `StreamerVariable` component to store temporary data that is shared by all segments.

Creating a Streamer

To create a streamer:

1. Analyze the source structure and identify the segment types.
2. Create or open a Conversion Agent Studio project.
3. In the project, configure a parser, mapper, or serializer that can process each type of simple segment.
4. In the same project, configure a `Streamer` component.
5. Within the `Streamer`, nest `ComplexSegment` and `SimpleSegment` components corresponding to the source structure.
6. For each `SimpleSegment`, define the opening marker and closing marker if required. Define the parser, mapper, or serializer that processes the segment.
7. Define the `Streamer` as the startup component of the project.
8. Run the project on a source document.
9. Open the output file, located in the project's `Results` folder, to view the output.

   If the streamer passes the segments to parsers, the output may fail to display because it contains multiple XML document elements. To solve this problem, wrap the output in a root tag. For more information, see "Output of a Streamer" on page 205.
10. Deploy the project as a Conversion Agent service.

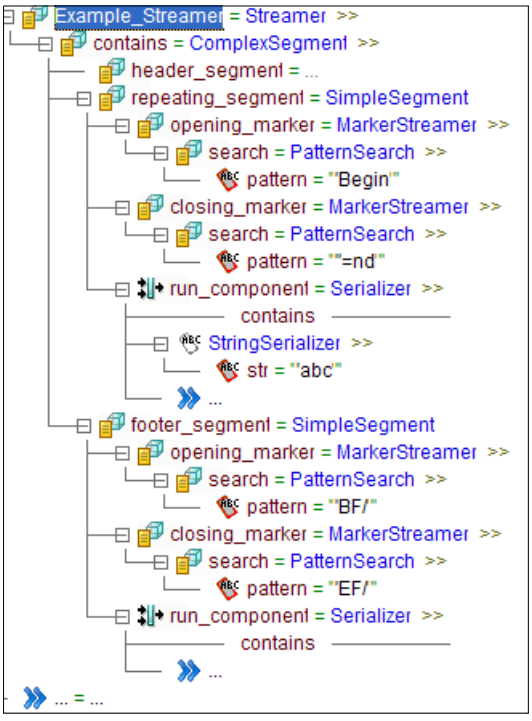
Examples

We present two streamer examples below to illustrate their configurations.

Example 1

The first example contains simple segments. Each segment has a predefined opening and closing marker.

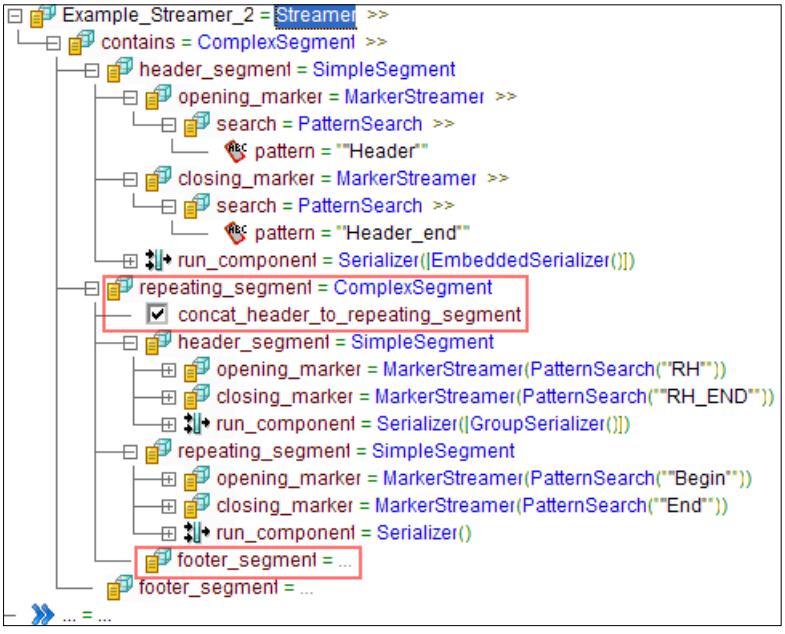
The streamer passes the header and repeating segments to a parser called `body_p`. It passes the footer to a parser called `foot_p`.



Example 2

The following streamer contains a nested, repeating `ComplexSegment`. The nested `ComplexSegment` segment has its own header and nested, repeating `SimpleSegment`. The nested `ComplexSegment` does not have a footer.

Notice that the property `concat_header_to_repeating_segment` has been selected. The effect of this property is to concatenate the header to each instance of the repeating segment. The streamer passes the concatenated segments to the parser `body_p`.



Using a Streamer in an API Application

If you use the Conversion Agent API to run a streamer service, use the special streamer classes to activate the service. For more information, see the API references.

The Conversion Agent COM API does not support streamers.

Streamer Quick Reference

The components used in streamers are:

Component	Description
`ComplexSegment`	Defines a source structure having a header, a repeating portion, and a footer.
`MarkerStreamer`	Defines the start and end of simple segments.
`SimpleSegment`	Defines a source unit having an opening marker and a closing marker. Specifies the transformation that processes the unit.
`Streamer`	Splits a large source into segments for separate processing.
`StreamerVariable`	A user-defined variable whose scope includes all segments of a streamer.

Streamer Component Reference

This section documents the streamer components that are available in Conversion Agent.

ComplexSegment

A `ComplexSegment` defines a source structure having a header, a repeating portion, and a footer.

Table 14-1. Basic Properties

Property	Description
`header_segment`	The header portion of the source. Within this property, you can nest a `SimpleSegment` that defines the header. If you do not assign the property, the source is assumed not to contain a header.
`repeating_segment`	The repeating portion of the source. Within this property, you can nest a `SimpleSegment` that defines the repeating data. You can also nest a `ComplexSegment` that has its own header-repeating-footer structure.
`footer_segment`	The footer portion of the source. Within this property, you can nest a `SimpleSegment` that defines the footer. If you do not assign the property, the source is assumed not to contain a footer.

Table 14-2. Advanced Properties

Property	Description
`concat_header_to_repeating_segment`	If selected, the system concatenates the `header_segment` to each instance of the `repeating_segment`. It passes the result of the concatenation to the `run_component` of the `repeating_segment`. For more information, see “Header Concatenation” on page 205.

MarkerStreamer

A `MarkerStreamer` defines the opening and closing markers of simple segments. It is simple to a regular `Marker` anchor, but it used only in streamers. For more information about how Conversion Agent searches for markers, see “Anchors” on page 67.

Table 14-3. Basic Properties

Property	Description
`search`	The way in which the `MarkerStreamer` finds text. The options are: - `TextSearch`. Searches for an explicit string. - `PatternSearch`. Searches for a regular expression. - `OffsetSearch`. Skips a predefined number of characters following the preceding reference point. - `NewlineSearch`. Searches for a newline character.

Table 14-4. Advanced Properties

Property	Description
`adjacent`	If selected, the `MarkerStreamer` must be adjacent to the end of the preceding segment. This is useful to ensure that the segments are not separated by any other text, including whitespace.
`count`	Specifies from which occurrence of the marker to begin processing. Use `count=3` to skip the first and second occurrences of the marker.
`marking`	Specifies whether the marker should be used as a reference point to identify the succeeding segment or marker. The possible values are: - `full`. Places a reference point before and after the current marker. - `begin position`. Before only. - `end position`. After only.
`name`	A name for the marker, displayed in the event log.
`remark`	A description of the marker.
`disabled`	If selected, the marker is disabled and not used.

Using the Marking Property to Define Segment Boundaries

You can use the marking property to control whether the data in the opening and closing marker is included in the segment and passed to a transformation.

The rule is that the streamer passes the data between the innermost reference points surrounding the segment. For example:

- If the opening marker has `marker = begin position`, the innermost reference point is at the start. The entire marker is included in the segment.
- If the opening marker has `marker = end position` or `full`, the innermost reference point is at the end. The marker is excluded from the segment.

The inverse relationships apply to the closing marker.

To illustrate this, consider a simple segment having the following structure:

```
BEGIN...data...END
```

A `MarkerStreamer` identifies the opening marker by searching for the text `BEGIN`. Another `MarkerStreamer` identifies the closing marker by searching for `END`.

The following table illustrates how the marking property affects the segment boundaries.

Marking of Opening Marker	Marking of Closing Marker	Segment Passed to the Transformation
full	full	`...data...`
full	begin	`...data...`

Marking of Opening Marker	Marking of Closing Marker	Segment Passed to the Transformation
full	end	`...data...END`
begin	full	`BEGIN...data...`
begin	begin	`BEGIN...data...`
begin	end	`BEGIN...data...END`
end	full	`...data...`
end	begin	`...data...`
end	end	`...data...END`

SimpleSegment

A `SimpleSegment` defines a data unit having an opening marker and a closing marker. It defines the parser, mapper, or serializer that should process the component.

The markers are defined by using regular expressions. For more information about the regular expression syntax, see “RegularExpression” on page 121.

Table 14-5. Basic Properties

Property	Description
`opening_marker`	A regular expression identifying the segment start. If omitted, the segment is assumed to start at the beginning of the source or at the end of the preceding segment.
`closing_marker`	A regular expression identifying the segment end. If omitted, the segment is assumed to end at the end of the source or at the start of the next segment.
`run_component`	A mapper or serializer that Conversion Agent should use to process the data in the segment.

Streamer

The `Streamer` component splits its input into segments, and it passes each type of segment to a predefined parser, mapper, or serializer.

The `Streamer` must be defined at the top-level of the IntelliScript, and it must be the startup component of the transformation.

Within a `Streamer`, you must nest a `ComplexSegment`. The `ComplexSegment`, in turn, can contain nested `SimpleSegment` or `ComplexSegment` components.

Table 14-6. Basic Properties

Property	Description
`contains`	A complex segment that defines the overall structure of the source.

Table 14-7. Advanced Properties

Property	Description
`disabled`	If selected, Conversion Agent ignores the component. This is useful for testing and debugging, or for making minor modifications in a project without deleting the existing components.
`name`	A name that you assign to the component. Conversion Agent includes the name in the event log. This can help you find an event that was caused by the particular component.

Table 14-7. Advanced Properties

Property	Description
`remark`	A comment describing the component.
`root_tag`	An XML tag in which the `Streamer` wraps the combined output from all the segments. For more information, see “Output of a Streamer” on page 205.

StreamerVariable

A `StreamerVariable` is a user-defined variable whose scope includes all segments of a `Streamer`. For example, if a streamer contains three parsers, the value of a `StreamerVariable` is available to all three parsers.

For example, a parser that processes a header segment might retrieve data from the header and store it in a `StreamerVariable`. Another parser, which process the repeating segment, can access the value of the `StreamerVariable`. You cannot use a regular `Variable` for this purpose because the value of the variable is not shared between segments.

In other respects, the `StreamerVariable` component is similar to a regular `Variable`. However, a `StreamerVariable` must have a simple, single-occurrence data type. For more information, see “Variables” on page 60.

You can define a `StreamerVariable` only at the top level of the IntelliScript.

Table 14-8. Basic Properties

Property	Description
`val_type`	The XSD data type that the variable can store. Assign a simple type such as `xs:string` or `xs:integer`. Streamer variables do not support complex or multiple-occurrence types.

Table 14-9. Advanced Properties

Property	Description
`initialization`	An initial value for the `StreamerVariable`, assigned when the transformation starts. Select `InitialValue` and enter the value.

CHAPTER 15

Project Properties

This chapter includes the following topics:

Overview

The project properties are options that you can set for the behavior of a project. They control essential features of the project such as the input and output encodings, the authentication support, and the XML validation.

The project properties are saved with the project. They affect the behavior in all circumstances where you run the project:

- In the Conversion Agent Studio environment
- When you deploy the project as a Conversion Agent service and run it in Conversion Agent Engine.

For many projects, you can accept the default values of the project properties. Nevertheless, before you deploy a project as a service, always review the project properties and confirm that the settings meet your needs.

To set the project properties:

1. Select the project in the Conversion Agent Explorer.
2. Click File > Properties.

Alternatively, follow this procedure:

1. Open a TGP script file belonging to the project in an IntelliScript editor.
2. Click Project > Properties.

Properties versus Preferences

Do not confuse the project properties with the Conversion Agent Studio preferences:

- The preferences affect the display in Conversion Agent Studio. They apply to all projects equally.
- The project properties affect the operation of a transformation both in Conversion Agent Studio and in Conversion Agent Engine. You can set the properties independently for each project.

Property Pages

The properties window organizes the properties in several pages. The following sections describe the properties on each page.

Info Properties

The Info page of the project properties displays general information, such as the storage location of the project.

Authentication Properties

If the project accesses a location that requires a login, you can store the login information on the Authentication page of the project properties. This feature is useful, for example, if a parser processes source documents that are located on a password-protected web site.

The options are as follows:

Option	Description
Enable authentication	Select this option if the remote location requires a login.
Prompt before execution	When a login is required, Conversion Agent prompts the user to enter a user name and password.
Save in project	When a login is required, the project automatically submits the user name and password that are specified in the login information.
Login information	The user name and password.

Encoding Properties

The Encoding page of the project properties lets you specify how the input, output, and working files of a project are encoded.

Supported Encodings

Conversion Agent supports the following encodings:

Encoding	Description
Big5	Chinese
Big5-HKSCS	Chinese with Hong Kong Supplementary Character Set
EBCDIC-37	US/Canada
EBCDIC-284	Spanish
EBCDIC-424	Hebrew
EUC-KR	Korean
GB2312	Chinese
GB18030	Chinese
ISO-8859-1	Latin-1 (English and West European)
ISO-8859-2	Latin-2 (East European)
ISO-8859-3	Latin-3 (South European)
ISO-8859-4	Latin-4 (North European)

Encoding	Description
ISO-8859-5	Cyrillic
ISO-8859-6	Arabic
ISO-8859-7	Greek
ISO-8859-8	Hebrew
ISO-8859-9	Latin-5 (Turkish)
ISO-8859-15	Latin-9
KSC_5601	Korean
Shift_JIS	Japanese
TIS-620	Thai
UTF-16	Unicode
UTF-16BE	Unicode
UTF-7	Unicode
UTF-8	Unicode
Windows-874	Thai
Windows-1250	Central European
Windows-1251	Cyrillic
Windows-1252	ANSI English and West European
Windows-1253	Greek
Windows-1254	Turkish
Windows-1255	Hebrew
Windows-1256	Arabic
Windows-1257	Baltic
Windows-1258	Vietnamese

The proprietary Hebrew BaseCodePage continues to be supported in projects that were upgraded from previous Conversion Agent versions. In new projects, use one of the other Hebrew code pages.

Additional encodings may be supported. For an up-to-date list, select one of the Custom options on the Encoding page and open the drop-down list.

Limitations

The encoding support is subject to the following limitations:

- The example pane of the IntelliScript editor may fail to display the Chinese, Japanese, and Korean encodings properly.
- Conversion Agent interprets East Asian and Unicode multiple-byte encodings as binary byte streams. It is not aware of the encoding semantics such as the breaks between characters.
- The UTF-8 encoding is not fully supported as a working encoding. The other Unicode encodings are not supported as working encodings.
- If you define the working encoding as East Asian or UTF-8, avoid defining `Marker` anchors that contain multiple-byte characters. Conversion Agent may misinterpret a `Marker` that contains such characters.

Input Encoding

The Input area of the Encoding page specifies how the source document of a transformation is encoded.

Option	Description
Extract code page from source	If selected, Conversion Agent uses a code page that is specified in the source document, for example, in the `encoding` attribute of an XML document. If Conversion Agent does not find an encoding specification in the document, it uses the encoding defined in the settings described below.
Use working encoding	If selected, Conversion Agent assumes that the input has the same encoding as the working files of the project, as defined in the Working area of the properties page.
Custom	Select the encoding from the list.
Input encoding schema	The encoding of special characters: none or XML. In the XML encoding schema, symbols such as < or > are represented as entities, such as `<` and `>`. Serializers and mappers ignore this option. A serializer or mapper always assumes that its input uses the XML encoding schema.
Byte order	The byte order of binary data. The options are Little Endian, Big Endian, or no binary conversion. The default is Little Endian, which is appropriate for most data on the Windows operating system.

Working Encoding

The Working area of the Encoding page specifies the encoding of the project's working files, including the TGP script files and the IntelliScript.

Option	Description
Use Conversion Agent default codepage	Uses the system default encoding.
Custom	Select the encoding from the list. The ISO-8859 and Windows encodings are supported.
Working encoding schema	The encoding of special characters: none or XML, as for the input encoding.

You must select a working encoding that is compatible with the encoding of your XSD schema. For more information, see “Encoding of the XSD Schema” on page 53.

Output Encoding

The Output area of the Encoding page defines the encoding of the project output.

Option	Description
Use working encoding	Use the same encoding as for the working files.
Same as input	Use the same encoding as for the input.
Custom	Select the encoding from the list.
Encoding schema	The encoding of special characters: none or XML, as for the input encoding. Parsers and mappers ignore this option. A parser or mapper always encodes its output using the XML encoding schema.
Byte order	The byte order of binary data. The options are Little Endian, Big Endian, or no binary conversion. The default is Little Endian, which is appropriate for most data on the Windows operating system.

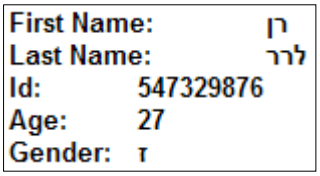
Tips for Configuring the Encoding Properties

Here are some tips for configuring the encoding properties:

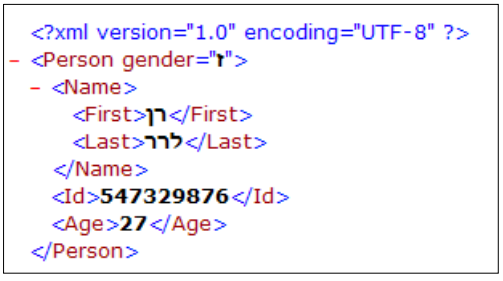
- The input encoding should be the native encoding of the source document.
- The output encoding be the required encoding of the output document.
- The working encoding should be identical to the input or output encoding. Try setting it to the same value as the output encoding. It that does not work, try setting it to the input encoding.
- The working encoding must be able to represent the language of the source document. Otherwise, it may be difficult to define components such as `Marker` anchors.

Encoding Example

Suppose that you want to parse the following source document. The values of First Name, Last Name, and Gender use the Hebrew alphabet. The document has the Windows-1255 (Hebrew) encoding.



The desired output is an XML file containing English tag names and Hebrew data. The output is required in the UTF-8 encoding.



You can configure the parser project with the following encoding:

- Input encoding = Windows-1255 (Hebrew)
- Working encoding = Windows-1255 (Hebrew)
- Output encoding = UTF-8

In the IntelliScript, you can configure anchors by using either English or Hebrew search text.

Namespaces Properties

The Namespaces page of the project properties is used to configure XML namespaces.

You must define the namespaces in the `targetNamespace` attribute of the XSD schemas. In the project properties, you can edit only the namespace alias.

For more information, see “Data Holders” on page 51.

Output Control Properties

The Output Control page of the project properties specifies options for how Conversion Agent should generate the project output.

Option	Description
XSLT stylesheet URL	Browse to an XSLT stylesheet to display XML output of the project. Conversion Agent adds an XSLT processing instruction to the XML output, for example: `<?xml-stylesheet type="text/xsl" href="C:\stylesheets\MyStylesheet.xsl"?>` When you display the XML in Internet Explorer, the browser applies the stylesheet.
Create event log	By default, Conversion Agent Studio generates event logs for the project. You can click the Advanced button and define the events to include in the log. If you deselect the option to create event logs, Conversion Agent Studio does not generate an event log. The Events view displays only minimal information, such as the service initialization and termination. This property has no effect when you run a service in Conversion Agent Engine. For more information about the Engine event logs, see the *Conversion Agent Engine Developer Guide*.
Save parsed documents	Specifies whether Conversion Agent should save a copy of the parsed documents with the event log. The event log uses the copy to display the source of an event.
Add binary encoding prefix to output file	Adds a binary byte-order mark at the start of the output file. Some Unicode applications use the mark to identify the encoding.
Disable automatic output	By default, a parser or serializer writes all output that it generates to the results file. If you select this option, the output is not written unless the parser or serializer runs the `DumpValues` action. This is useful mainly for debugging.
Disable value compressions	Disables XML output optimizations. The optimizations improve performance, especially when processing large documents. Do not select this option unless advised by an SAP representative.

Project References Properties

The Project References page is a standard Eclipse properties page, which is not used by Conversion Agent.

Refactoring History Properties

The Refactoring History page is a standard Eclipse properties page, which is not used by Conversion Agent.

XML Generation Properties

On the XML Generator page of the project properties, you can specify how the project ensures that its XML output is valid.

Property	Explanation

Schema location
No namespace schema location
	These options insert `schemaLocation` and `noNamespaceSchemaLocation` attributes in the document element of the output XML. For example, you might specify the following values:
	`Schema location = http://example.com/NS1`
	`No namespace schema location = http://example.com/NoNS`
	If the document element of the output XML is called `Doc`, Conversion Agent outputs the following code:
	`<Doc`
	`  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`  xsi:schemaLocation="http://example.com/NS1"`
	`  xsi:noNamespaceSchemaLocation="http://example.com/NoNS">`

XML output mode	This option specifies what Conversion Agent should do if a parser does not map any data to an XML element or attribute that is defined in the XSD schema. The options are: - **Full.** Conversion Agent attempts to add the missing data holders to the XML output. It assigns the default value, if the schema defines one. It there is no default, it assigns 0 to integer data holders, 0.0 to floating data holders, or an empty value to other data holders. - **Compact.** Conversion Agent does not add the missing data holders, and it removes empty data holders. A data holder containing the number 0 is not considered empty and is not removed. - **As Is.** The XML output contains the data holders that the parser explicitly set. Conversion Agent does not add missing data holders, and it does not remove empty values. These options do not cause a parser to produce invalid XML, provided that you select the Validate Added option. Under certain conditions, however, the Compact or As Is options can cause a parser to output partial or empty XML. For example, suppose that you choose the Compact mode, and the parser does not create a required element. Conversion Agent removes its parent element in an attempt to create valid XML. If the parent element is also required, the grandparent is removed. This process continues until Conversion Agent reaches an optional element or until the XML is empty.
Add default values for required elements/attributes	These options instruct a parser to output XML elements or attributes that are required by the XSD schema. The options override the Compact and As Is output modes. Conversion Agent assigns values to the required data holders as in the Full output mode.
Validate added required XML elements or attributes	If selected, this option causes Conversion Agent to validate the elements or attributes that it adds because of the Full or Add Required options. If adding the element or attribute would invalidate the output XML, Conversion Agent does not add it. This option is selected by default. Deselecting the option may result in invalid XML.
<?xml?>	The options under this heading add a processing instruction at the beginning of the output XML, for example: `<?xml version="1.0" encoding="Windows-1252"?>` Select the XML version and the value of the `encoding` attribute. For the encoding, you can choose the output encoding or a custom encoding designator. For more information, see “Encoding Properties” on page 214.

Property	Explanation
Add custom processing instructions	This option adds custom processing instructions to the XML header. Type the processing instructions, including the `<??>` symbols.

Add XML root element	This option lets you wrap the output XML in a tag that is not configured in the IntelliScript and possibly is not defined in the XSD schema. For example, if the output of a parser is `<Result>1.0</Result>` and you set the root element to `OutputWrapper`, the project generates the following output:
	`<OutputWrapper>`
	`  <Result>1.0</Result>`
	`</OutputWrapper>`
	You must use this option if you run a parser on multiple source documents. For more information, see “Running on Additional Source Documents” on page 223.

CHAPTER 16

Running and Testing Projects

This chapter includes the following topics:

Overview

When you develop a transformation, test and debug it thoroughly before you put it into production. You can use several tools for testing, debugging, and troubleshooting, such as:

- Color-coding a source document in the example pane of the IntelliScript editor
- Running the transformation in Conversion Agent Studio and viewing the results
- Viewing the event log that Conversion Agent generates for each run
- Cross-identifying an anchor in the example source, in the IntelliScript, and in the Events view

Color-Coding the Example Source

As you construct a parser, Conversion Agent Studio color-codes the anchors that you have defined in the example source.

- In the learn-example style, the specific anchors that you use to define the document structure are color-coded, for example, the anchors in the first iteration of a repeating group.
- In the mark-example style, all the anchors that Conversion Agent finds in the document are color coded, for example, all iterations of a repeating group.

By examining the color-coded text, you can confirm that the parser identifies the anchors correctly.

You can choose the following options from the IntelliScript menu or the toolbar to control the color-coding style:

Option	Description
Learn the Example Automatically	Enables automatic color coding in the learn-example style. When you define anchors in the IntelliScript, the Studio automatically colors the corresponding location in the example.
Learn Example	Color-codes the anchors in the learn-example style. You can use this command to activate the color coding if you have deselected the option to Lean the Example Automatically. You can also use this command to return to the learn-example style, after you have displayed the mark-example style.
Mark Example	Runs the parser and color-codes the anchors in the mark-example style.
Stop Learning or Marking the Example	Stops the color-coding operation. If the example source is very long, you can use this option to halt the color display and speed up the response.

Figure 16-1. Color-Coding in the Learn-Example Style

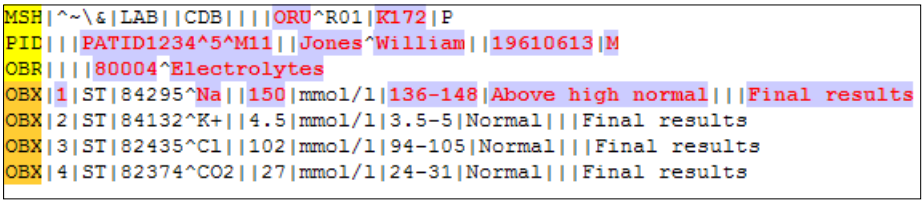
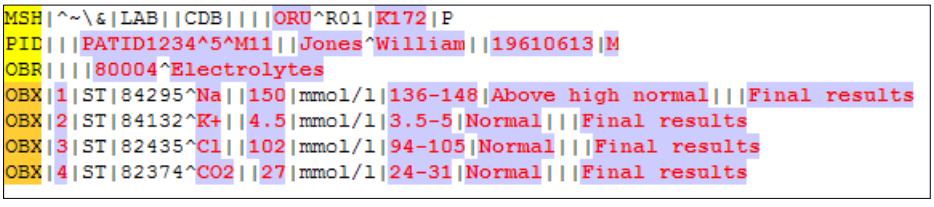


Figure 16-2. Color-Coding in the Mark-Example Style



Displaying Additional Test Documents

You can test a parser by opening additional source documents in the example pane. To do this, click IntelliScript > Test Document and browse to the desired file. Then click IntelliScript > Mark Example and confirm that the parser color-codes the anchors correctly.

Running in Conversion Agent Studio

To test a transformation, you can run it in the Conversion Agent Studio environment.

To run a transformation in the Studio:

1. Set the startup component of the project.

   The startup component is a parser, serializer, mapper, global transformer, or streamer that the project should activate. You can set the startup component in any of the following ways:

   - In an IntelliScript editor, right-click the component and click Set as Startup Component.
   - In the Component view, right-click the component and click Set as Startup Component.
   - Click Run > Run and select the component from a list.

2. Optionally, set the initial values of the variables defined in the project.

- Click Run > Run
- Click the Details button.
- Enter the names and values of variables.

The initial values that you assign in this way are used when you test the project in the Studio. For this purpose, they override the `initialization` property of the variables. They have no effect when you later deploy the project as a service.

For more information, see "Initializing Variables at Runtime" on page 63.

3. Run the transformation in one of the following ways:
   - Click Run > Run, then click the Run button
   - Click Run > Run *`StartupComponentName`*

   The Studio displays the Events view, which informs you of any problems that occurred in the execution. For more information, see "Viewing the Event Log" on page 224.
4. View the results by double-clicking the output file, in the `Results` folder of the Conversion Agent Explorer.

If the Output File is not Displayed

Occasionally, a serious error may cause Conversion Agent to generate an output file that cannot be viewed in the default viewing application. To diagnose the problem:

- Examine the Events view for a description of the problem. For more information, see "Viewing the Event Log" on page 224.
- Try opening the output file in an external application such as Notepad.
- If the output file is not created at all, examine the Output Control page of the project properties, and confirm that the option to Disable Automatic Output is not selected. For more information, see "Output Control Properties" on page 218.

Running on Additional Source Documents

By default, Conversion Agent Studio runs a transformation on the example source. Test the transformation on other source documents, as well.

Parsers

To test a parser on additional source documents, assign the `sources_to_extract` property. The value is a single file or a set of files, optionally containing wildcards. For more information, see "Parser" on page 13.

If you select multiple sources, you must select the option to add an XML root element. If you do not do this, the XML that the parser generates is not well formed because it does not have a unique root. For more information, see "Project Properties" on page 213.

Serializers and Mappers

To test a serializer or a mapper, set the `example_source` property to the desired source. If you leave the `example_source` blank,. the Studio prompts for the input file.

Transformers and Streamers

When you run a global transformer or a streamer, the Studio prompts for the input file.

Viewing the Event Log

When you run a transformation in the Conversion Agent Studio environment, the Events view displays the events that occur during the execution. Examine the events for failure or warning messages.

By default, the Studio event log is the file `Results/events.cme` in the project folder.

Event-Log Properties

In the project properties, you can configure the events that Conversion Agent writes to the log. For more information, see "Output Control Properties" on page 218.

Event Display Preferences

You can customize the event display by using the Window > Preferences option. On the Conversion Agent page of the preferences, you can configure:

- The types of events that the Studio displays, such as notifications, warnings, or failures.
- Whether the failure events propagate ("bubble up") in the events tree. Propagation lets you find the failure events more easily because they are labeled at the top levels of the tree.

The preferences are independent of the event-log properties. The properties control the events that the system stores in the log. The preferences control how the stored events are displayed.

To configure event preferences:

1. Click Window> Preferences.
2. Select the Conversion Agent Events category.
3. Under the Filters heading, choose the events you want Conversion Agent to display. The choices are:
   - Notifications
   - Warnings
   - Failures
4. Optionally, click Propagate All Events.

   Alternatively, select individual events to be propagated and click Propagate Selected Events. You can select multiple events by pressing Control and clicking on each relevant row.

Figure 16-3. Event Display without Propagation

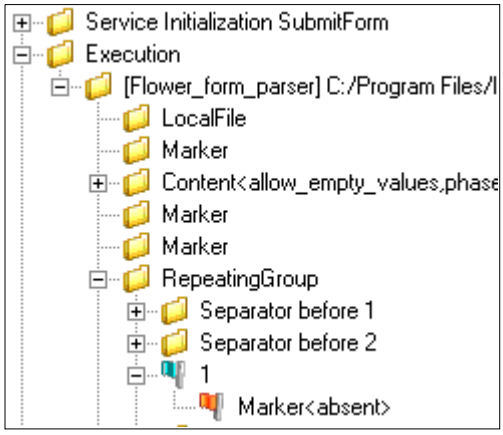
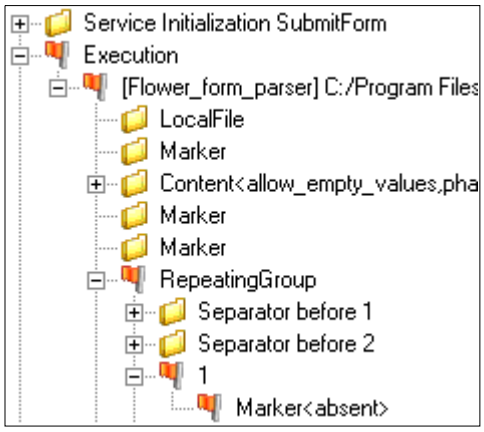


Figure 16-4. Event Display with Propagation



Understanding the Event Log

The event log displays the detailed events that occurred during the execution. For example, it displays an event for each anchor that a parser found.

To display the events at a particular stage, select the stage in the left pane of the Events view.

The events are labeled with status icons, which have the following meanings:

Status Icon	Meaning	Description
	Information	A normal operation performed by Conversion Agent.
	Warning	A warning about a possible error. For example, Conversion Agent generates a warning event if an operation overwrites the existing content of a data holder. The execution continues.
	Failure	A component failed. For example, an anchor fails if Conversion Agent cannot find it in the source document. The execution continues.
	Optional Failure	An optional component, configured with the `optional` property, failed. For example, an optional anchor is missing from the source document. The execution continues.
	Fatal error	A serious error occurred, for example, a parser has an illegal configuration. Conversion Agent halts the execution.
	Unknown	The event status cannot be determined.

Warnings, failures, and optional failures may be perfectly normal under some circumstances. For example, a `RepeatingGroup` anchor may display an optional failure after its last iteration because it does not find more data to parse. If the event log displays warnings or failures, investigate why they occur, and determine whether they are normal or signal a problem.

Using Named Components

We suggest that you assign the `name` property of the components in your transformations. Conversion Agent uses the `name` to label the events. This can make the event log and the IntelliScript easier to understand, and it helps you identify the source of any failures.

Figure 16-5. IntelliScript with Named Marker Anchors

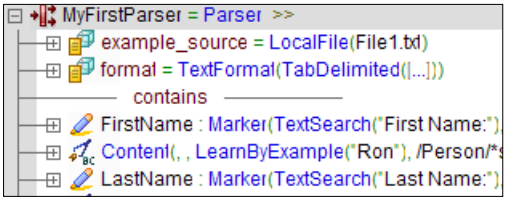


Figure 16-6. Event Log with Named Marker Anchors

Cross-Identifying Events

To help you trace the operation of a transformation and diagnose problems, you can identify the events and the components that caused them in the following ways:

- In the right pane of the Events view, double-click an event corresponding to an anchor, such as a `Marker` or `Content` event. The anchor that caused the event is highlighted in the IntelliScript and in the example source.
- In the example source, right-click an anchor and choose the following options:
  - **View Instance.** Finds the anchor definition in the IntelliScript.
  - **View Event.** Finds the corresponding event in the Events view.
- In the IntelliScript, right-click an anchor and choose View Marking. This finds the corresponding text in the example source.

Opening an Engine Event Log

When you deploy a service to Conversion Agent Engine, you can monitor the Engine event logs for errors or failures. You can open an event log in Conversion Agent Studio by dragging the `*.cme` file to the Events view.

For more information about Engine event logs, see the *Conversion Agent Engine Developer Guide*.

Failure Handling

A failure is an event that prevents a component from processing data in the expected way. An anchor might fail if it searches for text that does not exist in the source document. A transformer or action might fail if its input is empty or has an inappropriate data type.

A failure can be a perfectly normal occurrence. For example, a source document might contain an optional date. A parser contains a `Content` anchor that processes the date, if it exists. If the date does not exist in a particular source document, the `Content` anchor fails.

By configuring the transformation appropriately, you can control the result of a failure. In the above example, you might configure the parser to ignore the missing data and continue processing.

The event log displays warnings about failures. In addition, you can configure a transformation to write a failure message in a user log.

Using the Optional Property to Handle Failures

Failure Causes Parent to Fail

If the `optional` property of a component is not selected, a failure of the component causes its parent to fail. If the parent is also non-optional, its own parent fails, and so forth.

For example, suppose that a `Parser` contains a `Group`, and the `Group` contains a `Marker`. All the components are non-optional. If the `Marker` does not exist in the source document, the `Marker` fails. This causes the `Group` to fail, which in turn causes the `Parser` to fail.

Pictorially, we can represent these relationships in the following way:

```
Parser      //Failed
  Group       //Failed
    Marker      //Failed
```

Optional Failure Does Not Cause Parent to Fail

If the `optional` property of a component is selected, a failure of the component does not bubble up to the parent.

In the above example, suppose that the `Group` is optional. The failed `Marker` causes the `Group` to fail, but the `Parser` does not fail.

```
Parser      //Succeeded
  Group       //Failed
    Marker      //Failed
```

Rollback

If a component fails, its effects are rolled back.

For example, suppose that a `Group` contains three non-optional `Content` anchors, which store values in data holders. If the third `Content` anchor fails, the `Group` fails. Conversion Agent rolls back the effects of the first two `Content` anchors. The data that the first two `Content` anchors already stored in data holders is removed.

The rollback applies only to the main effects of a transformation, such as a parser storing values in data holders or a serializer writing to its output. The rollback does not apply to side effects. In the above example, if the `Group` contains an `ODBCAction` that performs an `INSERT` query on database, the record that the action added to the database is not deleted.

```
Group       //Failed
  Content     //Data holder is rolled back
  Content     //Data holder is rolled back
  ODBCAction  //INSERT query is not rolled back
  Content     //Failed
```

Setting the Optional Property

You can set the `optional` property of a component in two ways:

- Edit the advanced properties of a component in the IntelliScript
- Right-click the component and click Make Optional or Make Mandatory

Components that Lack an Optional Property

Certain components lack the `optional` property because the components never fail, regardless of their input.

An example is the `Sort` action. If the `Sort` action finds no data to sort, it simply does nothing. It does not report a failure.

Writing a Failure Message to the User Log

You can configure a component to output failure events to a user-defined log. For example, if an anchor fails to find text in the source document, it can write a message in the user log. This can occur even if the anchor is defined as optional, so that the failure does not terminate the transformation processing.

The user log can contain information such as:

- Failure level: Information, Warning, or Error
- Name of the component that failed
- Failure description
- Location of the failed component in the IntelliScript
- Additional information about the transformation status, such as the values of data holders.

Configuring User-Log Output

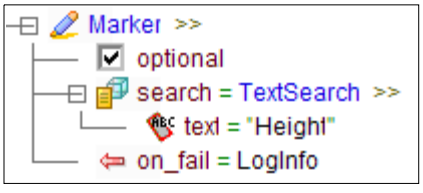
To define the user-log output, assign the `on_fail` property of the appropriate transformation components. the following components have an `on_fail` property:

- Parsers and anchors
- Serializers and serialization anchors
- Mappers and mapper anchors

The property can have the following values:

- **`LogError.`** Writes an error message containing the `VarLastFailure` system variable to the user log.
- **`LogWarning.`** Same as `LogError`, but displays the message as a warning rather than an error.
- **`LogInfo.`** Same as `LogError`, but displays the message as information rather than an error.
- **`CustomLog.`** Runs a serializer that writes a custom message to the user log or another location. For more information, see “CustomLog” on page 140.

The following an example of a `Marker` anchor with a `LogInfo` configuration.



If the `Marker` does not exist in the source document, the system writes the following entry in the user log:

```
***  INFO   *** : Marker, [MyParser[11].Marker], Can't find Marker<optional>('Height').
```

Viewing the User Log

The user log is a text file that you can view in Notepad. On Windows platforms, the default location of the user log is:

```
c:\Documents and Settings\<USER>\Application Data\SAP\ConversionAgent\UserLogs
```

On UNIX platforms, the default location is:

```
<INSTALL_DIR>/UserLogs
```

By default, each execution of a transformation generates a user log having a unique name:

```
<service_name>+<unique_string>.log
```

A transformation can set the user-log location at runtime by using `SetValue` actions to assign the following system variables. Set the phase property of `SetValue` to `initial`, ensuring that `SetValue` runs before any component that writes to the user log.

Variable	Description
`VarServiceInfo/StandardError/StandardErrorDir`	Directory path of the user log
`VarServiceInfo/StandardError/StandardErrorName`	File name of the user log

In the following example, a `SetValue` action sets the user-log directory to `c:\mydirectory`.

Disabling a Component

As you develop and test a transformation, you may wish to disable certain of its components temporarily. For example, if a particular anchor fails, you can disable the anchor and test the transformation without it.

You can enable or disable a component by setting its `disabled` property, in either of the following ways:

- Edit the advanced properties of a component in the IntelliScript.
- Right-click the component and click Enable or Disable.

CHAPTER 17

Deploying Conversion Agent Services

This chapter includes the following topics:

Overview

When you finish configuring and testing a transformation, you can deploy it as a Conversion Agent service. This lets Conversion Agent Engine access and run the project.

You can deploy a service both in the development environment where you use Conversion Agent Studio and on production servers. Deploying in the development environment allows you to develop and test applications that activate the service. Deploying in the production environment allows your applications to run the services on live data.

Note: There is no relation between Conversion Agent services and Windows services. You cannot view or administer Conversion Agent services in the Windows Control Panel.

Preparing a Project for Deployment

Before you deploy a service, test the transformation in Conversion Agent Studio. Then, review points such as:

- Setting the startup component
- Removing any testing or debugging settings that might be inappropriate in a deployed service

Setting the Startup Component

Before you deploy the service, you must set one of its top-level components as the startup component. This is the component that Conversion Agent starts when it runs the service.

The startup component can be a parser, serializer, mapper, global transformer, or streamer. Collectively, these are called runnable components because an application can run them.

Multiple Services with Different Startup Components

If the project contains multiple runnable components, you can deploy it multiple times under different service names. Before you deploy each service, you can select a different startup component.

In this way, for example, you can define multiple parsers in the same project and deploy services that run the parsers.

Reviewing Development Configurations

For testing and debugging purposes, you may have introduced various configurations or project settings that are unneeded or inappropriate in a deployed service. Review your project for such settings. For example:

- You may have inserted components such as `WriteValue` for debugging purposes. You can remove the debugging components.
- You may have used the `sources_to_extract` property of a parser to test multiple source files quickly. You can delete the property value.
- On the XML Generation page of the project properties, you may have selected the option to Add an XML Root Element to support multiple output documents from a single run. You can deselect the option.
- If you configured event logging on the Output Control tab, review the settings.
- You may have configured the `initialization` property of variables. If you plan to pass the initial values as service parameters from an application, you can delete the `initialization` properties, or you can leave them as defaults. For more information, see “Initializing Variables at Runtime” on page 63.

Conversion Agent Repository

Conversion Agent services are stored in the Conversion Agent repository directory. Conversion Agent Engine accesses the services that exist in the repository.

On Windows platforms, the default repository location is:

```
c:\Program Files\SAP\ConversionAgent\ServiceDB
```

On UNIX platforms, the default location is:

```
/opt/SAP/ConversionAgent/ServiceDB
```

To verify or change the repository location:

1. On the computer where you plan to deploy the service, open the Conversion Agent Configuration Editor.
2. View or edit the following setting:

   `CM Configuration/CM Repository/File system/Base Path`

Deploying a Service in a Development Environment

You can deploy a project as a Conversion Agent service on the development computer where Conversion Agent Studio is installed. This allows you to develop, test, and run applications that activate the service.

To do this, you must have write privileges for the Conversion Agent repository and for the `log` folder. For more information about the required permissions, see the *Conversion Agent Administrator Guide*. In case of doubt, contact your system administrator.

To deploy a service:

1. In Conversion Agent Studio, open and select the project.
2. Click Project > Deploy.
3. In the Deploy Service window, set the following options:

Option	Description
Service Name	The name of the service. By default, this is the project name. To ensure cross-platform compatibility, the name must contain only English letters (`A-Z`, `a-z`), numerals (`0-9`), spaces, and the following symbols: `% & + - = @ _ { }` Conversion Agent creates a folder having the service name, in the repository location.
Label	A version identifier. The default value is a time stamp indicating when the service was deployed.
Startup Component	The runnable component that the service should start.
Author	The person who developed the project.
Description	A description of the service.

4. Click the Deploy button.

   The Studio displays a message that the service was successfully deployed. The service appears in the Repository view.

Redeploying

Conversion Agent Studio cannot open a deployed project that is located in the repository. If you need to edit the transformation, work on the original project and redeploy it.

To edit and redeploy a project:

1. Open the development copy of the project in Conversion Agent Studio. Edit and test it as required.
2. Redeploy the service to the same location, under the same service name. You are prompted to overwrite the previously deployed version.

Redeploying overwrites the complete service folder, including any output files or other files that you have stored in it.

Removing a Deployed Service

To remove a Conversion Agent service:

1. In Conversion Agent Studio, display the Repository view.
2. Right-click the service and click Remove.

   This removes only the copy in the repository. It has no effect on the development copy of the project in the Studio workspace.

Deploying a Service to a Production Server

You can deploy a Conversion Agent service from the Conversion Agent Studio computer to a remote computer such as a production server. The remote computer can be a Windows or UNIX-type platform, where Conversion Agent Engine is installed.

To deploy a service to a remote computer:

1. Deploy the service on the development computer. For more information, see "Deploying a Service in a Development Environment" on page 233.
2. Copy the deployed project directory from the Conversion Agent repository on the development computer to the repository on the remote computer. For more information about the repository locations, see "Conversion Agent Repository" on page 232.
3. If you have added any custom components or files to the Conversion Agent `autoInclude\user` directory, you must copy them to the `autoInclude\user` directory on the remote computer.

   For more information about custom components and the `autoInclude` directory, see the *Conversion Agent Engine Developer Guide*.
4. Conversion Agent Engine determines whether any services have been revised by periodically examining, by default every 30 seconds, the timestamp of a file called `update.txt`. This file exists in the repository root directory. The content of the file can be empty.

   If this is the first time that you have deployed a service to the remote repository, `update.txt` might not exist. If so, copy it from the local repository.

   If `update.txt` exists, update its timestamp as follows:

   - On Windows: Open `update.txt` in Notepad and save it.
   - On UNIX: Open a command prompt, change to the repository directory, and enter the following command: `touch update.txt`

Alternatively, if the development computer can access the remote file system, you can change the Conversion Agent repository to the remote location and deploy directly to the remote computer.

Deploying on Multiple Servers

You can deploy services on multiple servers or on a cluster server. For more information, see the *Conversion Agent Administrator Guide*.

Running a Service

After you deploy a service, you are ready to run it in Conversion Agent Engine. You can do this in several ways:

- By using the Conversion Agent Engine command-line interface.
- By programming an application that uses the Conversion Agent API to submit source documents to the Engine. The API is available in several programming languages.
- By posting source documents to the Engine via the CGI interface.

For more information about running a service, see the *Conversion Agent Engine Developer Guide*.

In addition, you can run services by using the Conversion Agent process module for SAP PI. For more information, see *Deploying and Using Conversion Agent*.

INDEX

D

E

F

M

N

O

P

Q

R

S

T

U

V

W

X

NOTICES

This Informatica product (the “Software”) includes certain drivers (the “DataDirect Drivers”) from DataDirect Technologies, an operating company of Progress Software Corporation (“DataDirect”) which are subject to the following terms and conditions:

1. THE DATADIRECT DRIVERS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.

2. IN NO EVENT WILL DATADIRECT OR ITS THIRD PARTY SUPPLIERS BE LIABLE TO THE END-USER CUSTOMER FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR OTHER DAMAGES ARISING OUT OF THE USE OF THE ODBC DRIVERS, WHETHER OR NOT INFORMED OF THE POSSIBILITIES OF DAMAGES IN ADVANCE. THESE LIMITATIONS APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS.