

TIBCO Foresight® Translator

User's Guide

Software Release 3.7
October 2017

Two-Second Advantage®



Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, Two-Second Advantage, TIBCO BusinessConnect, TIBCO BusinessWorks, TIBCO Foresight EDISIM, TIBCO Foresight Instream, TIBCO Foresight Instream Translator 5010 Step Up/Step Down Adaptor, and TIBCO Foresight Translator are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

The United States Postal Service holds the copyright in the USPS City State Zip Codes. (c) United States Postal Service 2017.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Copyright © 2010-2017 TIBCO Software Inc. All rights reserved.

TIBCO Software Inc. Confidential Information

General Contact Information

TIBCO Software Inc.
3303 Hillview Avenue
Palo Alto, CA 94304 USA
Tel: +1 650 846 1000
Fax: +1 650 846 1005

Technical Support

E-mail: support@tibco.com
Web: <https://support.tibco.com>

(Note: Entry to this site requires a username and password. If you do not have one, you can request one. You must have a valid maintenance or support contract to use this site.)

Contents

1	Introduction	1
	Intended Audience	1
	Capabilities.....	1
	File Formats for Translation.....	2
	De-Identification Maps.....	3
	HIPAA 4010-5010 Step-Up and Step-Down Maps	3
	Major Steps	3
	Sample Maps	4
	Important Files	4
	ISLserver	5
2	Guidelines	7
	Guideline Requirements	7
	Guideline Formats.....	7
3	Mapping Basics	9
	Preparing for Mapping	9
	Opening Translation Tool.....	9
	Opening Existing Maps	10
	Saving Maps	10
	Creating a New Map	11
	Navigating	12
4	Mapping Details	15
	Starting a Map.....	15
	Initial Checking of Source and Target	15
	Mapping Elements and Fields	17
	Unmapping	19
	Scrolling after Mapping.....	19
	Mapping Entire Segments	20
	Mapping Loops	21
	Mapping Repeating Items	22
	Mapping Multiple Fields to one Field	23
	One Pass and Two Pass Mapping.....	24
	Guideline Settings.....	26
	Offsets.....	26
	Backing up Maps.....	27
	Colors.....	28
	Translation Tool Properties.....	29

	Tutorials	32
5	Guideline Settings	33
	Viewing Guideline Settings	33
	Source Guideline Settings	33
	Format Tabs	33
	Flat File Source Settings	34
	EDI Source Settings	37
	XML Schema Settings	38
	Allow Delimiter as Part of Data.....	39
	Allow CR/LF as Part of Data.....	39
	Character Replacement.....	40
	Target Guideline Settings	41
	EDI Target Settings	41
	Output Format Settings	42
	Flat File Target Settings	47
	XML Schema Settings	48
	HL7 Target Settings.....	48
	Two Pass Mapping	49
6	Testing a Map	51
	Introduction to Testing	51
	Testing from Translation Tool	51
	Editing In Test Data Input Pane	53
	Saving Translated Data	53
	Validating Translated Data	54
	Comparing Source and Target Data	55
	Debugging and Logging	56
	Testing from a Batch File	57
7	Encoding Options	59
	Encoding and Translation Tool	60
	Encoding and Translator.exe	62
	Encoding and Java API.....	62
8	Running Foresight Translator	63
	Foresight Translator Command Line	63
	Partner Automation	67
	Temporary Files	67
9	Creating, Viewing, and Testing Rules	69
	Source vs. Target Rules	69
	Target Rules	70
	Source Rules	74
	Applying Rules to all like Element IDs	75
	Viewing Rules	75
	Copying Rules.....	77
	Deleting Rules.....	77

	Comparing Rules	79
10	Rules Reference	81
	Reserved Variables.....	81
	Current_Date	82
	Current_Element	83
	Current_Segment	84
	Current_Time.....	85
	FS_ALLSPACE	86
	FS_COMP_SEP	86
	FS_ELM_SEP	87
	FS_ELMID	88
	FS_QUOT.....	88
	FS_REP_SEP	89
	FS_SPACE	90
	FSID.....	91
	FSLOOPID.....	92
	FSLOOPSEQUENCE	93
	FSSEGID	94
	FSSEGSEQUENCE	95
	RECORDCOUNTER	96
	Target_Element.....	97
	Variables	97
	Using Variables	97
	Populating Variables.....	98
	Literal Values	100
	Operands	102
	Rules.2DArray.....	103
	CreateByStringList.....	103
	CreateByStringListAsRow	105
	CreateNewArray	106
	Delete	107
	DeleteAllArrays	108
	DeleteColumnsFromArray	109
	DumpAllArrays.....	110
	DumpArray.....	111
	DumpArrayToCSVFile	112
	GetAllColumnValues	113
	GetAllColumnValuesAsArray.....	114
	GetColumnValue	114
	GetRowCounter	115
	GetSubElementValue	116
	GetValueFromArray.....	118
	HasValue	119
	InsertARow	120
	InsertToArray	121
	InsertToArrayF	123
	Loop.....	125
	MoveToNextRow	125
	OutputMemoryLayout	126

Rules.Counter	127
Increment.....	127
Reset	128
Rules.Database.....	129
Before using Database Rules.....	129
RunQuery	130
RunStoredProcedure	133
Rules.DateTime	135
CompareDate	135
Conversion	137
GetGMTDateTime	138
GetLocalDateTime.....	139
ProcessingDate	140
ToRDX.....	145
ToXmlDateTimeType.....	146
Rules.Encoding	148
Base64decoding.....	148
Base64encoding.....	149
Rules.External.....	150
ISIServerDB.....	150
ISIServerWS.....	150
UserAPIExit	151
Rules.File	152
CreateFileFromArray	152
CreateSearchTable	153
RunTime	154
SwapFile.....	155
WriteStringToFile.....	155
Rules.Format.....	157
COBOL_PIC	157
FloatingDecimal.....	159
LeftJustified	162
MonetaryAmount	163
Packeddecimal	164
Pad	165
RightJustified	166
SetDecimal	167
ToCDATA	168
ToNxType	169
Trim.....	170
TrimLeft.....	171
TrimRight	172
Rules.HL7	173
AddEscape	173
Rules.ICDCodes	174
Rules.JavaAPI.....	175
GetValue.....	175
UpdateValue.....	176
Rules.Mapping	177
Cancel.....	177

CancelWithCondition	179
CodeLookup	181
GetVariableSize.....	181
MappingSegmentWithCondition	183
MappingWithCondition	183
OutputControl	185
SetElementUsage.....	187
SetQualifier	188
WriteTargetSegment	188
Rules.MultipleValues.....	189
MultipleValues	189
Rules.NewRecord	191
Create	191
CreateANewSegment.....	193
CreateANewSegmentWithCondition	194
CreateByTargetOrdinal.....	196
CreateInMemoryByTargetOrdinal	198
CreateLoopBy2DArray	199
CreateMemoryLayout	201
CreateXmlTargetBy2DArray.....	201
DeleteSegmentValues.....	201
InsertRecordBy2DArray.....	202
OutputMemoryLayout	202
PostCreate.....	203
Rules.Rules	205
RunAfterComposite	205
RunAfterLoop	205
RunAfterSegment	206
RunAfterSegmentExit	206
RunAtEndOfFile.....	206
RunBeforeFile.....	207
RunBeforeLoop	207
RunBeforeSegment	207
RunNoData	208
RunOnLoopExit	210
Rules.Search.....	211
Element.....	212
ElementInCurrentSegment	213
Segment	213
SegmentsInLoop	214
Rules.String.....	217
AddXMLEscapeCodes	217
Append	218
Compare.....	219
CompareC	221
CompareX.....	222
ContainC.....	224
ContainS.....	225
HasValue	226
IsEmpty.....	227

Length.....	231
Loop.....	232
RemoveC.....	233
RemoveXMLEscapeCodes	234
Replace.....	235
Substring.....	236
ToLowercase	237
ToUppercase	238
Rules.StringList.....	239
Insert.....	239
Search	240
ListSearchX	241
ToString	242
Rules.TableLookup	243
GetUniqueSequentialNumber	243
SearchInTable	245
Rules.Value.....	251
AddBack	251
AddFront.....	252
Append	253
CheckType	254
Default	256
GetInterchangeCodeValue	257
IsUnPacked	258
ReplaceWith	259
SetValue	261
SumFromArray	262
Rules.Variable.....	263
Add	263
Delete	264
Divide.....	265
DumpAllVariables	266
DumpVariables	267
GetEnvironmentVariable	267
GetVariableSize.....	267
Init	267
IsAllSpace.....	268
IsEmpty.....	269
Multiply.....	270
Set	271
Subtract	272
Rules.XML.....	273
GetAttribute	273
Conditional Rules	274

List of Tutorials.....	281
Important	281
Mapping Tutorial 1 – EDI to XML.....	282
Opening Source and Target	282

	Mapping Data	284
	Looking at Properties.....	285
	Appending Values	286
	Testing your Map.....	287
	The Customer Complex Element	288
	Repeating Loops	289
	Inserting a Literal Value.....	290
	Calculations	291
	Mapping Tutorial 2 – EDI to Fixed Length Flat File	295
	Open Source and Target Guideline.....	295
	Mapping Data	296
	Inserting a Literal Value.....	297
	Saving your Map.....	298
	Testing your Map.....	299
	Mapping two Fields into One.....	300
	The NAME Record	301
	Listing Rules	303
	Mapping from Multiple Segments.....	304
	The CLMS Record.....	306
	The SVC Record	307
	Consulting an External File.....	308
	Conditional Rules	309
	Mapping Tutorial 3 – EDI to EDI	312
	Importing Source and Target Guideline	312
	Applying Rules.....	313
	Specifying Delimiters	314
	Mapping Data	315
	Checking Properties	316
	Checking your Map.....	317
	More Mapping.....	318
12	Appendix B: Java API	321
	Running the Java Demo	321
	Before using the Java API.....	321
	Running the Java Demo.....	321
	Paths and Environment Variables.....	322
	Java Files	323
	Building and Running Java from Translation Tool	323
	Test Java API Dialog.....	325
13	Appendix C: Packed and Unpacked Data	329
	Default Handling of Packed Decimal Source Data	329
	N Source Data, R Target Data	329
	Packed Decimal Chart	331
14	Appendix D: Return Codes	333
	Seeing Return Codes.....	333
	Virus Checking and TIBCO Foresight Products	333
	Foresight Translator Return Codes.....	333

15	Appendix E: XML Schemas	335
	Introduction	335
	Generating an XML Schema with EDISIM Standards Editor	336
16	Appendix F: Foresight Translator Demos	341
	Overview	341
	Demos to run from a Script	342
	Demos to use with Translation Tool.exe	343

1 Introduction

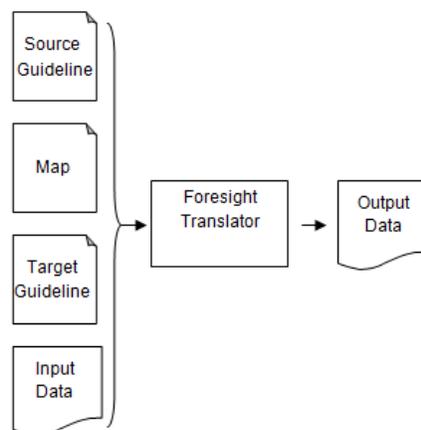
Intended Audience

This document is intended for developers or business analysts who are translating files from one format to another. Familiarity with the source and target formats is assumed.

Please see [Appendix F: Foresight Translator Demos](#) on page 341 for a list of demos that you can run with TIBCO Foresight® Translator.

Capabilities

Foresight® Translator takes in data in one format and creates a new file in another format.



Foresight translation programs in Foresight Translator's \Bin directory include:

- Translation Tool.exe - mapping and testing GUI
- Translator.exe - command line translator

File Formats for Translation

The input EDI can be wrapped or unwrapped.

From (Source)	To (Target)
ACH	XML
EDI	EDI – X12 non-HIPAA (such as VICS and UCS) and X12 HIPAA, version 4010 and later Flat File XML
EDIFACT	EDIFACT Flat File Tradacoms XML
Flat File (fixed and delimited)	Flat File ACH EDI – X12 and HIPAA, version 4010 and later HL7 Tradacoms XML
HL7	HL7 EDI Flat File XML
NCPDP	NCPDP EDI Flat File XML
Tradacoms	EDIFACT Flat File XML
XML	ACH EDI – X12 and HIPAA, version 4010 and later Flat File HL7 2.6 NCPDP Tradacoms

De-Identification Maps

Each HIPAA 5010 document has a map that overwrites sensitive information with dummy values. These maps are automatically installed into Foresight Translator's \Database directory and have names that include the word "Deidentify."

The maps overwrite:

- Names
- Social security numbers
- NPI numbers
- phone numbers
- email addresses
- addresses and zip codes

You can edit these maps to adjust the values that you want in the output.

After de-identification, the data may not pass validation.

HIPAA 4010-5010 Step-Up and Step-Down Maps

TIBCO Foresight has pre-defined maps that will translate HIPAA types 1-2 between 4010A and 5010 Errata. This package is called the TIBCO Foresight® Instream® Translator 5010 Step Up/Step Down Adaptor. Contact your TIBCO Foresight account executive or TIBCO Foresight Support if you would like to discuss purchasing this package.

Major Steps

1. Create guidelines.

Use TIBCO Foresight® EDISIM Standards Editor®. See page 5.

2. Create and test map.

Use Translation Tool.exe. See page 9.

3. Translate.

Use Translator.exe. See page 63.

Sample Maps

See [Appendix F: Translator Demos](#) on page 341.

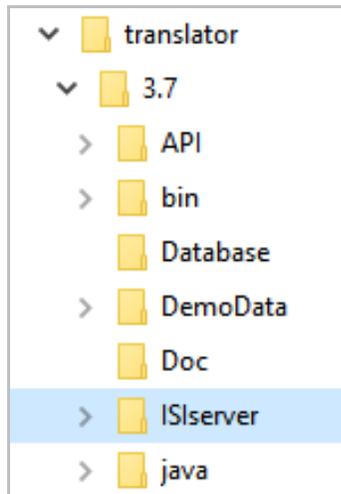
Important Files

File	Directory	Purpose
RuleConfig.xml	Bin	Rule definitions used by Foresight Translator and Translation Tool.
Translation Tool.exe	Bin	Graphical user interface to create and test maps.
Translator.exe	Bin	Command line translator.
Various maps, guidelines and schemas	Database	Maps with filenames DEMO*.map, and guidelines and schemas that they use. Standards and guidelines that are not in the Database directory can be copied from EDISIM@.
Various data files	DemoData	Several data files used with the demos. See Appendix F: Translator Demos on page 341.
TIB_translator_<n.n>_usersguide.pdf	Translator's Documentation archive on docs.tibco.com	Documentation for Foresight Translator.
Scripts	Scripts	Scripts that start with T_ or Translator , plus version-Translator.bat. For details, please see Appendix F: Translator Demos on page 341.

ISIs server

Instream Integration Server (ISIs server) is an optional service that allows Foresight Translator to talk to a web service or to one or more Oracle or SQL databases.

ISIs server is installed as part of Foresight Translator. You will find it in the /ISIs server directory as shown here:



See **ISIs server.pdf** (found in /ISIs server/doc) for more information.

2 Guidelines

Guideline Requirements

Before defining the translation map, Foresight Translator's \Database directory must have a guideline or schema describing the input and output files.

Data Format	Import this format into Translator for Source and Target
EDI	.STD (Saved from EDISIM Standards Editor)
Flat File	.STD (Saved from EDISIM Standards Editor)
XML	.XSD (Exported from EDISIM Standards Editor [recommended] or a schema from another source)

Guideline Formats

You need two guidelines for each translation map:

- Source – describes the data before translation
- Target – describes the data after translation

All involved XSD, STD, and MAP files must be in Foresight Translator's \Database directory.

Please see the tutorials in [Appendix A: Tutorials](#) on page 281.

EDI Guidelines

You can use any X12 or HIPAA guideline starting with version 4010 or later.

This can be:

- A Foresight-distributed guideline.
- One you create with EDISIM Standards Editor. Please see **TIB_fsp_edisim_<n.n>_feditor.pdf** in EDISIM's Documentation directory.

XML Guidelines

You can use a schema (XSD) as the source or target. This can be a schema that you exported from EDISIM Standards Editor (recommended) or a schema from another source. Refer to [Appendix D: Return Codes](#) on page 333 for more information.

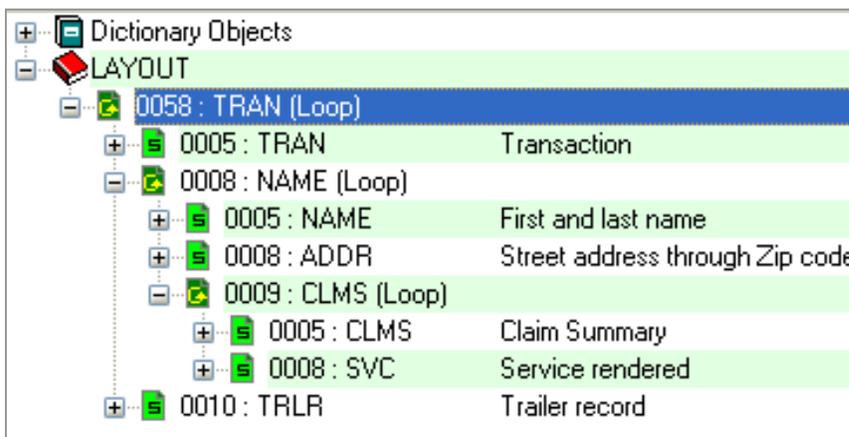
If your schema includes attributes, you will not be able to map them directly, but you can use the GetAttribute rule to retrieve the data if it is on a “record.” Please see [GetAttribute](#).

Flat File Guidelines

Flat file guidelines must be saved in STD format from EDISIM, as described in **FlatFilesAtForesight.pdf**.

In order for a flat file guideline to be used with Foresight Translator:

- The top line must be a repeating loop that encloses the entire data structure.



- There can be no header or trailer record outside of the loop.

3 Mapping Basics

Preparing for Mapping

- Put the source and target guidelines or schemas in Foresight Translator's \Database directory.

For EDI and flat file, these are in STD format.

For XML, this is in XSD format.

- Check field sizes, segment maximum repeats, and loop maximum repeats in the source and target.

You can lose data if a source field is mapped to a smaller target field, or if the source has more loop or segment iterations than the target allows.

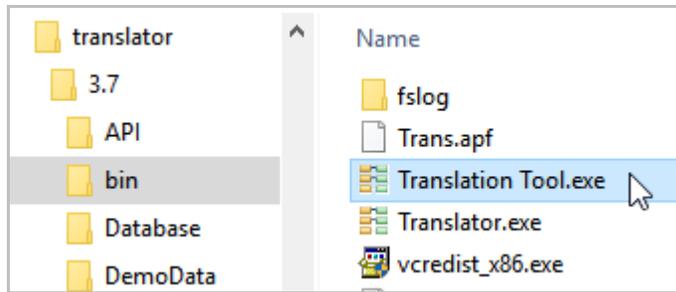
- Have source data for testing.

Opening Translation Tool

Translation Tool is the mapping and testing program.

Ways to start Translation Tool, depending on your particular installation environment:

- **Start | All Programs | Foresight | Translation Tool.**
- **Start | All Programs | TIBCO | <Translator>**
- Execute Translation Tool.exe from Windows Explorer:



Opening Existing Maps

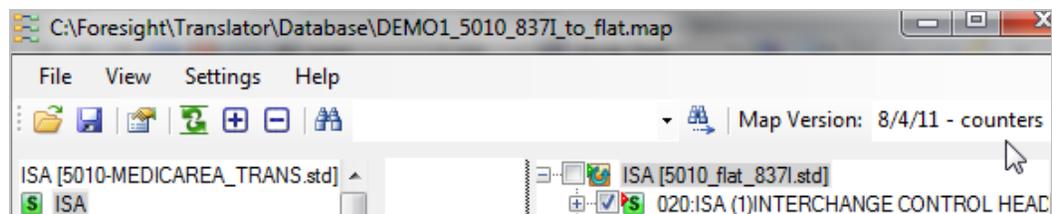
To open an existing map, use any of these from within Translation Tool:

- File | Open Map File ...
- File | Open Backup Map File
- File | Import Backup Map File
- File | Recent map file
- The first toolbar button: 

Maps must have a file type of **.map** or **.xml** and be in Foresight Translator's \Database directory.

Saving Maps

- Save your map to Foresight Translator's \Database directory.
- Use file type map or xml. The preferred type is map, which is less likely to be confused with xml data.
- If you'd like, you can assign a Map Version to the map before saving it. This is a short description or a number of your choice to help you identify a revision of the map:

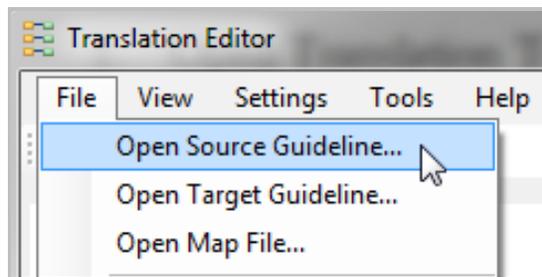


Creating a New Map

1. Open Translation Tool.

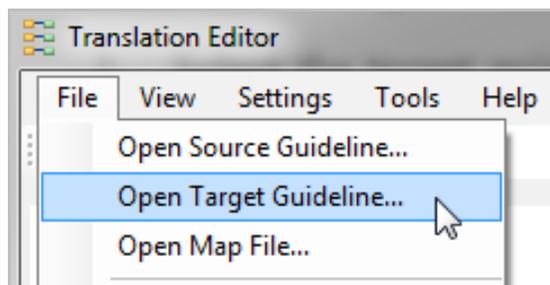
You will see an empty three-pane screen.

2. Select the source guideline with **File | Open Source Guideline...**:



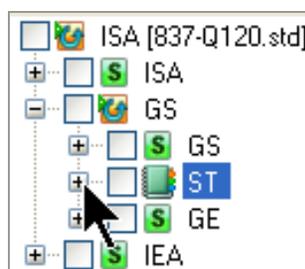
This is the guideline that describes the input data.

3. Select the target guideline with **File | Open Target Guideline...**:



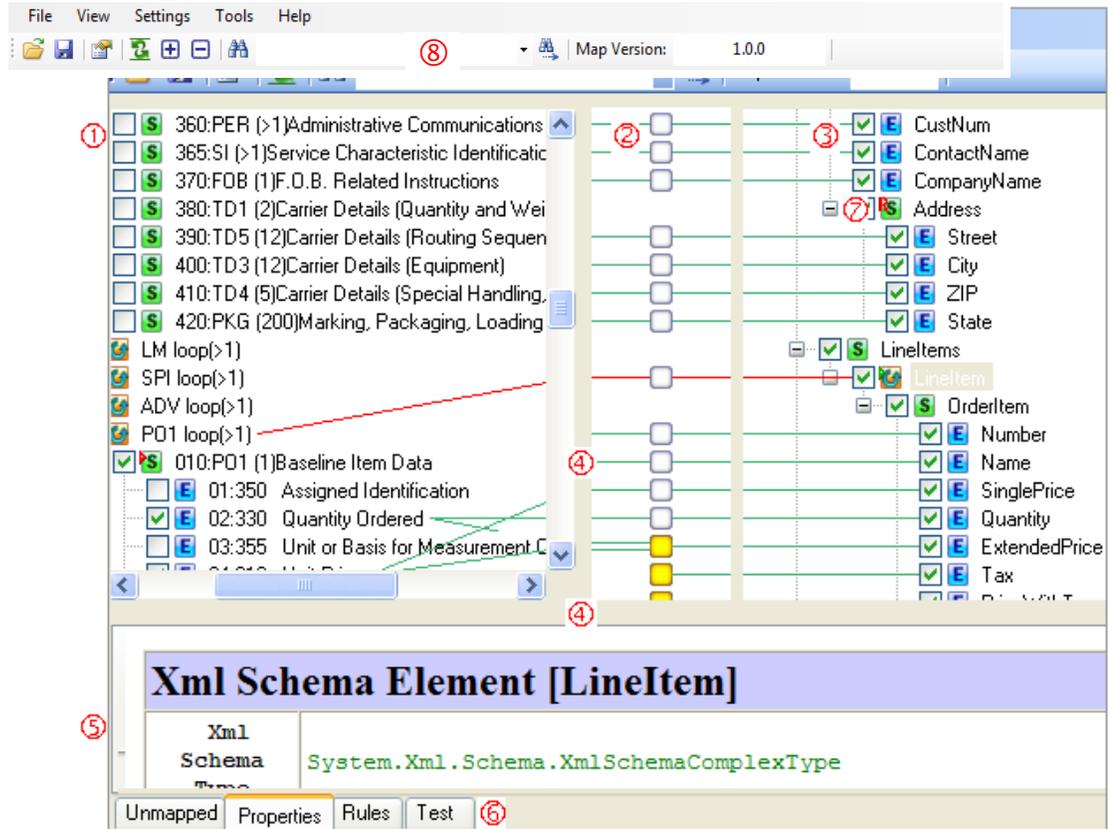
This is the guideline that describes the output data.

4. Open objects by double-clicking on them or by clicking the plus sign in front of them:



With the source and target guidelines both opened, you are ready to map.

Navigating



- = segment or record
- = element or field
- = composite or complex element
- = loop
- = Red R means repeating
- = Red triangle means mandatory

- ① Source pane, a hierarchical list of each data structure in the input files.
- ② Mapping pane, showing map lines and the small box that is the entrance to the rule editor. A yellow rule entrance box indicates a rule on the target or on both target and source. A green box indicates rules only on the target.
- ③ The target pane, a hierarchical list of each data structure in the output files.
- ④ Splitters that you can drag to adjust the size of the panes.
- ⑤ Properties pane.

- ⑥ Properties pane tabs.
- ⑦ Red R means the item repeats.
- ⑧ Search for text in the descriptions by clicking on the source side or target side, typing the text in this field and then clicking  for Find First or  for Find Next.

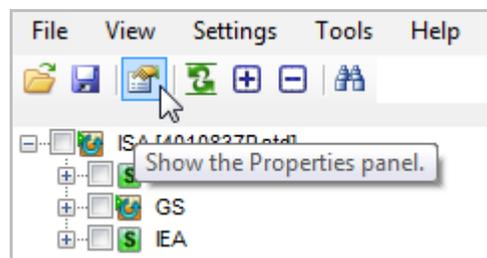
4 Mapping Details

Starting a Map

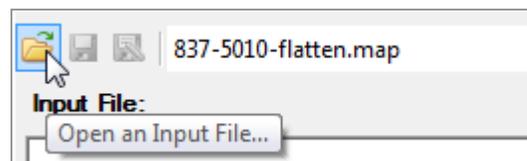
Initial Checking of Source and Target

After selecting the base and target and performing a save, run data through the map to see if Translation Tool can make sense of it.

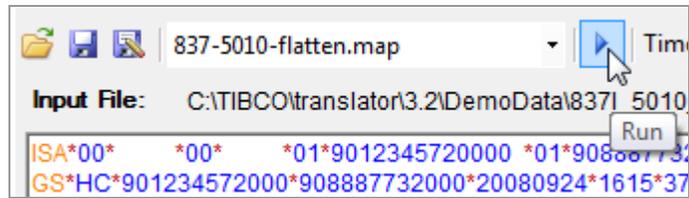
1. Click the **Properties** button on the toolbar to display the **Properties** pane at the bottom:



2. Test some data:
 - a. Click the **Test** tab at the bottom.
 - b. Click the **Open an Input File** button on the bottom toolbar.

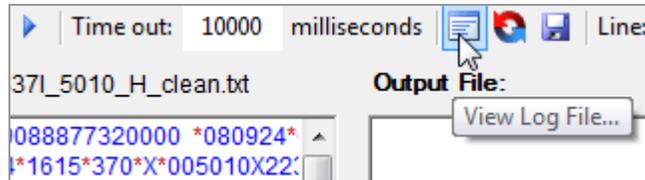


- c. Navigate to a file that matches the source and click **Open**.
- d. Click the **Run** button



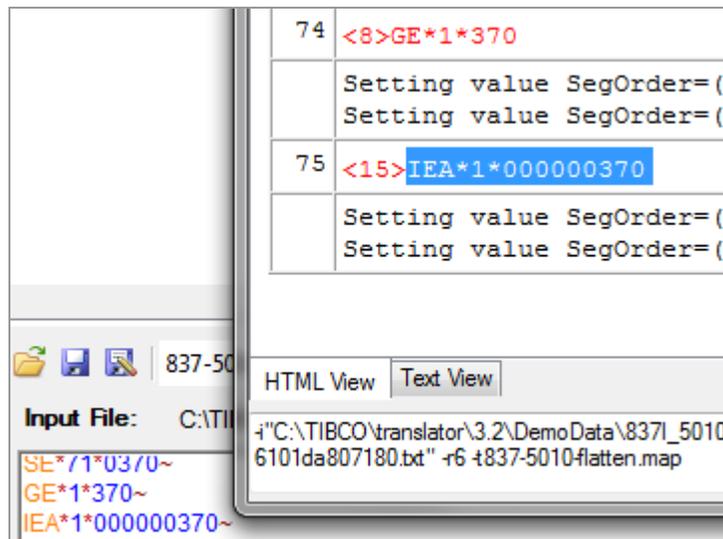
3. Since we haven't mapped, there will be nothing under OutputFile but we need to check the log.

a. Click the **Log** button:



b. Look at the last line in the log.

If it matches the last line in the input file, like this example does, you are ready to start mapping. If not, something is wrong with your source or target guideline.



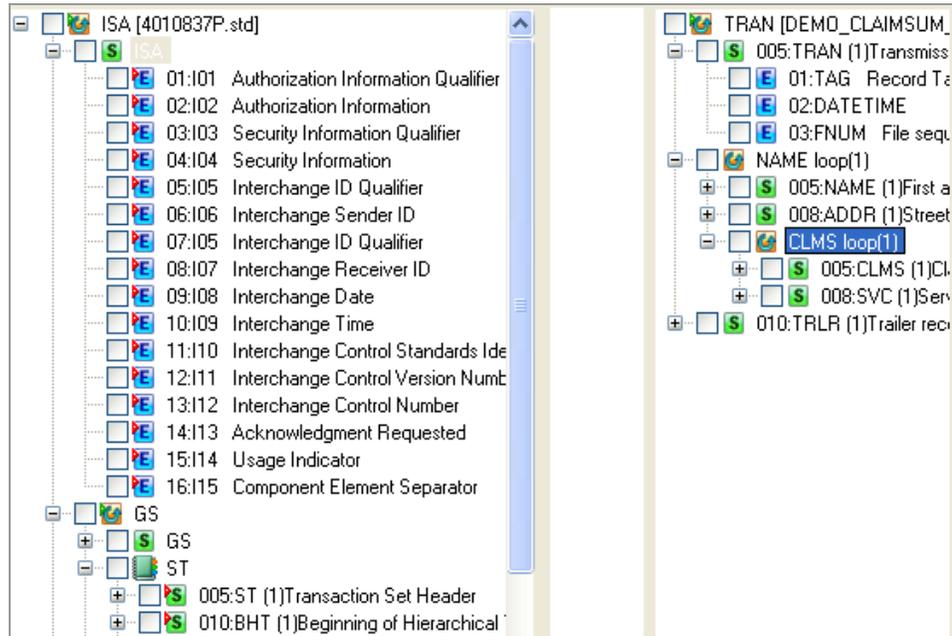
4. Close the log.

Mapping Elements and Fields

After choosing a source and target, or opening an existing map:

1. Open a few segments or records in each pane to display some elements or fields.

The icon for an item that contains data is **E** (for element).



2. Find a data item in the left pane that is to be mapped.
3. Find the corresponding data item in the right pane.

Segments can only be mapped to other segments, and elements can only be mapped to other elements.

Correct

This source TXP segment can be mapped to 280 TXP – another segment.

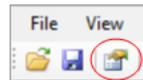


Incorrect

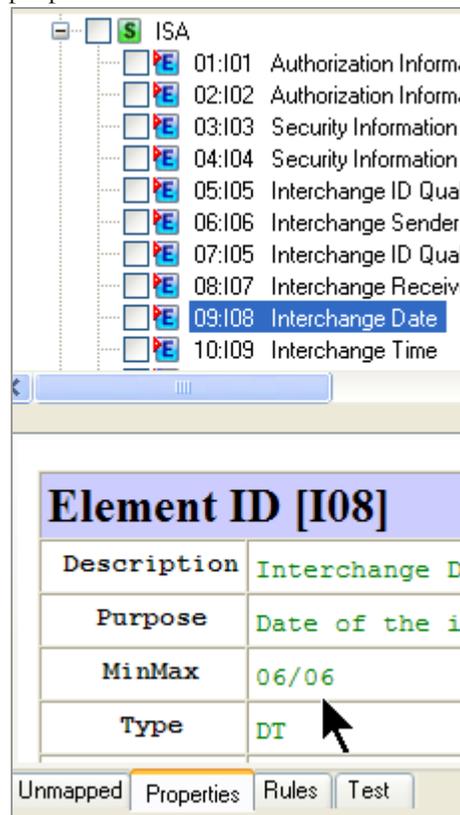
This source element cannot be mapped to 280 TXP.



4. Display the **Properties** pane with this toolbar button:



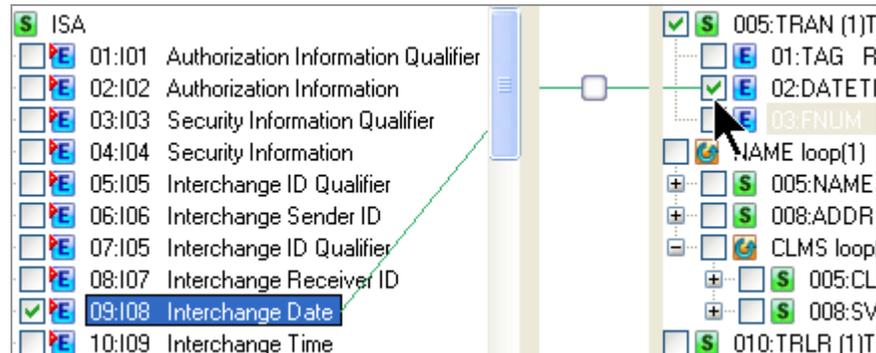
5. Click the element in the source and check its minimum and maximum length on the properties tab.



6. Click the corresponding element in the target and be sure that its maximum is at least as long.
7. In the top pane, use your mouse to drag from the source data item to the target.

When successfully mapped:

- You will see a check mark in the box in front of the source and the target item.
- If you have **View | Show lines** selected, you will also see a red or green line connecting them.



- A Red Line means mapping is present and the mapping point in the **Target** guideline (right-hand side) is selected. That is, you have clicked on the mapped item in the Target guideline.
- A Green Line means mapping is present but the connection is not selected.

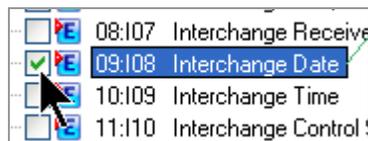
8. Choose **File | Save** and save the map to Translator's **Database** directory.

The green checkmark alerts you that an item is mapped. Although simply clicking on the box will add a checkmark, this does not map anything.

Unmapping

Caution: This deletes all rules on the items.

Click the box in front of the source item to remove the checkmark and unmap it:



Scrolling after Mapping

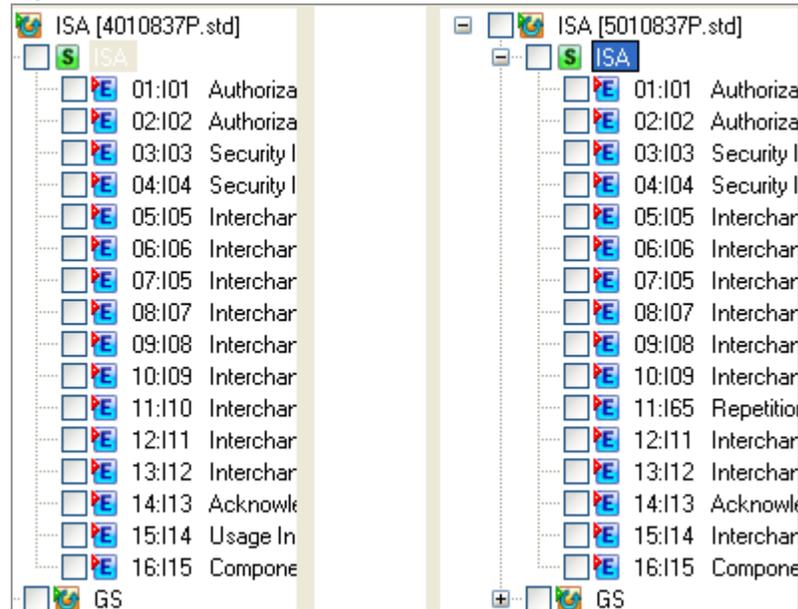
You may have trouble scrolling up or down after mapping. This is because Translator Tool keeps the currently selected connection (the one with the red line) showing on the screen.

To scroll elsewhere, click an unmapped target item, or turn off **View | Show lines**.

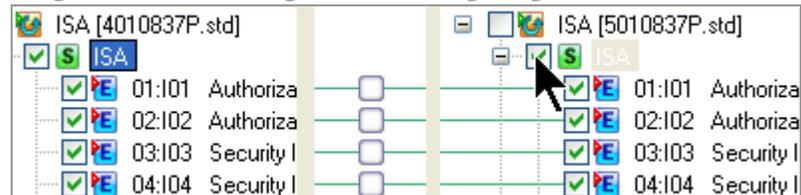
Mapping Entire Segments

To map an entire segment at one time:

1. Open the source and target segment and make sure they have the same number of used elements and that they should all be mapped in order, like these two ISA segments:

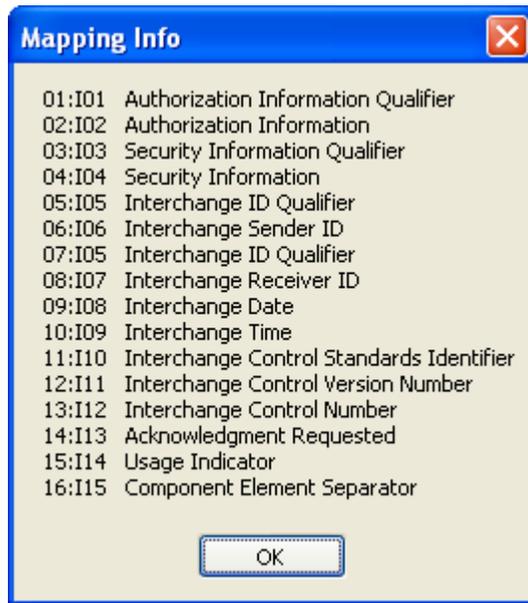


2. Drag from the source segment to the target segment:



Release and notice that all elements in the segment are mapped.

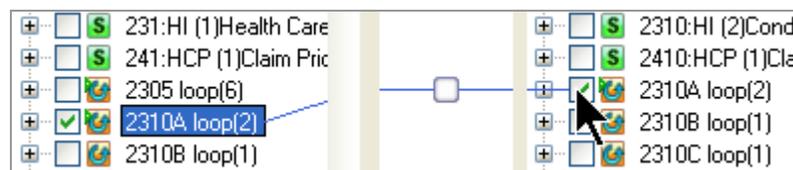
If you have **View | Show Mapping Message** selected, a list of mapped elements pops up:



If the segments do not have the same number of used elements, you cannot map them by dragging between segments. You will need to drag element-by-element.

Mapping Loops

If a source and target loop has the same structure, you can map the entire loop at once by dragging from the source loop header to the target.



If the structure is different, it will map only the segments within the loop that appear to correspond.

Mapping Repeating Items

Pay attention to your looping structure and repeats when you are mapping, so that you don't lose data in the output.

Example

There are different numbers of loops in this source and target 837, but the actual nesting levels are the same (ISA / GS / ST / 2000A / 2000B / 2300). The 2010BC and 2010BD are not factors since the 2300 is not nested within them.

You will need to check the maximum allowed number of repetitions to be sure the source doesn't exceed the target.

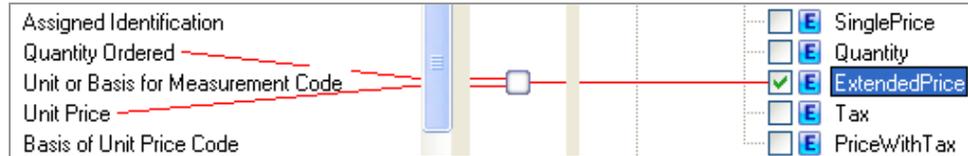
Source	Target	Concerns
2000A = 50 max	2000A = 10 max	The output could be missing data for 40 of the 2000A loops.
2000A = 10 max	2000A = 50 max	No problem. All 10 of the 2000A loops from the source can be accommodated in the output.
2000A = 50 max 2000B = 10 max	2000A = 10 max 2000B = 10 max	The output could be missing data for 40 of the 2000A loops.
2000A = 50 max 2000B = 50 max	2000A = 50 max 2000B = 10 max	The output could be missing data for 40 of the 2000B loops for each 2000A.

The same logic applies to EDI, XML, and flat file data, although the looping structure names may be different.

Mapping Multiple Fields to one Field

To map two or more source fields into one target field:

1. Drag both source fields to the target field:



These can be multiple numeric fields, or one date and one time field.

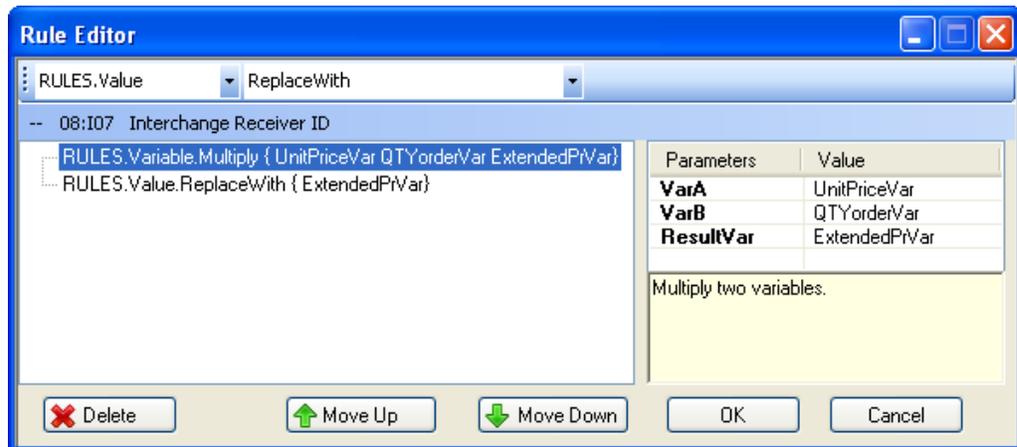
2. Use a [MultipleValues](#) rule (see page 189) to add, subtract, multiply, or divide the numeric values, or to append the time to the date value.

To see a pop-up box listing multiple mapped fields, save and then choose **View | Show Multi-Mapping**.

You will need to turn on 2 Pass Mapping if you map two source fields to one target field. See [One Pass and Two Pass Mapping](#) on page 24.

Possible Alternative without Two Pass

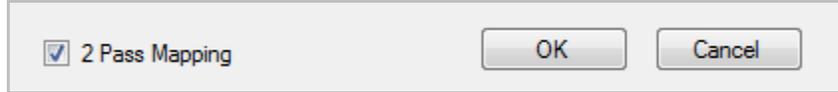
You may be able to accomplish the same result with Variable rules:



See [Rules.Variable](#) on page 263.

One Pass and Two Pass Mapping

Under **Settings | Target Guideline Settings ...** you can specify whether you want Translator to go through your data once or twice during mapping:



Choose 2 pass if:

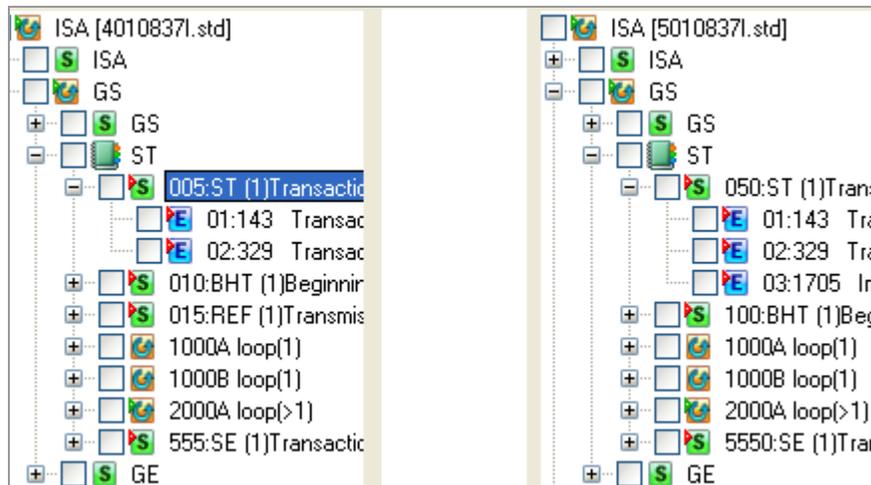
- The records/segments are in a different order in the source and the target.
- Source data items in a segment or record are not all mapped to one target segment or record (unless you use [OutputControl](#). See page 185).
- You use [MultipleValues](#) (see page 189).

1 pass

1 pass saves time and memory, and is useful when the source and target segments/records are in the same order. This is the normal choice for EDI-to-EDI maps. The output will come out in the order in which it is defined in the source.

For example, the 4010 source and the 5010 target in this map have the same structure, with minor variations. The mapping will be straight across: ST-to-ST, BHT-to-BHT, etc. It can be done in one pass.

The output can come out in the same segment order as you see in the source. There may be minor variations, such as some consecutive REFs or DTPs that have been rearranged, but if they are all at the same level, there is no need to perform a second pass to rearrange them. They are acceptable in any order.



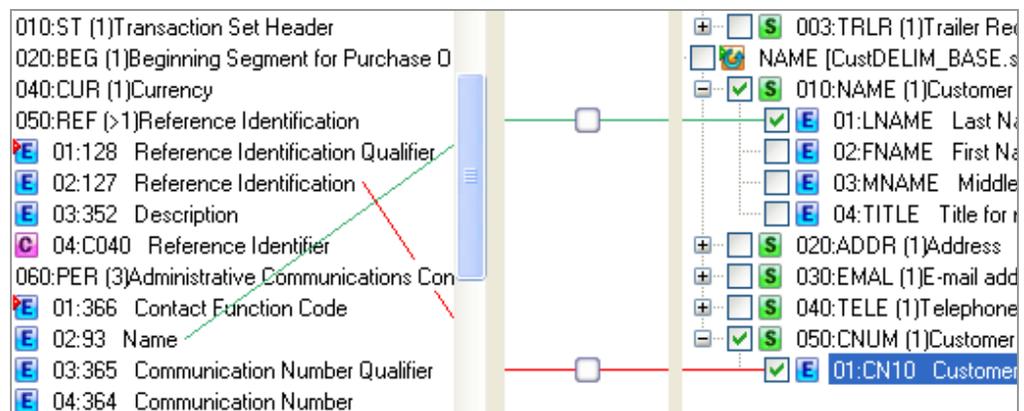
2 pass

The second pass:

- Rearranges the output records/segments to match the order in which they appear in the target guideline.
- Adds data that was missing when the target record was written.
- Processes Multiple Mapping rules.

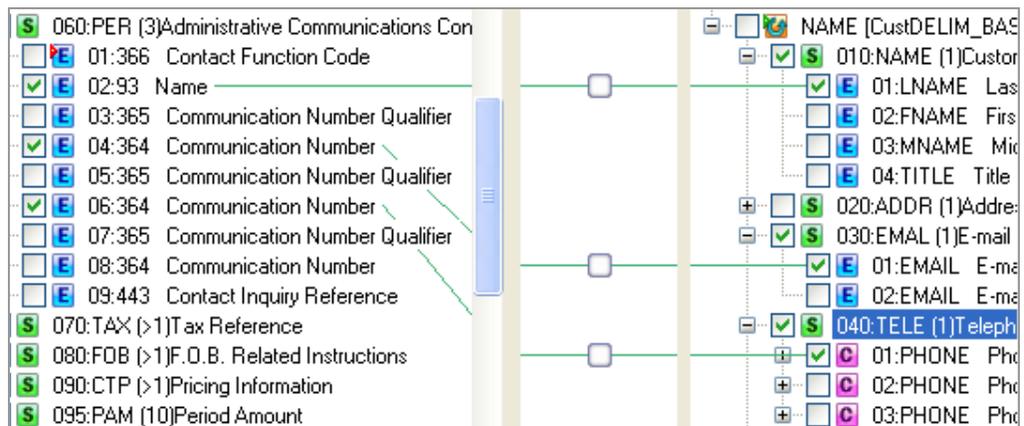
Example 1

This map requires 2 Pass because the target's NAME record will have been passed before the source's PER is reached. During the second pass, the NAME data will be added.



Example 2

This map requires 2 passes because data items in the PER segment are mapped to three different target segments. If they were all mapped to one target segment, then it would not have required 2 passes.



Guideline Settings

To see settings available for the current source or target, choose **Settings | Source Guideline Settings...** or **Settings | Target Guideline Settings...** These are specific to the current map. See [Guideline Settings](#) on page 33 for more detailed information.

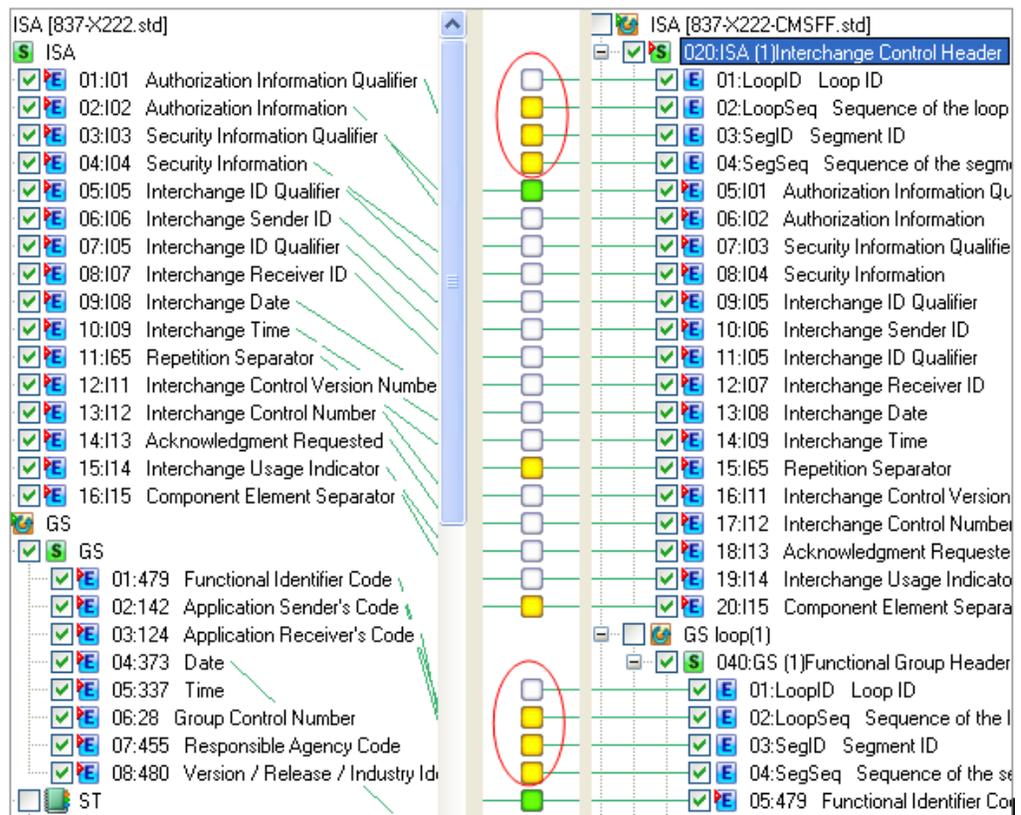
Offsets

This works for all target types.

If you always want to skip the first field(s) in the source or target, you can specify an offset to speed up mapping. This will be used throughout the current map.

Example

In this map from flat file to EDI, the source and target records have exactly the same information in the same order, except each target record starts with four fields that are not mapped.



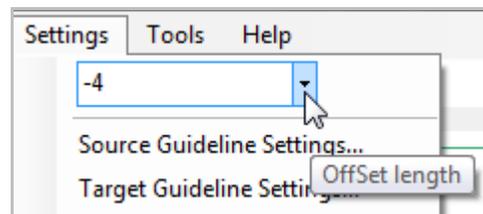
You have two ways to make this work:

1. Drag each mapped field separately.
2. Or, simply drag each segment to its corresponding segment, and specify an “offset” to skip the first n target fields.

Either accomplishes the same result, but the second option is faster.

To specify an offset:

1. Choose **Settings**.
2. Type or choose the number of fields to skip. This is the first option under Settings:



To skip source fields, use a negative number.

To skip target fields, omit the minus sign.

3. Drag each segment/record to its corresponding one in the target. The fields should then correctly map themselves.

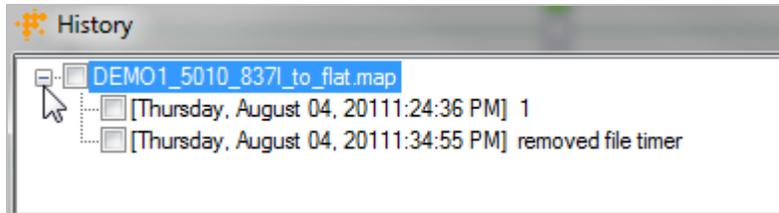
Backing up Maps

File | Save as Backup Map File lets you save a backup to Foresight Translator’s \Mapbackup\files directory, and include an optional comment. (This does not save the map to the Database directory. Use **File | Save** for that.)

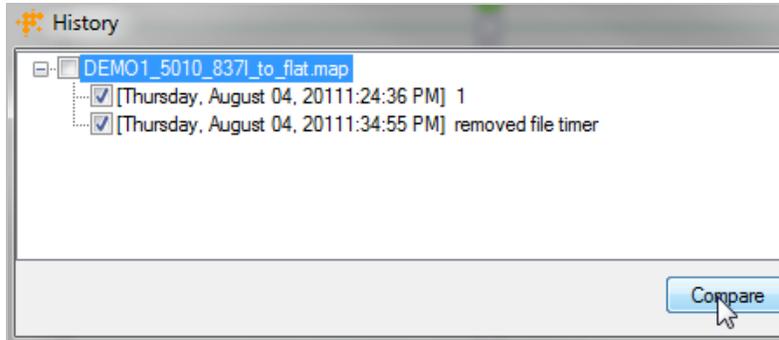
To open a backed-up map later, choose **File | Open Backup Map File**. By clicking the **+** in front of the map, you can see a list of all backups for that map.

(If you have backed up maps with the automatic backup feature that was available in previous versions of Foresight Translator, you can open these maps with **File | Import Backup Map File...** . Automatic backups have been replaced by the method described above, which allows comments and only makes backups when you want them.)

File | History lets you see a list of the backups for the current map



If you have Beyond Compare™ installed, and the path to it is correct under **Settings | Translation Tool Settings ...**, you can click checkboxes in front of two files you want to compare, and click **Compare**.

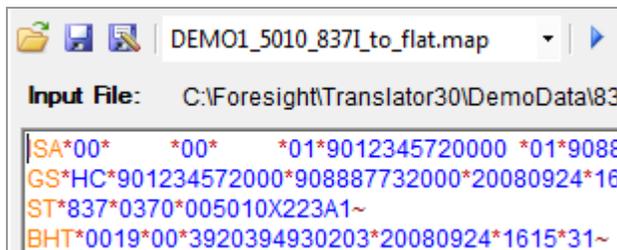


Colors

To change the colors of the EDI in the bottom left test pane:

1. Choose **Settings | Translation Tool Settings ... | EDI Color Coding**.
2. Click **EDI Color Coding** and then click the color patch for the object type that you want to change.
3. Close the box. Your settings are automatically saved.

The next time you open Translation Tool, your color choices will be used:



Color takes memory and slows performance. For very large files, you might want to control how much input text to color, as described in [Input Data Color Text length](#) on page 29.

Translation Tool Properties

To change these settings:

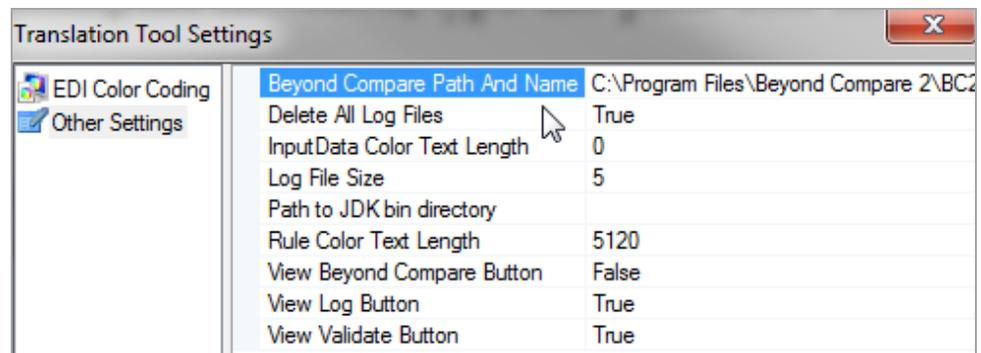
1. Choose **Settings | Translation Tool Settings ... | Other Settings**.
2. Make changes as shown below. When you click on a property, help text appears in the bottom pane.
3. Close the box and your settings are automatically saved.

Changes take effect the next time you enter Translation Tool.

Properties include:

- Beyond Compare Path and Name

If you have Beyond Compare installed, type the path and filename to your Beyond Compare executable:



If you also set **View Beyond Compare Button** to true, you will have a convenient Beyond Compare button on the Test tab's toolbar.

- Delete All Log Files

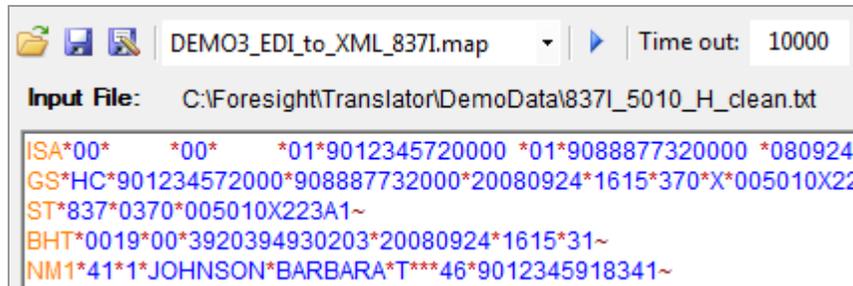
True Delete all log files in the bin/fslog directory when you close Translation Tool.

False Retain the logs until you delete them.

See [Debugging and Logging](#) on page 56.

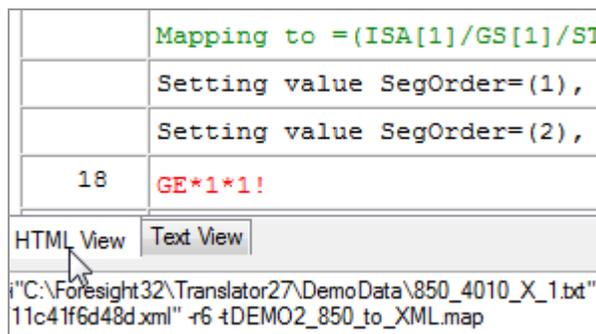
- Input Data Color Text length

This lets you turn off color display for the Input file if it contains more than this number of characters. Reducing the number saves memory and increases speed when opening a very large file.



- Log File Size

The default log file view is HTML, and you can choose Text View instead:



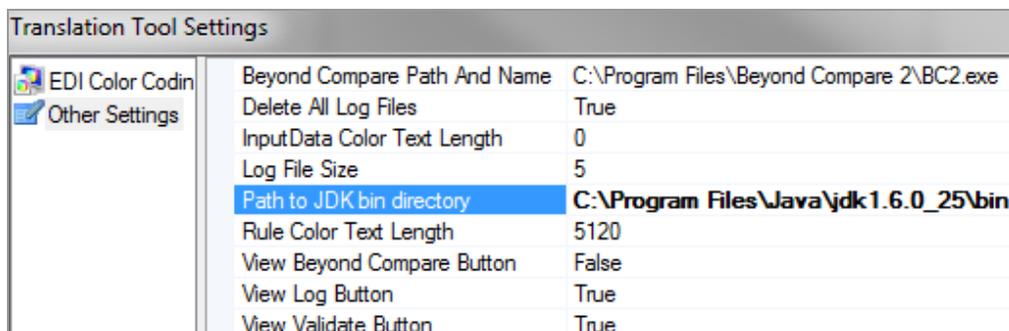
If your log is extremely large, HTML View will take a long time to open. The Log File Size sets a default limit of 5 MB for HTML View. If a log is larger than that, the HTML View tab will not appear and the log will display in Text View.

If you have a powerful machine, you may want to set this limit higher.

- Path to JDK bin directory

If you are using the Java testing feature from Translation Tool, enter the path to the jdk\bin directory here.

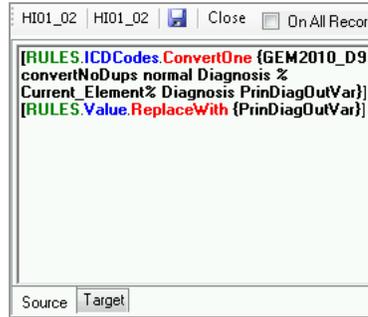
Example:



Please see [Building and Running Java from Translation Tool](#) on page 323.

- Rule Color Text Length

Rules normally display in color in the main rules box:



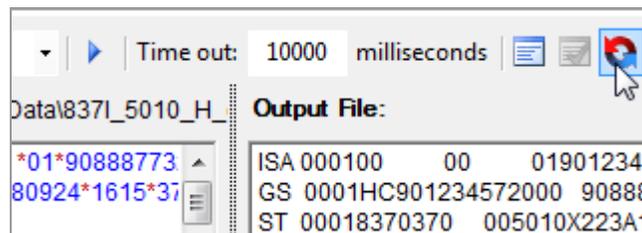
Colored fonts slow down display in this box. It is only noticeable if you have a lot of characters to display.

To speed up the display, set Rule Color Text Length to a lower number of characters. For example, if you set Rule Color Text Length to 1000, and the rules for a certain item are 1001 characters long, all rules on that item will show in a black font in this box.

- View Beyond Compare Button

This is only available if you have Beyond Compare installed and your data is EDI and/or flat file.

True Display a button for Beyond Compare on the Properties Test tab. This lets you compare the source and target data.



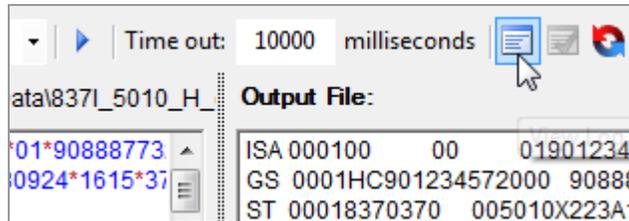
False Do not display a Beyond Compare button.

See [Comparing Source and Target Data](#) on page 55.

Also set **Beyond Compare Path And Name**, which is also under Other Settings.

- View Log Button

True Display a **View Log** button on the Properties Test tab.



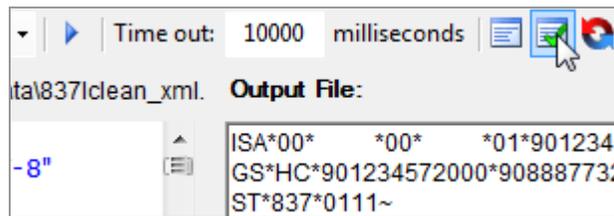
False Do not display a View Log button.

See [Debugging and Logging](#) on page 56.

- View Validate Button

This is only available for EDI target data.

True Display a **Validate** button on the Properties Test tab. This lets you validate the target data against the target standard if it is EDI. To validate, you must have Instream installed.



False Do not display a **Validate** button.

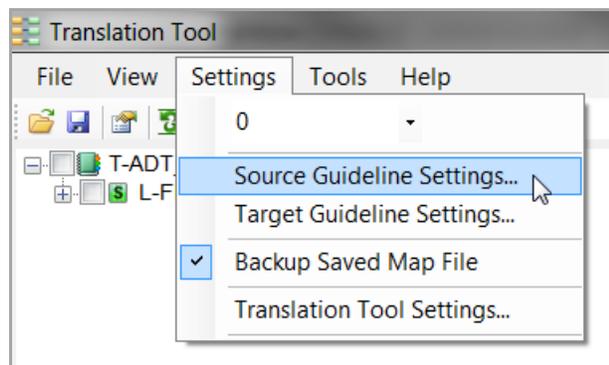
Tutorials

Please see Appendix A: Tutorials on page 281.

5 Guideline Settings

Viewing Guideline Settings

To see settings available for the current source or target, choose **Settings | Source Guideline Settings...** or **Settings | Target Guideline Settings...**

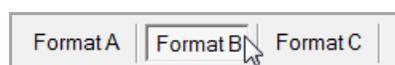


Source Guideline Settings

Guideline settings are specific to the current map. XML, EDI, and flat file types all have different settings; therefore some options may be inactive.

Format Tabs

Depending on the source guideline type, the Format Tabs across the top of the Settings box may be active. Select a tab by clicking on it.



Flat File Source Settings

Flat File Source - Format A

Use this when the source data has fixed length fields and a record key.

Delimited Fixed Flat File with ID	
Key Start Position:	<input type="text" value="3"/>
Key Length:	<input type="text" value="3"/>
Record Terminator:	<input type="text" value="0x0A"/>

Example:
The key starts in position 5.
The key has a length of 4.
The record ends with ^.

```
0001NM1 00041 1JOHNSON^  
0001PER 0001ICARTHURJONES^
```

You need to identify a record key if:

- A flat file guideline is the source
- And its records do not start with a value that identifies the record

Key Start Position

Type the number of characters that precede the key in each record.

Key Length

Type the number of characters in key.

Record Terminator

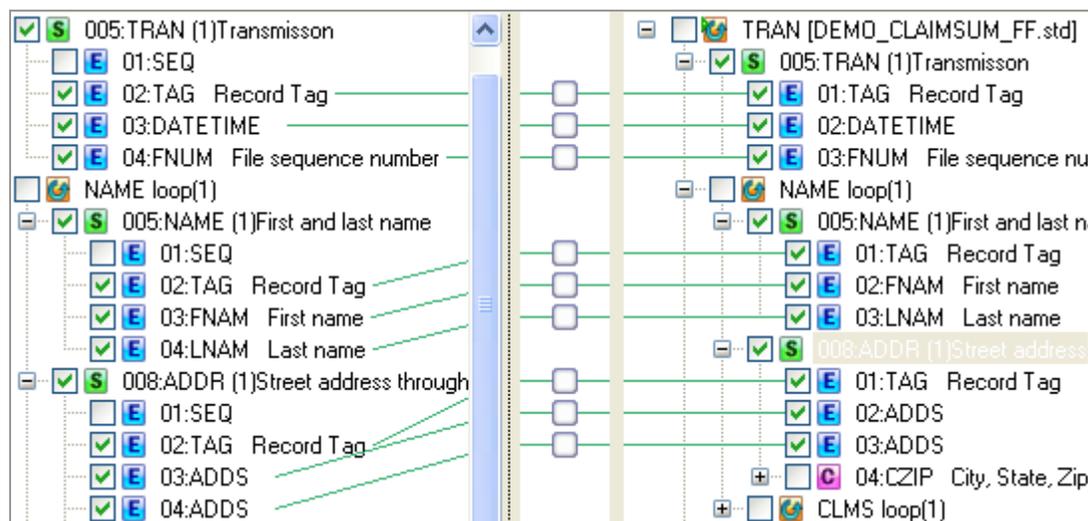
This can be an ASCII character up to 255. Type the keyboard character or the hex value preceded with 0x. The Delimiter is usually 0x0D.

Example

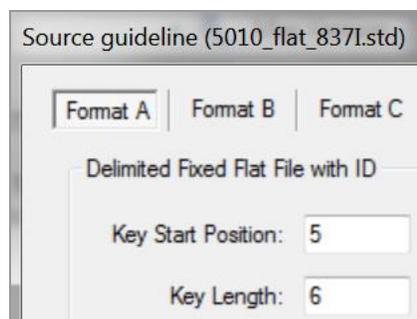
This data starts with a 5-digit sequence number followed by a 6-digit tag that identified the type of record.

```
00001TRAN 0704120909      000004218  
00002NAME MUFFY      JONES  
00003ADDR PO BOX 123      157 WEST 57TH SCINCINNATIOH43017  
00004CLMS 0001      20070412  
00005SVC none      20070412100.00  
00006SVC none      20070412100.00  
00007SVC none      20070412100.00  
00008SVC none      20070412100.00  
00009TRLR 24218
```

The map might look like this:

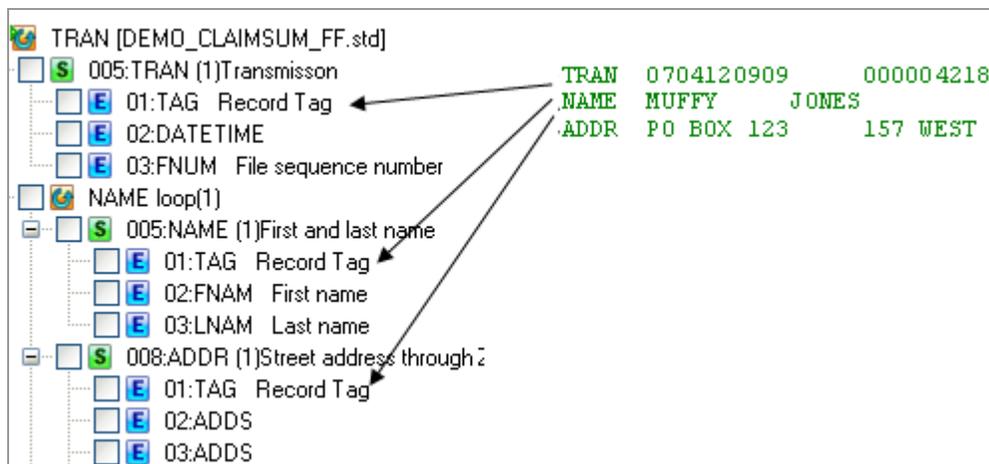


To specify that the record identifier is in this position, set this up under **Settings | Source Guideline Settings**



When you do not need to identify a record key

You do not need to specify record keys for flat files where the first field is a record key, like this:



Flat File Source - Format B

Use this when the source data is fixed or delimited without a key, and each line in the source should generate multiple lines in the output.

Example

This input:	Might generate this output:
11111 AAAAA	ENC*20110130~ LOC*EAST*DISTRICT55~ REP* 11111 ~ PROD* AAAAA ~
22222 BBBBB	ENC*20110130~ LOC*EAST*DISTRICT55~ REP* 22222 ~ PROD* BBBBB ~
33333 CCCCC	ENC*20110130~ LOC*EAST*DISTRICT55~ REP* 33333 ~ PROD* CCCCC ~

Please see DEMO6_flatB_to_EDI_270.map.

One Line without ID

Fixed Field Length

Delimited with:

```
AAABBBB BBBB CCCCC CDDDD FFFFF
AAABBBB BBBB CCCCC CDDDD FFFFF

AAA, BBBB BBBB, CCCCC, DDDD, FFFFF
AAA, BBBB BBBB, CCCCC, DDDD, FFFFF
```

Fixed Field Length

Source fields are fixed length.

Delimited with

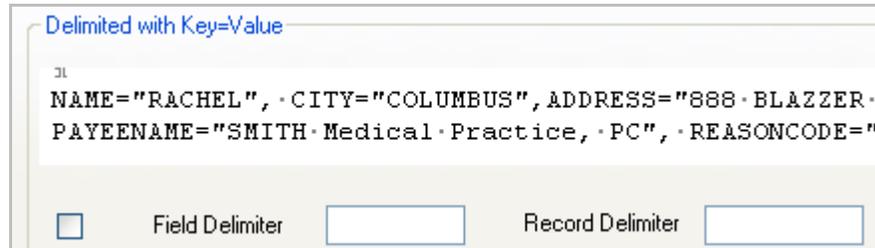
Source fields are delimited. Type the keyboard character or insert hexadecimal codes, prefixed with 0x, for ASCII equivalents up to 255.

Flat File Source - Format C

Use this when the source data records are in this format:

key = "value", *key* = "value", *key* = "value" ...

See DEMO10_850_FlatDelimited.map.



Delimited with Key=Value

```
NAME="RACHEL", CITY="COLUMBUS", ADDRESS="888 BLAZZER·  
PAYEENAME="SMITH·Medical·Practice, PC", REASONCODE="
```

Field Delimiter Record Delimiter

Field Delimiter

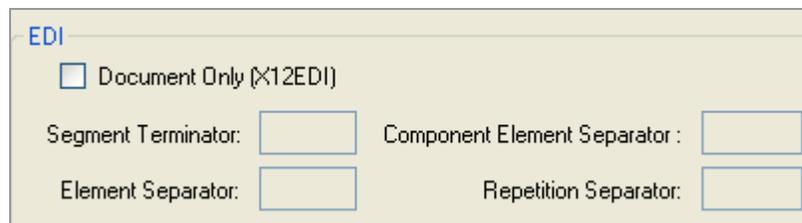
Type the source data's field delimiter or hexadecimal codes, prefixed with 0x.

Record Delimiter

This can be an ASCII character up to 255. Type the keyboard character or the hex value preceded with 0x. The Delimiter is usually 0x0D.

EDI Source Settings

Under **Settings | Source Guideline Settings...**, you can specify delimiters and separators if the file does not contain enveloping.



EDI

Document Only (X12EDI)

Segment Terminator: Component Element Separator:

Element Separator: Repetition Separator:

Document Only

Select this if the EDI source has no enveloping. You will then need to specify the source delimiters and separators.

Segment Separator

Element Separator

Component Element

Repetition Separator

These are available if you have selected Document Only. They can be ASCII characters up to 255. Type the keyboard character or the hex value preceded with 0x. Segment Terminator is usually 0x0D.

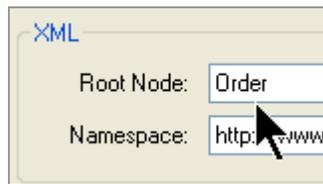
XML Schema Settings

For XML only.

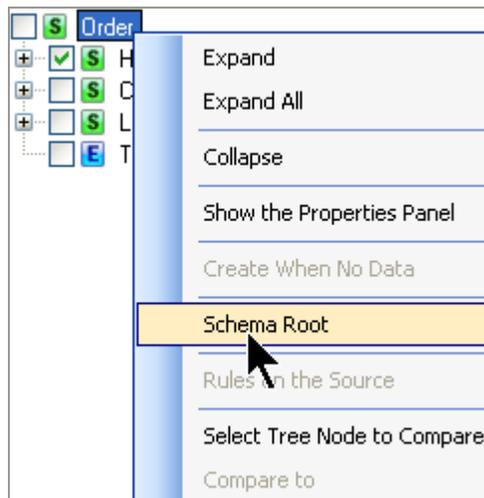
Root Node

Translator must know what the schema root is for XML sources or targets. Check to see if it was able to pull this information from the target schema:

1. Under **Settings | Source Guideline Settings...** or **Setting | Target Guideline Settings...**, be sure Root Node is filled out:



2. If not, type it or exit settings, right-click on the XML schema's top line and select **Schema Root**:



Namespace

Namespace is pulled from the target schema. It is a read-only field and cannot be changed.

Allow Delimiter as Part of Data

For HL7/Amadeus/Flat Files only.



HL7 / Amadeus / Flat File
 Allow Delimiter As Part Of Data

HL7/Amadeus and Flat File targets have default delimiters and you cannot change them. Checking **Allow Trailing Delimiter as Part of Data** lets the output contain trailing delimiters like this:

```
PV1|||I|^802^1|||8625^Physician^Michael|86-7468^||CR|||||||
```

Allow CR/LF as Part of Data

Remove or retain Carriage Return/Line Feeds (CR/LF) during translation.



Allow CR/LF As part Of Data

Unchecked – CR/LFs are removed during translation (default).

Checked – CR/LFs are retained during translation.

For example, using this data:

```
abcdef CRLF  
ghijkl CRLF
```

If not selected, data will translate with the CR/LFs removed.

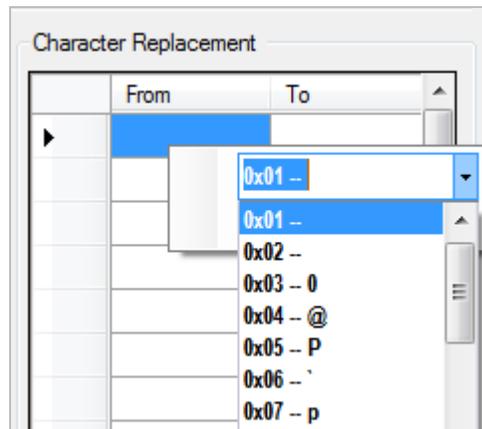
```
abcdefghijkl
```

If selected, the data is translated exactly as shown, including CR/LFs.

```
abcdef CR|LF  
ghijkl CR|LF
```

Character Replacement

For XML Source only. Replace a hex value with another hex value.



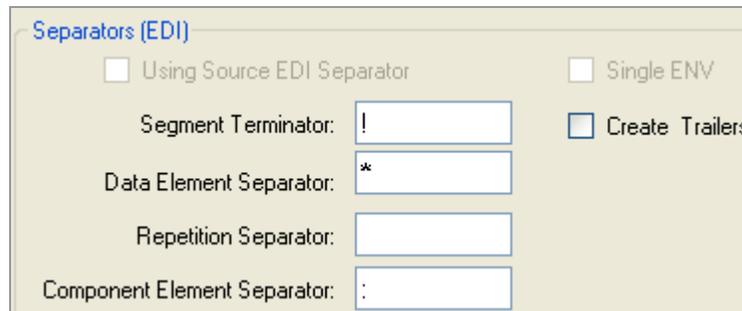
Type the hex value preceded with 0x for a total length of four characters, such as 0x0A.

Example: If the "To" character is set to "0x00," the "From" character will be removed.

Target Guideline Settings

EDI Target Settings

With an EDI target, you must make a selection in the Separators (EDI) section:



Separators (EDI)

Using Source EDI Separator Single ENV

Segment Terminator: Create Trailers

Data Element Separator:

Repetition Separator:

Component Element Separator:

Otherwise, the output will contain strange separators and delimiters like this:

ISAÿ00ÿ ÿ00ÿ ÿ01ÿ9012345720000 ÿ01ÿ9088877320000 0ÿ:

Using Source EDI Separator

Target separators will be the same as those in the source data.

Single ENV

Each input line is to create a whole interchange or whole transaction.

Create Trailers

On EDI targets, this will create the SE, GE, and IEA if the elements in these segments have PostCreate rules. See DEMO10_850_FlatDelimited.map.

Segment Terminator

Data Element Separator

Repetition Separator

Component Element Separator

For each, type the keyboard character or the hexadecimal code prefixed with Ox.

Set to Default

Currently, HL7 targets have default separators and you cannot change them. Selecting the **Set To Default** button automatically populates the separators fields with the defaults.

separators (HL7Enable)

Using Source EDI Separator Single ENV

Segment separator Create Trailers

Field separator

Field repeat separator

Component separator

Sub-component separator:

Output Format Settings

Wrap Data

All data types except XML.

Under **Settings | Target Guideline Settings...**, **Wrap Data** removes all line breaks and puts all output on one line, even though it may appear to be broken into separate lines if viewed in a text editor.

Output Format

Wrap Data

Output All in Upper Case

Output All in Lower Case

If not selected, each segment or record starts on a separate line.

Wrap Data is not available for XML output.

Output all in Upper / Lower Case

Output will match the case of the input unless you choose an output case under **Settings | Target Guideline Settings...**:

Output Format

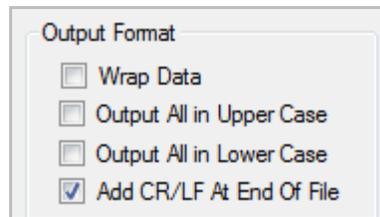
Wrap Data

Output All in Upper Case

Output All in Lower Case

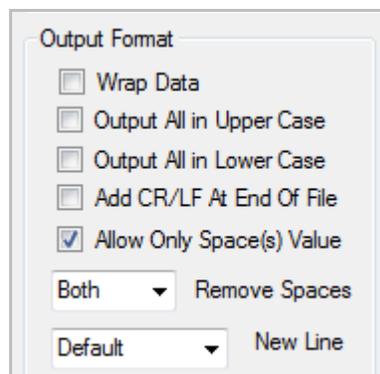
Add CR/LF At End Of File

Output will have a trailing CR/LF if you select this under **Settings | Target Guideline Settings....** This works for all data types except XML.



Allow Only Space(s) Value

Specifies how to translate data when values contain a space.

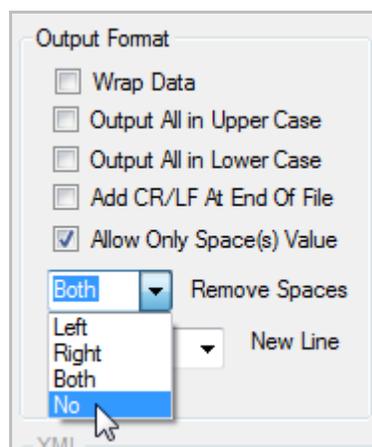


Unchecked – data containing spaces as values is translated as empty.

Checked – data containing spaces as values is translated as containing a value and the value is a space.

Remove Spaces

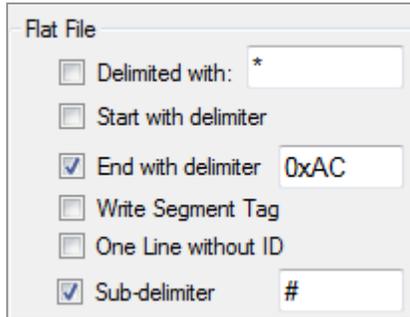
Translator can remove leading or trailing spaces from output if you select it under **Settings | Target Guideline Settings ... | Remove Spaces.** This works for all data types.



Delimiters and End-of-Line Handling

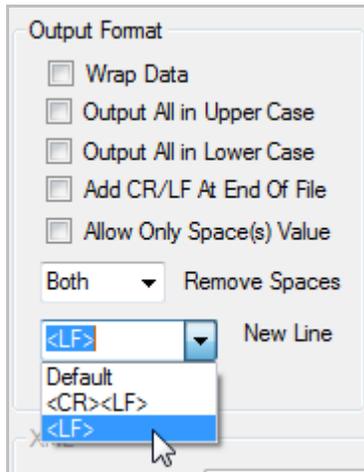
The element delimiter or sub-delimiter in a source or target can be an ASCII character up to 255. Type the keyboard character or the hex value under **Settings | Source Guideline Settings** or **Settings | Target Guideline Settings**.

Example:



For end of line handling in the target, choose <CR><LF> or <LF> under **Settings | Target Guideline Settings**.

Example:

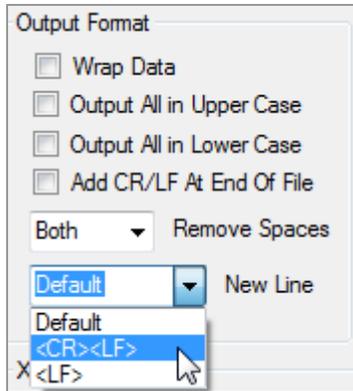


Defaults for both file-based and memory-based output:

Unix: 0A
Windows: 0D0A

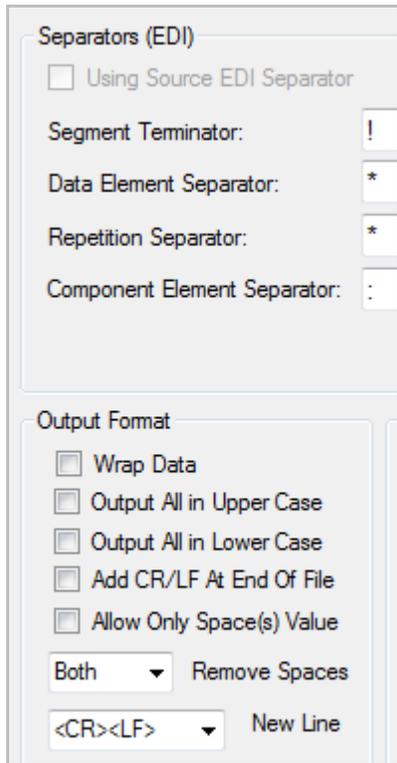
New Line

This lets you select the newline character for flat file or EDI output. This character will appear after the segment terminator or record terminator.



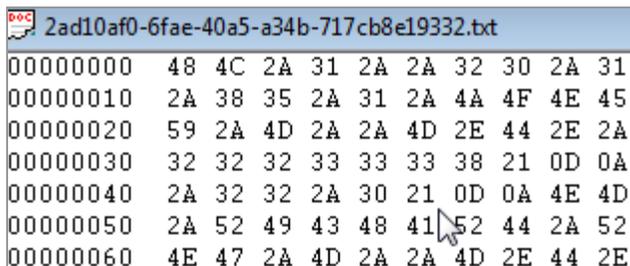
Default: Windows <CR><LF>
 UNIX <LF>

Example 1 – Keyboard segment terminator and <CR><LF> for New Line:



Output will appear like this in a text editor: HL*1**20*1!

Hex output will be like this, where the 21 is !, the 0D is <CR>, and the 0A is <LF>.



Example 2 – Hex segment terminator and <LF> for New Line:

Separators (EDI)

Using Source EDI Separator

Segment Terminator: 0xAC

Data Element Separator: *

Repetition Separator: *

Component Element Separator: :

Output Format Flat

Wrap Data

Output All in Upper Case

Output All in Lower Case

Add CR/LF At End Of File

Allow Only Space(s) Value

Both Remove Spaces

<LF> New Line

Output will appear like this in a text editor: HL*1**20*1~NM1*85*1 ...

Hex output will be like this, where the AC is the ~ and the 0A is <LF>.

Address	Hex Data
00000000	48 4C 2A 31 2A 2A 32 30 2A 31 C2 AC 0A 4E 4D 31
00000010	2A 38 35 2A 31 2A 4A 4F 4E 45 53 2A 4D 55 46 46
00000020	59 2A 4D 2A 2A 4D 2E 44 2E 2A 32 34 2A 31 31 31
00000030	32 32 32 33 33 33 38 C2 AC 0A 48 4C 2A 32 2A 31
00000040	2A 32 32 2A 30 C2 AC 0A 4E 4D 31 2A 44 4E 2A 31
00000050	2A 52 49 43 48 41 52 44 2A 52 45 46 45 52 52 49
00000060	4E 47 2A 4D 2A 2A 4D 2E 44 2E 2A 32 34 C2 AC

Flat File Target Settings

Under **Setting | Target guideline**, specify:

Delimited with

Type the character that terminates each field. This can be a keyboard character, or hexadecimal codes prefixed with Ox.

If you do not specify a delimiter for delimited flat file targets, delimiters specified in the guideline are used. Set these in EDISIM Standards Editor by opening the guideline and choosing **File | Properties | User-Defined Standard**.

Start with delimiter

Select this if you want each target record to start with the delimiter that you specified. This will go after the record tag if you Write Segment Tag.

End with delimiter

Select this if you want each target record to end with the delimiter that you specified.

Write Segment Tag

Determines whether the target's segment/record tag is in the output. In this example target, the tags are NAMEREC and ADDRREC.

Target Guideline	Output with Write Segment Tag	Output without Write Segment tag
<ul style="list-style-type: none"> S 010:NAMEREC (1) identification o <input checked="" type="checkbox"/> E 01:NAME <input checked="" type="checkbox"/> E 02:IDNUM S 020:ADDRREC (1) Address of Per <input checked="" type="checkbox"/> E 01:ADDR1 <input checked="" type="checkbox"/> E 02:ADDR2 <input checked="" type="checkbox"/> E 03:CITY <input checked="" type="checkbox"/> E 04:STATE <input checked="" type="checkbox"/> E 05:COUNTRY 	<p>NAMERECKAVERCORP^1122212</p> <p>ADDRREC682 FALL RIV PL^^^ ^</p>	<p>KAVERCORP^1122212</p> <p>682 82 FALL RIV PL^^^ ^</p>

One Line without ID

Future development.

Sub-delimiter

Type the character to be used as a sub-delimiter. This can be a keyboard character, or hexadecimal codes prefixed with Ox.

If you do not specify a sub-delimiter for delimited flat file targets, sub-delimiters specified in the guideline are used. Set these in EDISIM Standards Editor by opening the guideline and choosing **File | Properties | User-Defined Standard**.

Right Justify data (R and N type)

By default, Flat Files are left justified. Select this option if you want to right justify R/N type data elements.

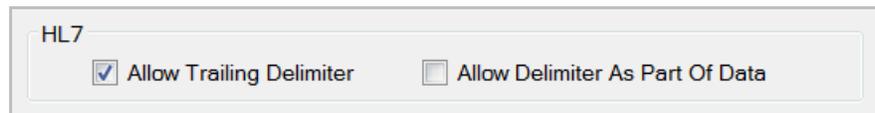
XML Schema Settings

See [XML Schema Settings](#) on page 38.

HL7 Target Settings

For HL7 only.

Currently, HL7 targets have default delimiters and you cannot change them. Translator provides two options for HL7 targets.



HL7

Allow Trailing Delimiter Allow Delimiter As Part Of Data

Allow Trailing Delimiters:

Checked – Allows trailing delimiters.

A|B|C|||

Unchecked – Removes trailing delimiters.

A|B|C

Allow Delimiter As Part Of Data:

Checked – Allows data to contain trailing delimiters.

PV1||I|^802^1||||8625^Physician^Michael|86-7468^||CAR|||||||

Unchecked – Removes trailing delimiters from data.

Two Pass Mapping

Check this box if you want Translator to go through your data twice during mapping:

2 Pass Mapping

See [One Pass and Two Pass Mapping](#) on page 24 for more detailed information.

6 Testing a Map

Introduction to Testing

Test your mapping frequently. This makes it much easier to verify your results. Do not wait until the end to test, especially if you are using rules.

You have two testing options

- [Testing from Translation Tool](#) – when developing your maps
- [Testing from a Batch File](#)

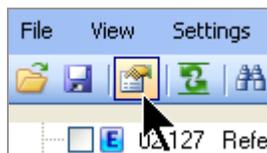
Translation Tool is for testing when developing your maps. It is not for extremely large files.

To process large files, use a batch file like `TranslatorTestMap` in your Scripts directory.

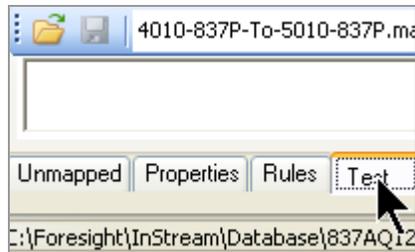
Testing from Translation Tool

To test from within Translation Tool:

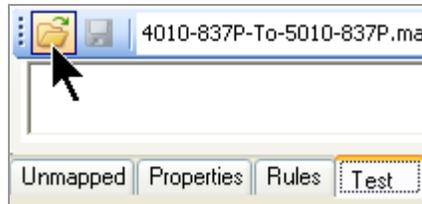
1. Click the **Show Properties Panel** toolbar button.



2. At the bottom, click the **Test** tab.



3. Click the **Open an Input File** button on the test panel.

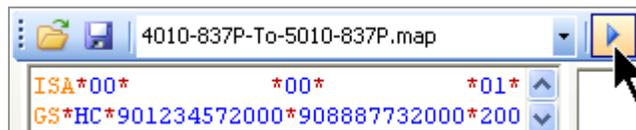


... and pick a test data file that contains the mapped fields.

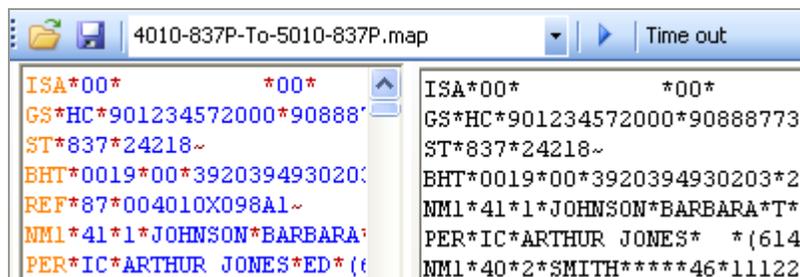
4. Check that the correct map name is showing.



5. Click **Run**.



6. Look at the pane to the right to see the values that the map will put in the translated file.



If the output EDI shows odd delimiters, like these:

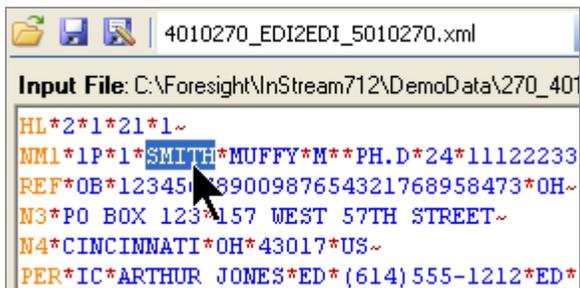
ISAÿ00ÿ ÿ00ÿ ÿ01ÿ9012345720000

... then choose **Settings | Target Guideline Settings ...** and fill out the Separators (EDI) section.

Editing In Test Data Input Pane

In Translation Tool's Test pane, you can edit the input data directly:

1. Click in the data and start typing.



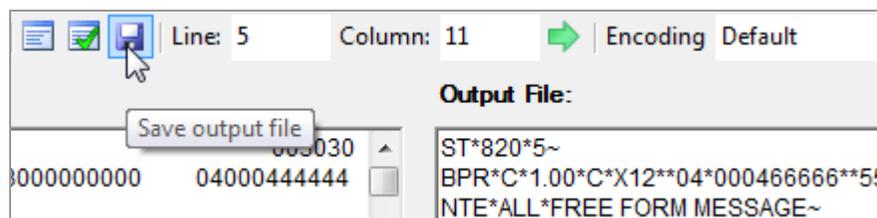
2. Click **Save** or **Save As** on the test pane's toolbar.



3. Click the Run button  to translate again.

Saving Translated Data

To save the translated data in the Output File pane, click this Task pane toolbar button.

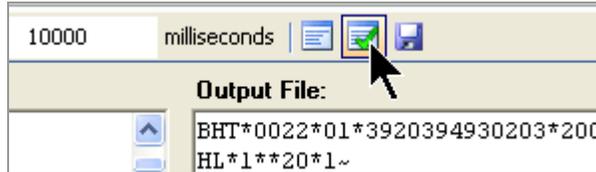


Translation Tool also puts the output and a log in Foresight Translator's `\bin\fslog` directory. This is cleared when you close Translation Tool.

Validating Translated Data

Requires Instream.

Translation Tool can validate translated data against the target guideline if the target data is EDI or XML. After translating, click this Task pane's toolbar button:



This button appears if you have set **View Validate Button** to **True** under **Settings | Translation Tool Settings ... | Other Settings**.

Understanding Validation Results

Please see [APF.pdf](#) for a list of severities and types.

Summary

Ignore Count	0	Info Count	12	Type 0 count	12	EDI Syntax Count	0
Warning Count	0	Error Count	0	Syntactical	0	Balancing Count	0
Fatal Count	0	User1 Count	0	Product Count	0	CodeSet Count	0
User2 Count	0			Situation Count	0	Payer Count	0

Number of errors for each severity (points to Ignore, Warning, Fatal, User2)
 Number of errors for each HIPAA type (points to Type 0, Syntactical, Product, Situation)

Analysis requested on file
C:\Foresight\TranslatorStandalone\Bin\fslog\0c00919e-3e40-4c84-bd33-9165b4d7e098.txt, 2026 bytes long

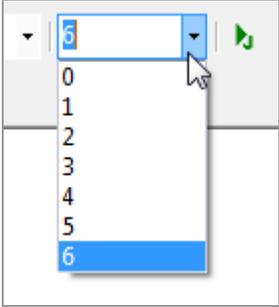
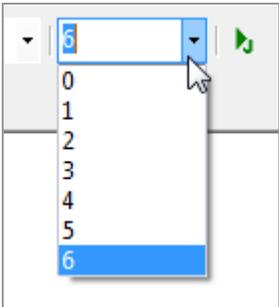
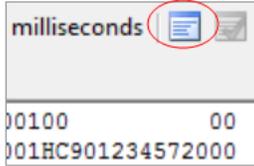
<i>Messages</i>	Message file loaded : C:\Foresight\Instream713 \bin\FSANERRS.TXT
	Message file loaded : C:\Foresight\Instream713 \bin\FSBRERRS.TXT

Comparing Source and Target Data

If you have Beyond Compare installed, you can compare the source and target data from within Translation Tool. This is only available if your data is EDI and/or flat file.

1. Go to **Settings | Translation Tool Settings ... | Other Settings** and set **View Beyond Compare Button** to True.
2. Translate the data.
3. Click this button on the Test pane's toolbar: 

Debugging and Logging

	Translation Tool	Translator	Java API
Log created	Set to non-0 in Test Pane: 	Created if -r command line parameter is used. See Translator Command Line on page 63.	Set both these environment variables: <ul style="list-style-type: none"> ▪ FS_TRANSLOGDIR – the path for the log ▪ FS_TRANSLOGGINGLEVEL - amount of information written to the log. Default is 1. See -r in Translator Command Line on page 63. Example: <pre>@SET FS_TRANSLOGGINGLEVEL=6 @SET FS_TRANSLOGDIR=c:\logs</pre>
Log location	Translator's bin/fslog directory	Same directory as Translator output.	See above.
Log filename	Long generated filename	Same name as output file, but with _report.txt appended.	Unique name is automatically created.
Log level	Set in Test Pane: 	-rn, where n is a digit from 1-6 (see -r in Translator Command Line on page 63).	Set FS_TRANSLOGGINGLEVEL – see above.
Viewing logs	Test tab's View Log button:  To make this button appear: Settings Translation Tool Settings ... Other Settings View Log Button → Set to True	Look at the log file in the same directory as Foresight Translator output.	Look at log file in directory specified with FS_TRANSLOGDIR environment variable.

	Translation Tool	Translator	Java API
Deleting logs	Settings Translation Tool Settings ... Other Settings Delete All Log Files → Set to True This empties Foresight Translator's \bin\fslog directory when you exit Translation Tool	Manually delete.	Manually delete.

Testing from a Batch File

You can use your own batch file containing a Translator command line or modify **TranslatorTestMap.bat** in Translator's Scripts directory.

The advantage of this method is that you can get a log file that is useful for debugging.

To use TranslatorTestMap.bat:

1. Edit the file with a text editor.
2. Update these lines:

Set Instreamroot	Translator's high-level directory.
set ediin	Path and filename of the source document.
set newedi	Path and filename of the output document created by Translator.
set map	Filename of the map, including the file type. Do not include the path. It must be in Translator's DemoData directory.
3. Save and execute the file.

You will get the translated file and log file in the same directory.

7 Encoding Options

Foresight Translator can convert a data value between Base64 and text. Please see [Rules.Encoding](#) on page 148.

For whole-file encoding, Foresight Translator can encode output files as:

- ISO8859-1 (ASCII) (default)
- UTF-8
- UTF-16

For input files, a BOM (byte order mark) is required for Unicode documents (UTF-8 and UTF-16) except for XML. The set ID, segment ID, record_key and all standards-based code values must be in 0x21 to 0x7E range.

For output files, specify the encoding unless you want ISO8859-1:

Translation Tool	Use the Encoding drop-down on the Test tab.
Translator.exe	Use the -e command-line parameter.
API	Use configuration_flag

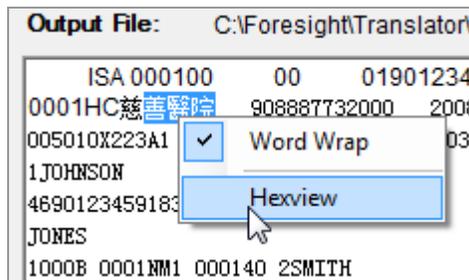
Encoding and Translation Tool

Output will be ISO8859-1 (ASCII) unless you use the Encoding drop-down list in the Test pane:



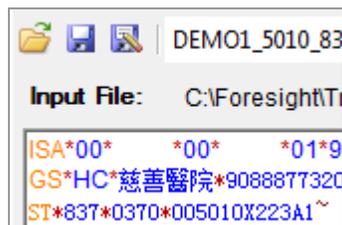
Viewing Hex characters in the Target pane:

Select the pertinent data, right-click, and choose **Hexview**:



Encoding Example

1. In Translation Tool, open **DEMO1_5010_837I_to_flat.map**.
2. In the Test pane, open **837I_5010_H_clean_encoding.txt** in Foresight Translator's \DemoData directory. Notice the Chinese characters in the GS02.



3. Select Encoding UTF-8:



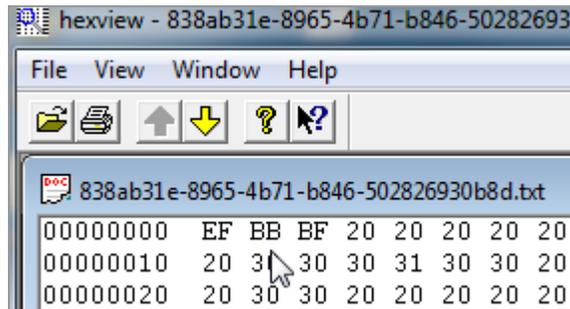
4. Run the translation.
5. Notice the output:

```

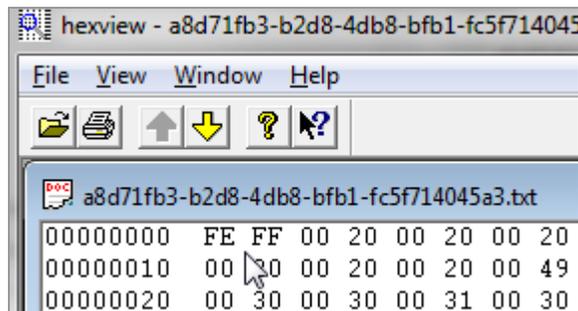
Output File:
ISA 000100 00 019012345720
GS 0001HC慈善醫院 908887732000
ST 00018370370 005010X223A1
BHT 00010019003920394930203

```

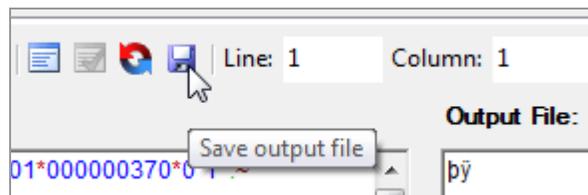
6. Go to Foresight Translator's \bin\fslog directory and open the most recent .txt file in a Hex editor. Notice the EF BB BF that designates UTF-8 encoding:



7. Now repeat the same translation with UTF-16. Notice the FE FF that designates UTF-16 encoding:



Be aware that the **Save Output** button does not necessarily save the data as encoded:



Encoding and Translator.exe

Output will be ISO8859-1 (ASCII) unless you use the `-e` command-line parameter to control encoding:

```
-eUTF-8  
-eUTF-16  
-eISO8859-1
```

If the `-e` parameter is omitted, Foresight Translator will write out the target file with the same encoding as the input file.

Example

```
C:\Foresight\Translator\bin\Translator.exe  
-i"C:\Foresight\Translator\DemoData\837I_5010_H_clean_encoding.txt"  
-o"C:\Foresight\Translator\Output\837I_5010_flat_clean_encoding.txt"  
-r6  
-eUTF-8 -tDEMO1_5010_837I_to_flat.map
```

For a command-line demo, run `T_837I_5010_targetUTF-8` in Foresight Translator's `\scripts` directory.

Encoding and Java API

Default behavior

Memory-based input/output

Output always produces a UTF-8 stream. The end user is responsible for changing the encoding if needed.

File-based input/output

Output is ASCII by default; and can be changed with `configuration_flag`.

Example: `configuration_flag |= UTF16_ENCODIN`

8 Running Foresight Translator

Foresight Translator Command Line

Translator.exe is the command line executable that performs the translation.

It is a standalone executable in Foresight Translator's \bin directory. Control returns when the translation completes.

The command line has the following case-sensitive parameters. Use quotation marks around paths or filenames that contain spaces.

Translator.exe *-iInFile -oOutFile -rn -eEncoding -tTranslationMap -cConfigPath -gVariable -version -pPartnerAutomationFile*

Important	All options are case sensitive Spaces are significant Use double quotation marks around paths that have spaces
Translator.exe	Path to Translator.exe. Example: C:\Foresight\Instream\Translator.exe
i	i nput file - path and filename of a file to be translated. Example: -iC:\EDIin\837p_20100130_512.edi
o	o utput file - path and filename for the translated file. The directory must already exist. If the output files already exist, they will be overwritten. Example: -oC:\XMLout\837p_20100130_512.xml

<p>r</p>	<p>report file (logging) used, where n is an integer from 1-6 to specify the logging level:</p> <ul style="list-style-type: none"> 1 Error only 2 Generic information 3 Parsing source data 4 Building target data 5 Business rules 6 All <p>This file goes to the same directory as the translated output. See Debugging and Logging on page 56.</p>
<p>e</p>	<p>encoding for the output file, one of these:</p> <ul style="list-style-type: none"> ISO8859-1 (default) UTF-8 UTF-16 <p>If omitted, you will get:</p> <ul style="list-style-type: none"> XML target – UTF8 Other targets - Text
<p>t</p>	<p>translation map – path and filename. This is the map created by Translation Tool.exe. Omit if you want to use partner automation to select the map.</p> <p>Example: -tc:\maps\837P-to-XML.xml</p>
<p>c</p>	<p>configuration file location – the high-level directory containing \$dir.ini (Windows) or fsdir.ini (UNIX).</p>
<p>g</p>	<p>Specify one or more variables and associated values for use with Foresight Translator.</p> <p>Format:</p> <p>Variable=value;variable=value</p> <p>Example: reportfile=c:\tibco\translator\3.7\output\report.txt;</p>

1

Specify segment, element, composite, and element repetition delimiters to use in the output EDI file. Each delimiter may be the actual character or the ASCII number representing the character. The ASCII number may be hexadecimal (by starting the number with 0x) or decimal.

If a new-line sequence is to be used as a segment terminator, set the Segment Delimiter to zero (0).

Default delimiters

Segment: ~
Element: *
Composite: :
Element repetition ^

Format:

```
segment_terminator<,>element_separator<,>composite_separator  
<,>repetition_separator
```

Where:

<,> A comma is needed to separate hexadecimal and decimal numbers only. No comma is needed for character entries.

Notes:

1. Avoid using characters that have special meaning on your operating system, such as % (Windows) and \$ (Unix).
2. You must specify delimiters for the first three fields; only the repetition separator field can be left blank.
3. Do not mix character types.

Example 1:

```
-1|\#*
```

Entries are characters. No comma is needed to separate entries.

Segment terminator = |, element separator = \, composite separator = #, repetition separator = *

Example 2:

```
-1|\\#*
```

In the absence of a subelement separator, use two slashes to distinguish it from the escape character slash.

Example 3:

```
-1#,!+
```

The comma character can be used as a delimiter.

(continued)

<p>l (con't)</p>	<p>Example 4: <code>-l0x33,0x34,0x35,0x23</code> Entries are hexadecimal. Commas are required to separate entries. Segment terminator = 0x33, element separator = 0x34, composite separator = 0x35, repetition separator = 0x23</p> <p>Example 5: <code>-l 0x1E, , ,</code> Invalid! You must specify delimiters for the first three fields; only the repetition separator field (the last field) can be left blank. Segment terminator = the RS control character (ASCII 30)</p> <p>Example 7: <code>-l 30,0x1E, , *</code> Invalid! Do not mix character types. Use all hexadecimal, ASCII, or characters.</p>
<p>version</p>	<p>Displays the release/version of the Translator software.</p> <p>Example: <code>-version</code></p> <p>Sample output: Translator version 3.4.0 [Build 100r(32 bit): 05/21/2014]</p>
<p>p</p>	<p>Path and filename to your partner automation file.</p>

Example

This example is actually all on one line, but it is broken into separate lines here for readability:

```
C:\Foresight\InStream\Bin\Translator.exe
-i"C:\Foresight\InStream\DemoData\5010-file1_edi.txt"
-o"C:\Foresight\InStream\output\5010-file1_flat.txt"
-tFS_5010_837I_to_flat.xml -r
```

Partner Automation

For X12 source only.

Creating a Lookup File

Create the lookup file, using the name that you specified above. It is the same as a validation setup file except that it identifies a map rather than a guideline. The selection criteria are the same as those in the validation partner automation file, but the only thing selected is the map. Please see **TIB_fsp-instream_<n.n>_tpa.pdf** for lookup file details.

Example lookup file

```
GS08/UNG7-12,GS01/UNB1-1,ISA05/UNB2-1,ISA06/UNB2-2,ISA07/ ...  
  
D.93A,UNOA,,,,,,,,,MYCUSCARtoFFdelim.xml,,
```

Using Partner Automation during Translation

Have Foresight Translator use the lookup file by omitting the **-t** parameter.

To Identify the lookup file, use one of these methods:

- In \$dir.ini or fsdir.ini:

Put a line like this in the [UserTables] section of your \$dir.ini or fsdir.ini file in Foresight Translator's \Bin directory, using the path and filename of your choice:

```
TSPARTNERAUTOMATION = "C:\Foresight\Translator\Bin\TSPartnerAuto.csv"
```

- On the command line, use **-p**.

Example: **-p"C:\Foresight\Translator\DemoData\Translator_TPA.csv"**

This overrides the setting in \$dir.ini.

Please see T_837I_5010_PartnerAutomation in Foresight Translator's \Scripts directory.

Temporary Files

Foresight Translator

Foresight Translator uses a temporary filename preceded with **FS_** while creating the output file and removes the **FS_** after it finishes translating the file. These always go in the directory where the output file is being created.

Translation Tool

When you test with Translation Tool, it stores output and logs in Foresight Translator's \bin\fslog directory. When you close Translation Tool, these are automatically deleted.

9 Creating, Viewing, and Testing Rules

Source vs. Target Rules

Rules are ways to manipulate the output besides a one-to-one mapping from a source field to a target field.

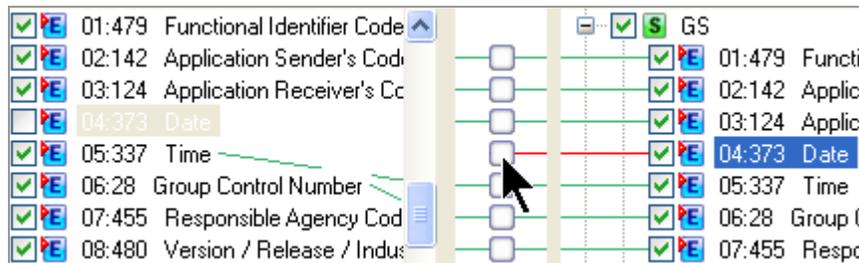
For additional examples, also see [List of Tutorials](#) on page 281.

You can put rules on the source or the target. Things to know:

- Foresight Translator processes the input file by moving down the source side, one item at a time, processing in this order:
 1. Source rules
 2. Target rules
 3. Write output
- For mapped items, most rules go on the source. In many cases, there is no difference in outcome, regardless of whether the rules are on the source or target.
- For unmapped items, ReplaceWith rules go on the target and other rules go on the source.

Target Rules

1. If the item is not mapped, drag a line from the target item's name to the center so that you have a small box like this:



If the item is mapped, you will already have the box.

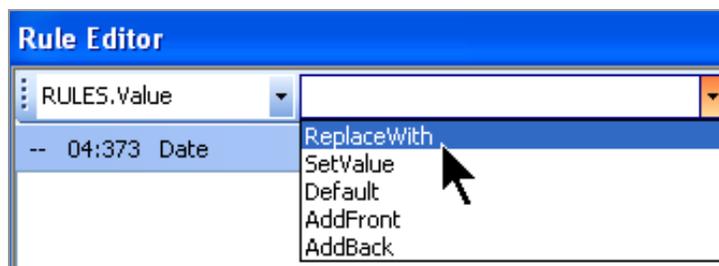
2. Click its box in the center pane.
3. At the bottom left in the rule box, be sure that **Target** is selected.



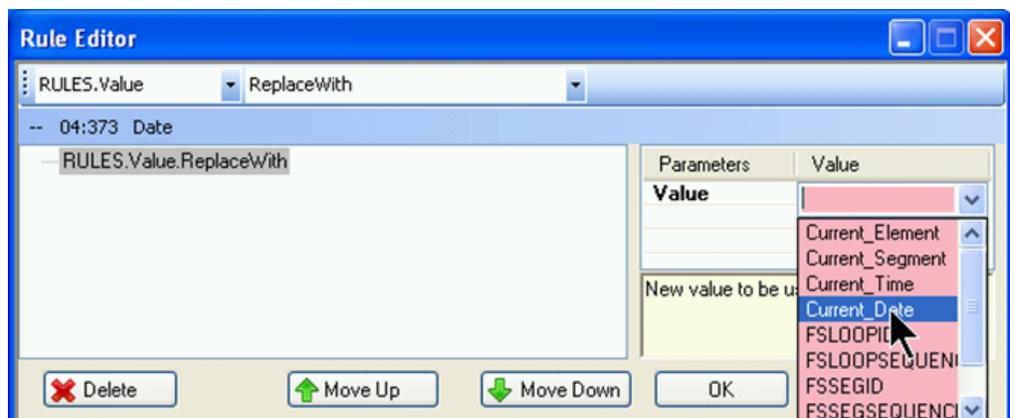
4. Click **Rule Editor**.



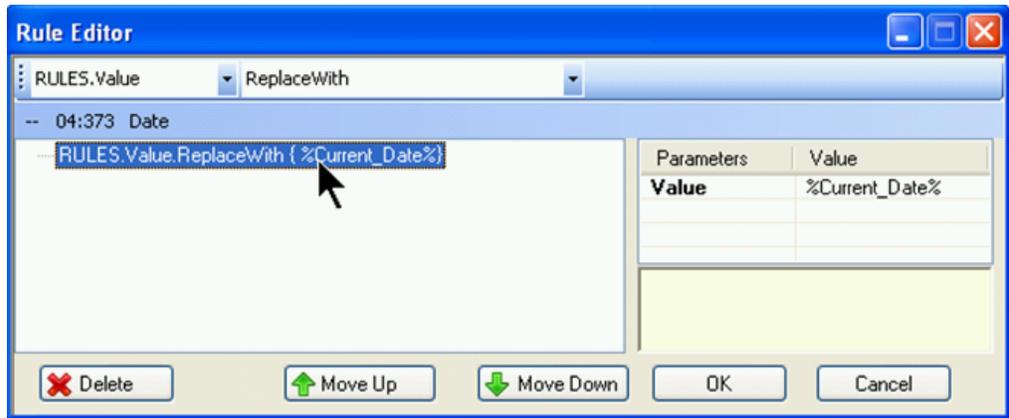
5. In the Rule Editor, use the two fields at the top to select a rule.



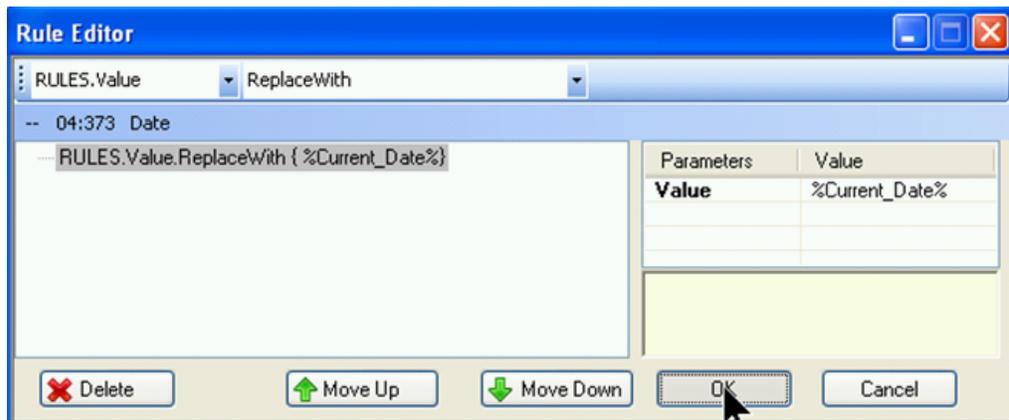
6. Type or select values for the parameters to the right.



7. Click on the rule in the left pane to update it.



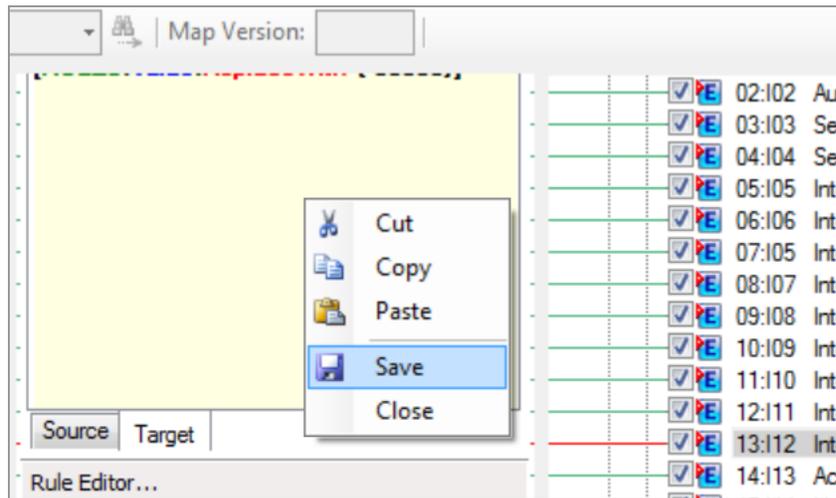
8. Click **OK** to close the rule editor:



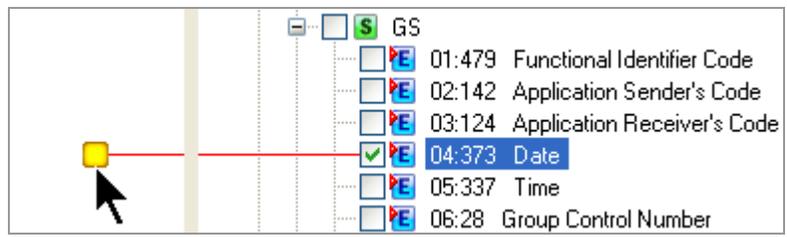
9. Save and close the main rule box:



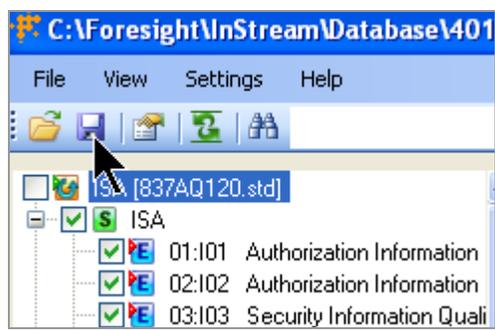
If this box is near the top of your screen, and the Save and Close buttons are not showing, right-click in the box to display a menu:



Since there is now a rule on the target, the small rule entrance box is yellow.



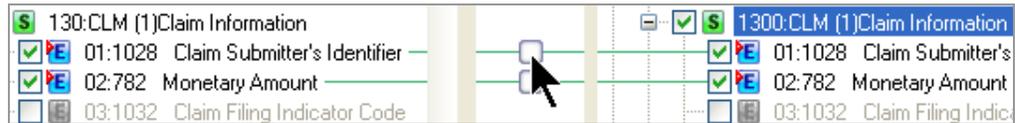
10. Save the map:



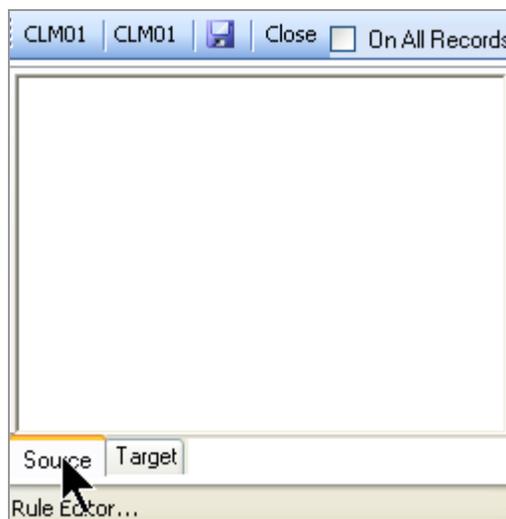
Source Rules

Creating a Rule on a Mapped Source Item

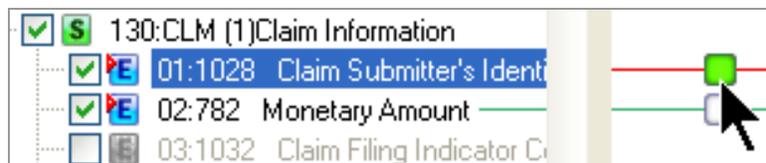
1. Click its box in the center pane:



2. At the bottom left in the rule box, click **Source**:



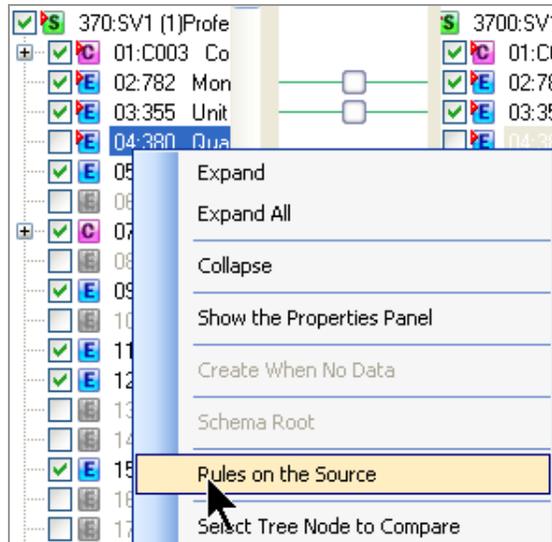
3. Click **Rule Editor...** and create the rule as usual.
4. Save and close the rule as usual.
5. The box in the center is now green, indicating a rule on the source:



If there is also a rule on the target, this box will be yellow instead of green.

Creating a Rule on an Unmapped Source Item

1. Right-click on the source object that is to get the rule and choose **Rules on the Source**.



2. Create and save the rule as usual.

Applying Rules to all like Element IDs

You can specify that a rule on an element should apply to all elements with the same element id in the target guideline.

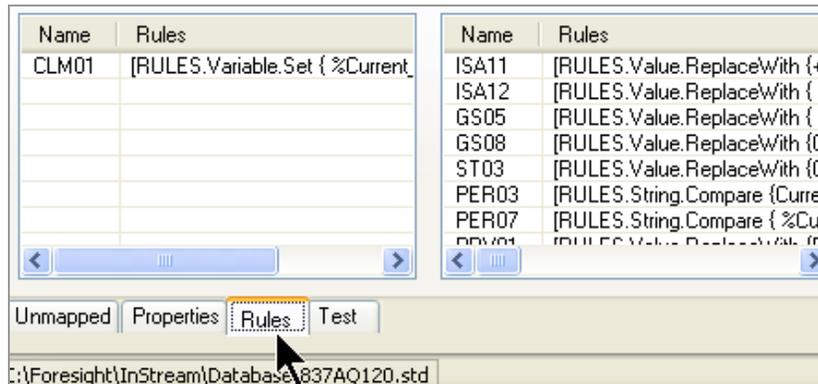
Viewing Rules

1. Display the **Properties** pane:



2. For source and target, expand all levels for which you want to see the rules.

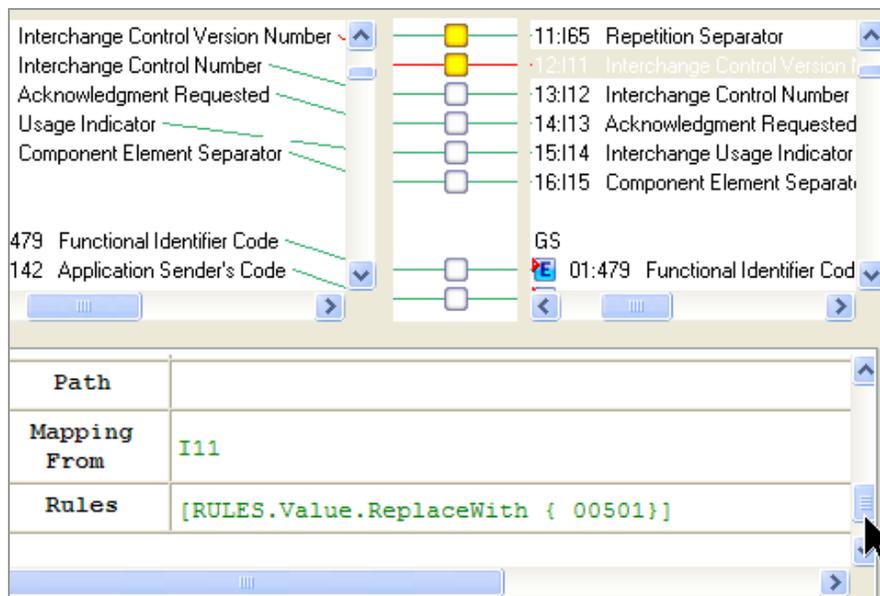
- At the bottom left, click **Rules**:



- Notice the list of source guideline rules on the left and target guideline rules on the right.
- To go to the object that has a particular rule, click on the rule in the bottom pane. Its item will be highlighted. If it has a mapping line, it will turn red.

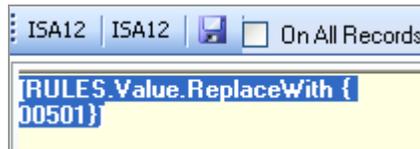
Seeing Rules on one Item

- Click on the source or target item.
- Display the Properties pane ().
- In the bottom pane, be sure the **Properties** tab is selected.
- Scroll down to see the rules.



Copying Rules

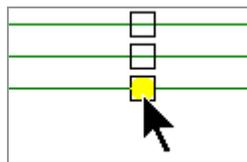
1. Open the rules box.
2. Select the rule.



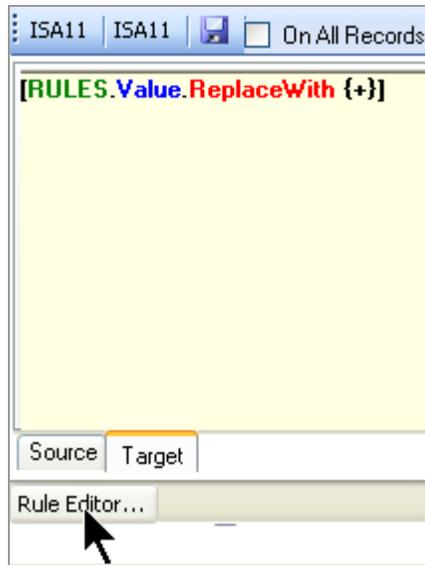
3. Press **Ctrl-C** or right-click and choose Copy. This puts the rule in the Windows clipboard.
4. Close the rule box.
5. Open the rule box where you want the copy to go.
6. If there are multiple rules, click where you want the rule to go.
7. Press **Ctrl-Y** to paste the rule.
8. Save.

Deleting Rules

1. Click the rule box in the center pane:



2. Click Rule Editor....

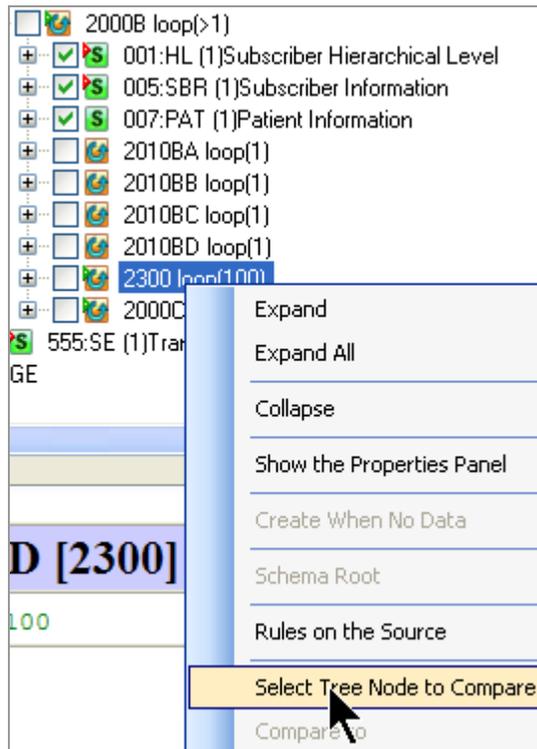


3. Click the rule.
4. Click the **Delete** button at the bottom left.
5. Click **OK** and then save the rule changes.

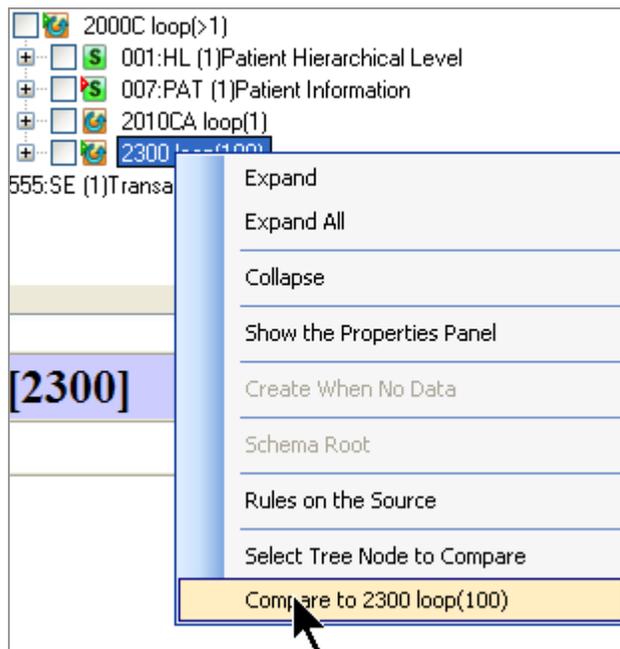
Comparing Rules

You can compare element rules between two segments or two loops that are on the same side (they are both source or both target):

1. Right-click on the first loop or segment and choose **Select tree node to compare**:



- Right-click on the second loop or segment and choose **Compare to** <first object>:



Using the comparison results dialog

Segment and element position (CLM01 = 1st element in CLM segment)

Item #1 being compared. Click to see full text below

Item #2 being compared. Click to see full text below.

Name	ISA [837AQ120.std]\GS\ST\2000A loop(>1)\%	Name	ISA [837AQ120.std]\GS\ST\2000A loop(>1)\%
CLM01	01:1028 Claim Submitter's Identifier	CLM01	01:1028 Claim Submitter's Identifier
CLM02	02:782 Monetary Amount	CLM02	02:782 Monetary Amount
DTP02	02:1250 Date Time Period Format Qualifier	DTP02	02:1250 Date Time Period Format Qualifier
DTP02	02:1250 Date Time Period Format Qualifier	DTP02	02:1250 Date Time Period Format Qualifier
NM109	09:67 Identification Code	NM109	09:67 Identification Code
SV102	02:782 Monetary Amount	SV102	02:782 Monetary Amount

RULES.Value.ReplaceWith { %FSLOOPSEQUENCE%}

Rule on SV102. Click to see full text below.

No corresponding rule on CLM01.

Detailed description: This image shows a comparison results dialog with two side-by-side tables. The left table lists elements from a source, and the right table lists elements from a target. Annotations with arrows point to specific elements and rules. One annotation points to the first element in the left table (CLM01) with the text 'Segment and element position (CLM01 = 1st element in CLM segment)'. Another points to the first element in the left table with the text 'Item #1 being compared. Click to see full text below'. A third points to the first element in the right table with the text 'Item #2 being compared. Click to see full text below.'. Below the tables, there are two scrollable areas. The left one contains the text 'RULES.Value.ReplaceWith { %FSLOOPSEQUENCE%}' and an annotation 'Rule on SV102. Click to see full text below.'. The right one is empty, with an annotation 'No corresponding rule on CLM01.'.

10 Rules Reference

Reserved Variables

Reserved variables are ones that do not have to be assigned in order for the system to determine the value. Except where noted otherwise, these can be used in business rule parameters anywhere a variable can be used.

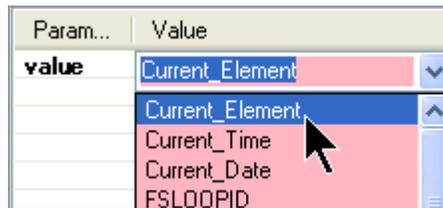
Variable	Page
Current Date	82
Current Element	83
Current Segment	84
Current Time	85
FS_ALLSPACE	86
FS_COMP_SEP	86
FS_ELM_SEP	87
FS_ELMID	88
FS_QUOT	88
FS_REP_SEP	89
FS_SPACE	90
FSID	91
FSLOOPID	92
FSLOOPSEQUENCE	93
FSSEGID	94
FSSEGSEQUENCE	95

[RECORDCOUNTER](#) 96

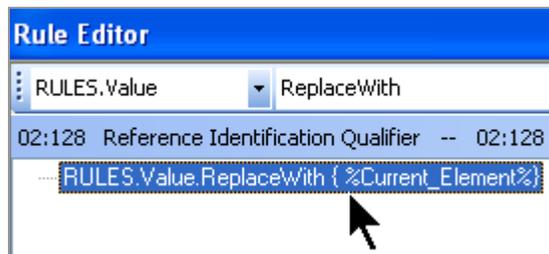
[Target Element](#) 97

To use a reserved variable:

1. In the rule editor, select it from the list.



2. The rule editor will surround it with % signs to show that it is a reserved variable:



You can also type reserved variables in the parameters area.

Do not attempt use these reserved names for other purposes.

Current_Date

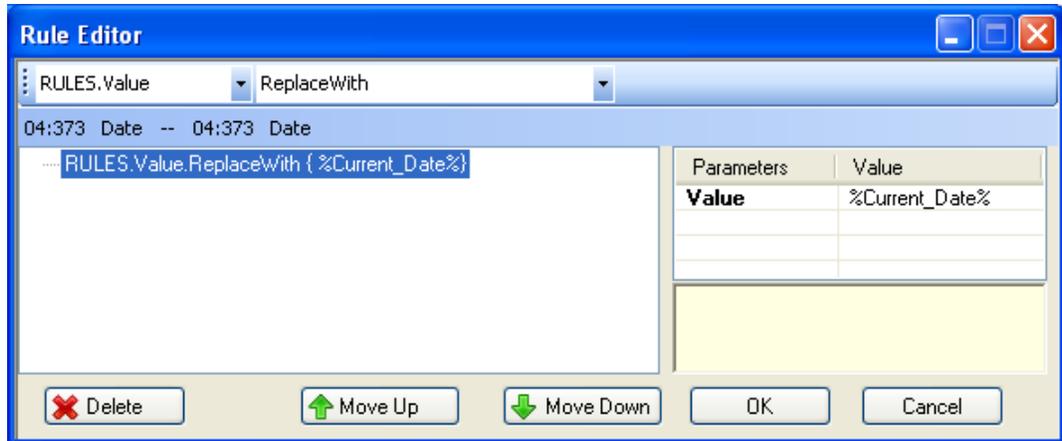
Represents the current date in *yyyymmdd* format. March 13, 2011 would be 20110313.

Format (case sensitive)

Current_Date

Example

Put the current date into the current element.



Current_Element

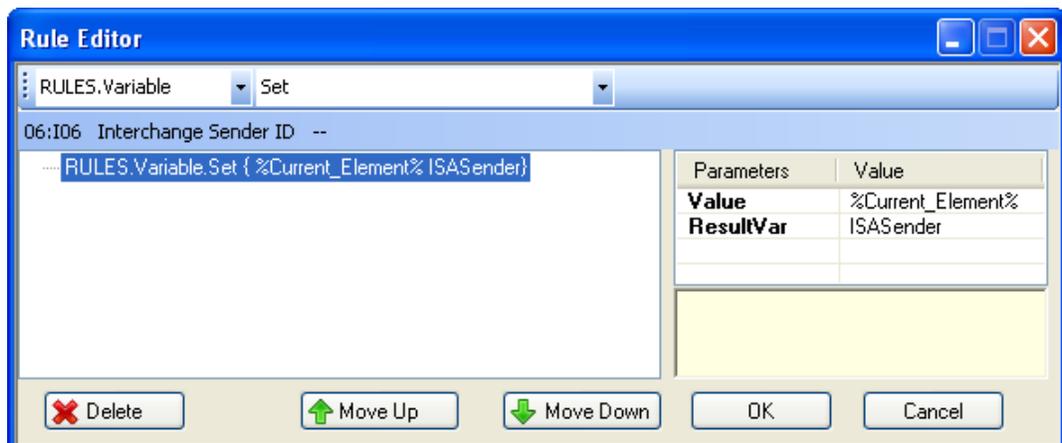
Represents the value in the current element or field.

Format (case sensitive)

Current_Element

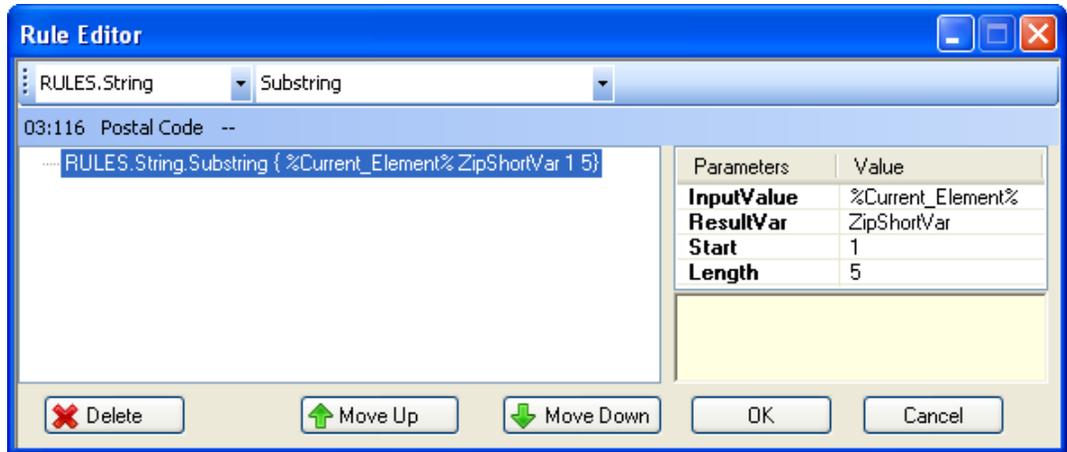
Example 1

This example places the value in the current element into variable **ISASender**:



Example 2

This example takes the first five characters of the current element and places them in variable **ZipShortVar**:



Current_Segment

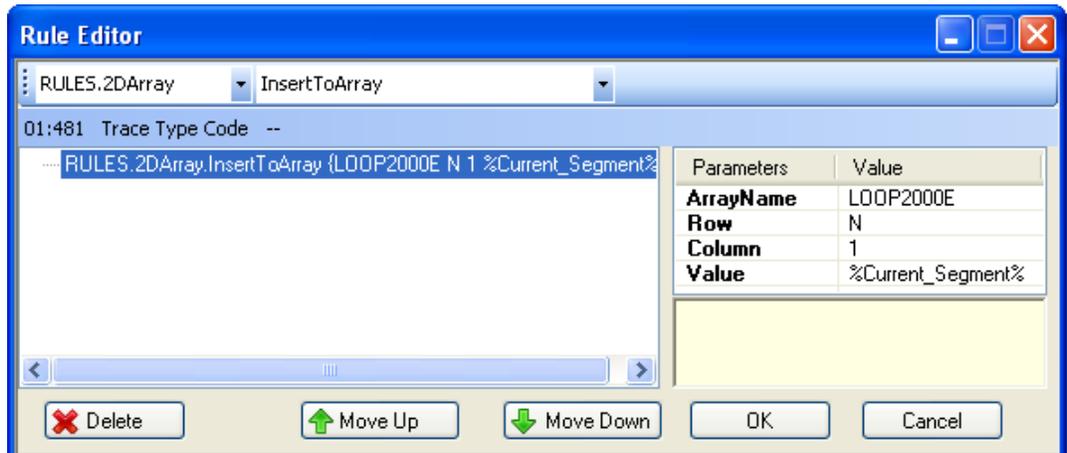
Represents the entire segment.

Format (case sensitive)

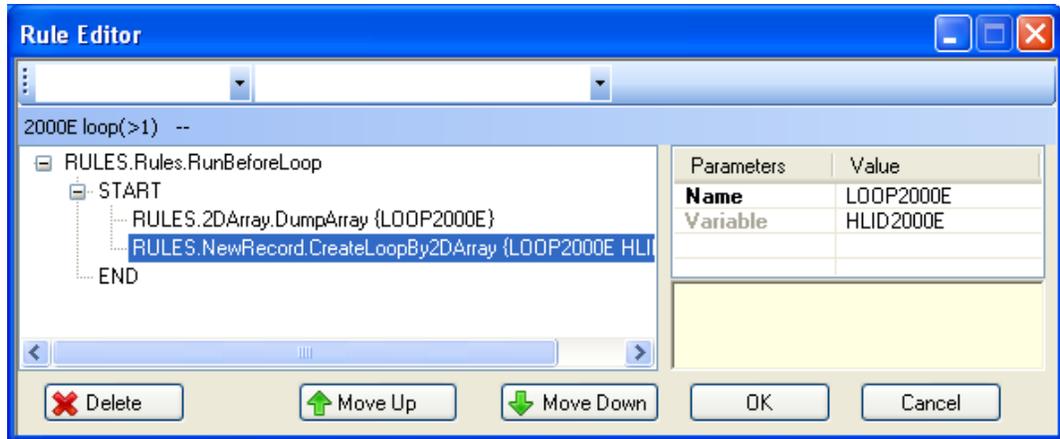
Current_Segment

Example

This rule puts the entire current segment, without a segment terminator, into column 1 of the next row (because Row = N in the parameters) of an array.



You could later write out the segments from the array or variable. HLID2000E is a variable that should be incremented for each row.



Current_Time

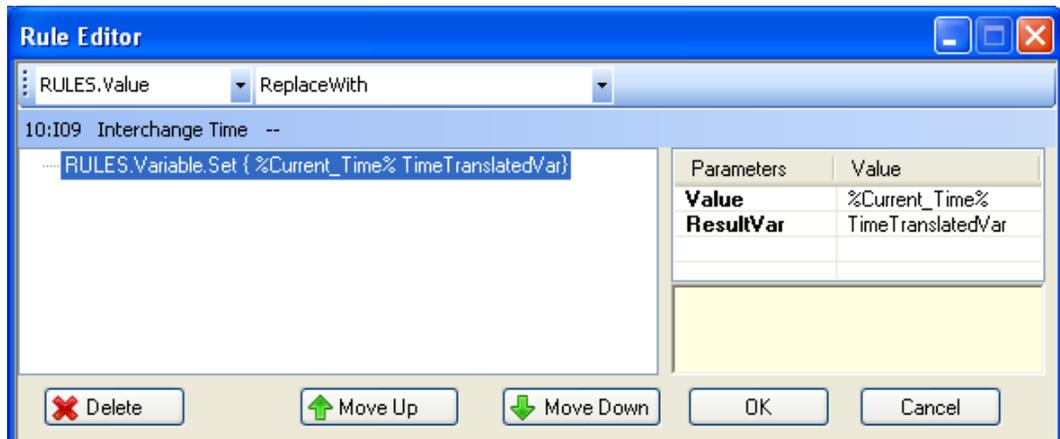
Represents the current time in *hh:mm:ss* format. 1:30 p.m. would be 13:30:00.

Format (case sensitive)

Current_Time

Example

This rule puts the current time into variable **TimeTranslatedVar**.



FS_ALLSPACE

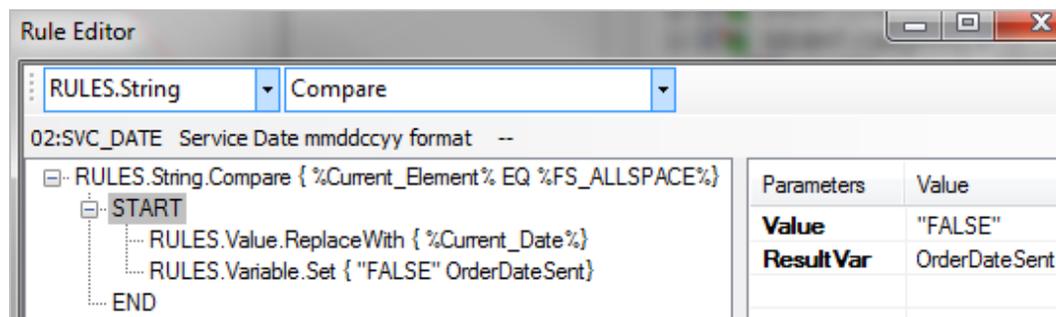
Determines whether the specified data is all spaces and takes action on a sub-rule based on the answer.

Format (case sensitive)

```
FS_ALLSPACE
```

Example

If the source data is all spaces, then the target will contain the current data and a variable will be set to indicate that an order date was not provided.



FS_COMP_SEP

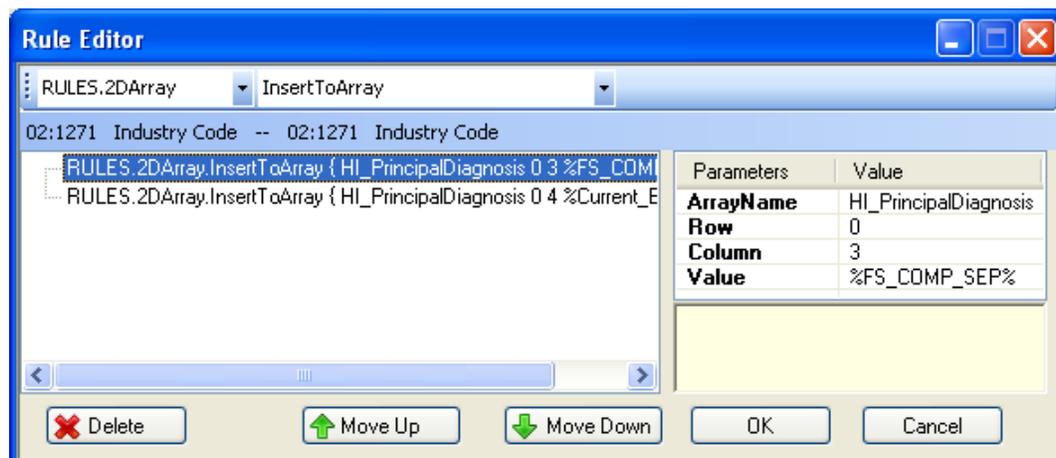
Represents the output file's composite separator.

Format (case sensitive)

```
FS_COMP_SEP
```

Example

This source rule puts the composite sub-element separator into an array when building a segment. The following rule puts the current element's value into the next column of the array.



FS_ELM_SEP

Represents the output file's element separator.

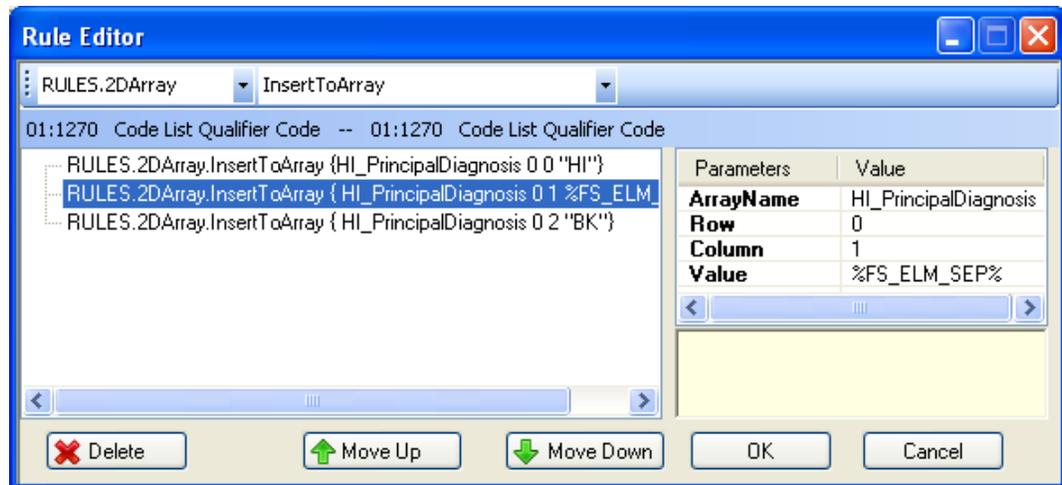
Format (case sensitive)

FS_ELM_SEP

Example

This source rule puts the following into array PrincipalDiagnosis:

- The literal value “HI” into the first row's first column
- The element separator into the first row's second column
- The literal value “BK” into the first row's third column



FS_ELMID

X12 EDI only.

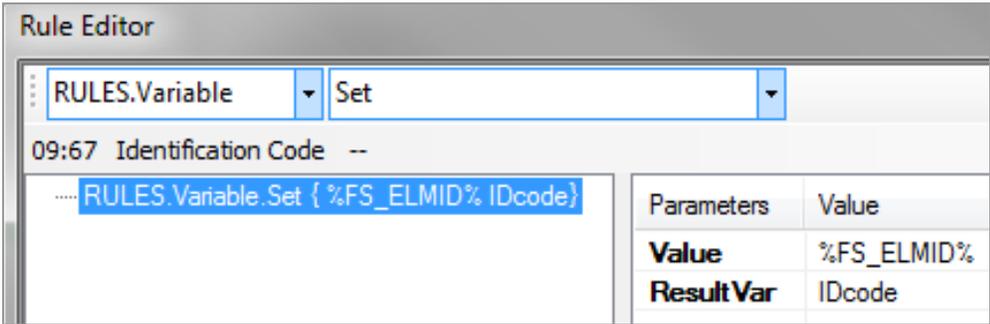
Represents the current target element ID.

Format (case sensitive)

```
FS_ELMID
```

Example

This rule puts the ID of the current target element into variable **IDcode**:



FS_QUOT

Represents a double quote.

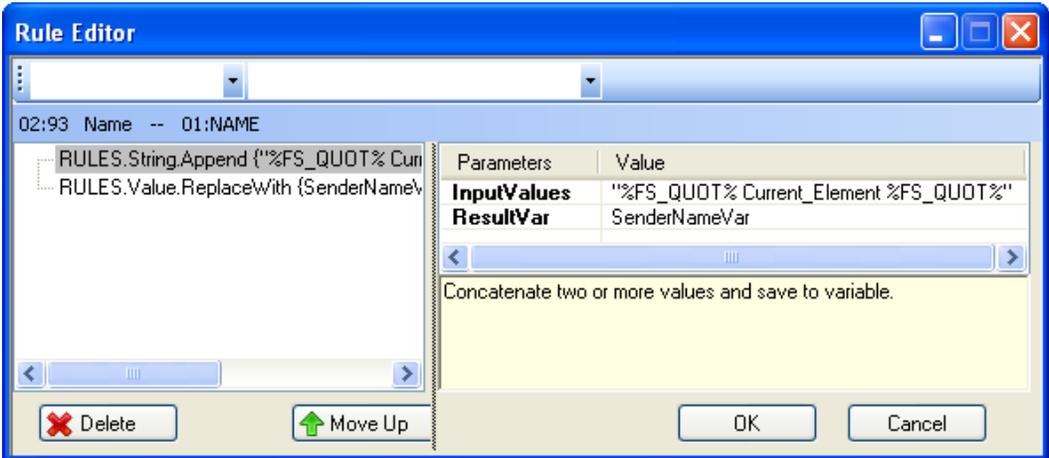
Format (case sensitive)

```
FS_QUOT
```

Example

The first rule puts double quotes before and after the value in the current element and places the result in variable SenderNameVar.

The second rule places the value and double quotes in the output.



FS_REP_SEP

Represents the output file's repetition separator (to separate repeated occurrences of a simple data element).

Format (case sensitive)

FS_REP_SEP

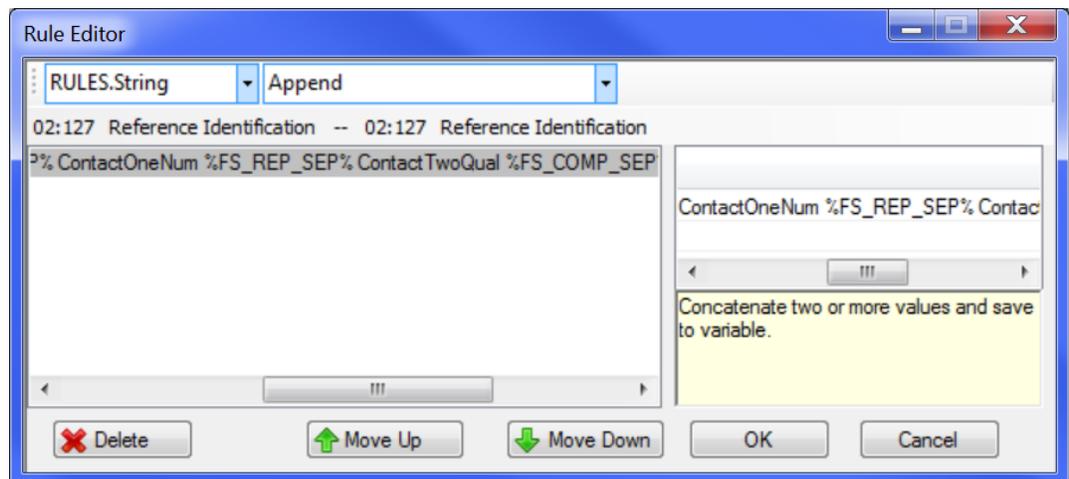
Example

In this String Append rule, the parameter is:

```
"ContactOneQual %FS_COMP_SEP% ContactOneNum %FS_REP_SEP%  
ContactTwoQual %FS_COMP_SEP% ContactTwoNum"
```

It could be used to generate this segment, which contains a repeating composite at the COM03:

```
COM*TE*(925)555-1212*AA:01^BN:(925)555-1200~
```



FS_SPACE

Represents a space.

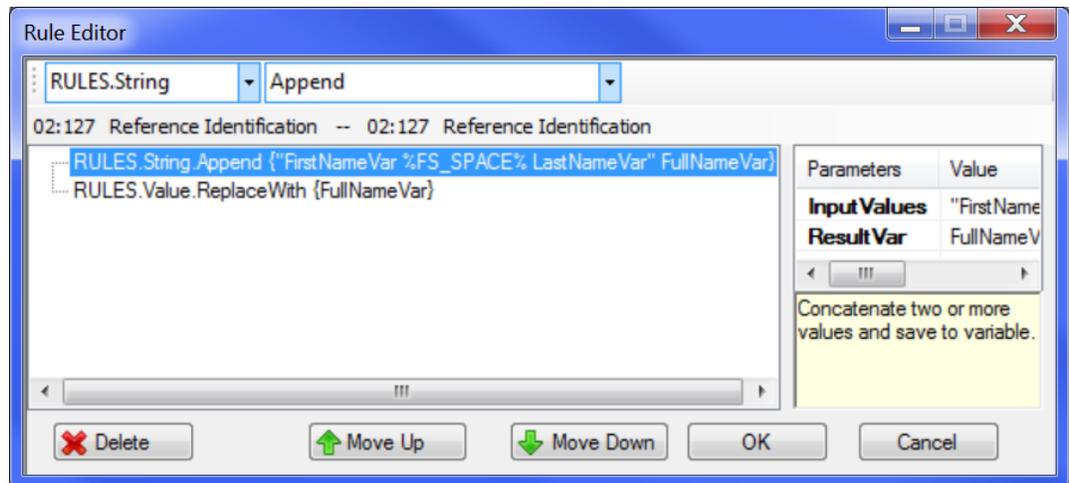
Format (case sensitive)

FS_SPACE

Example

The first rule puts the contents of **FirstNameVar** and **LastNameVar** into variable **FullNameVar**. We use FS_SPACE instead of an actual space, which is a delimiter in the Append rule.

If FirstNameVar contained ALAN and LastNameVar contained ANDERSON, FullNameVar will contain ALAN ANDERSON.



FSID

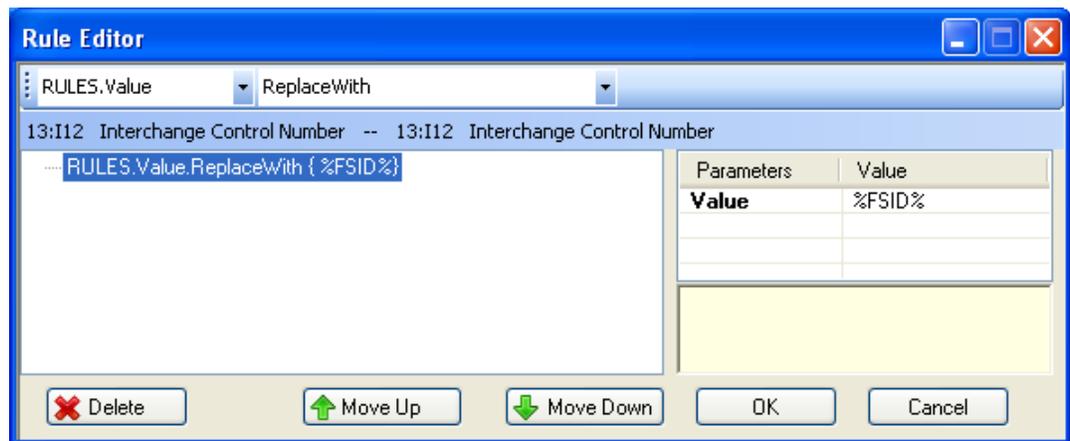
Represents the FSUID – an ID that is guaranteed to never repeat.

Format (case sensitive)

FSID

Example

This rule puts a unique 26-character ID into the current target element or field. If the field is not 26 characters long, the ID is truncated.



A typical value might be KC1VL817RC8TV7KNJCCKEBJ0Q3.

FSLOOPID

X12 EDI only.

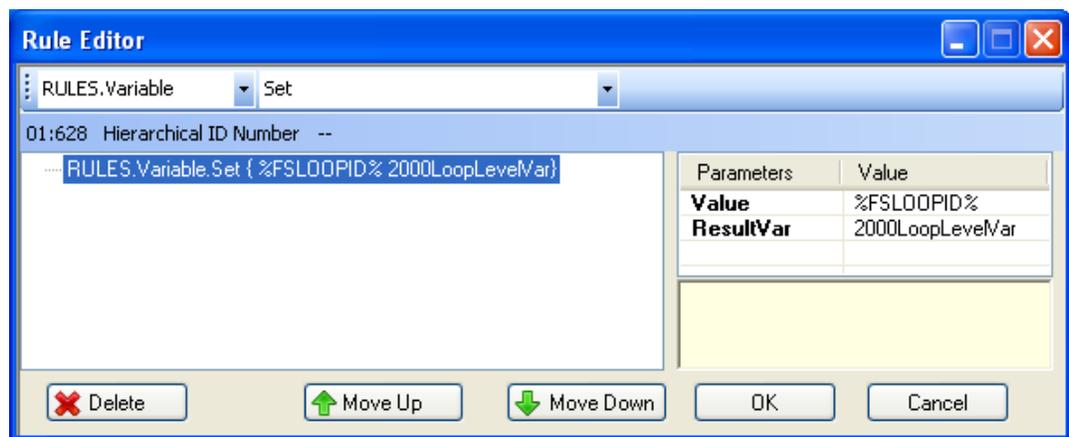
Represents the current target loop ID.

Format (case sensitive)

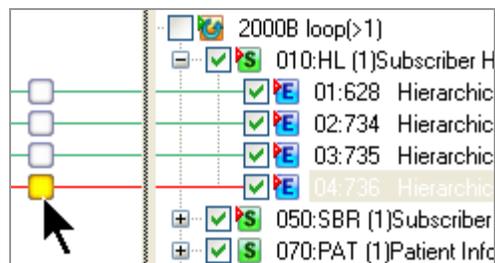
FSLOOPID

Example

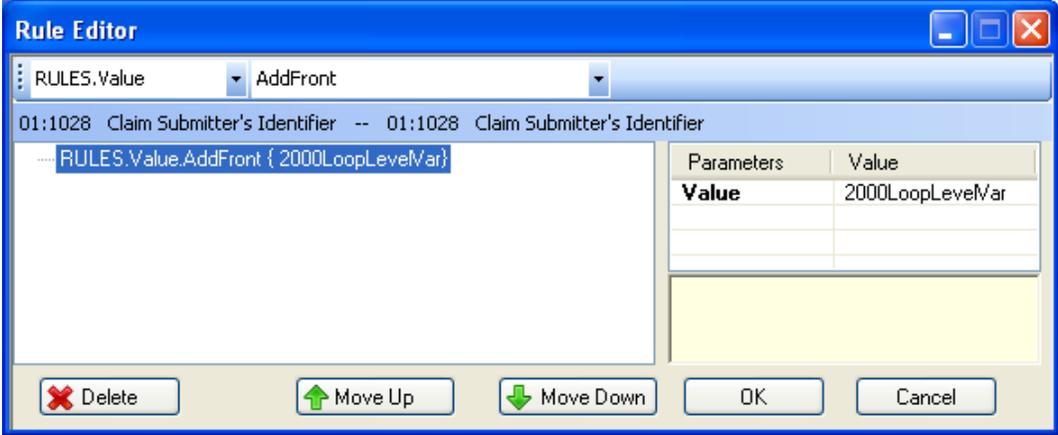
To put the ID of the current target loop into variable **2000LoopLevelVar**, put this rule on an element in the loop:



If you put this rule on a target element in this HL, you should get the value 2000B in the variable **2000LoopLevelVar**:



You can then use that variable to populate a target element or field with 2000B. This rule puts the variable's contents (2000B, for example) in front of the mapped value:



FSLOOPSEQUENCE

X12 EDI only.

Represents the current iteration of the current loop, starting with 1.

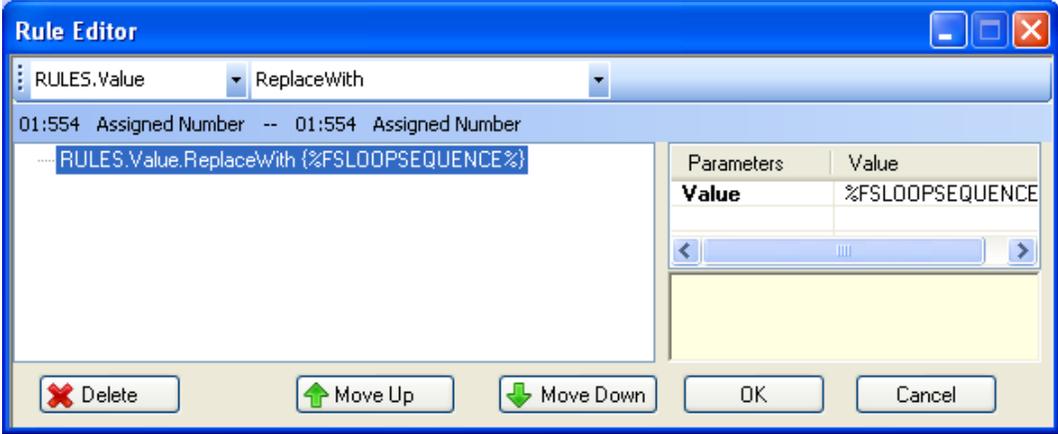
Format (case sensitive)

FSLOOPSEQUENCE

Example

This rule puts the iteration of the current target loop into the current element. The value resets at the end of the current set of multiple loops. For example, if this rule is placed on the LX01, it would automatically reset for each CLM:

```
CLM1      LX - contains 1
           LX - contains 2
CLM2      LX - contains 1
           LX - contains 2
```



If you need to fill out the value to a certain length, use something like this before the ReplaceWith rule:

```
[RULES.Format.RightJustified { 0 %FSSEGSEQUENCE% ""}]
```

FSSEGID

X12 and flat file only.

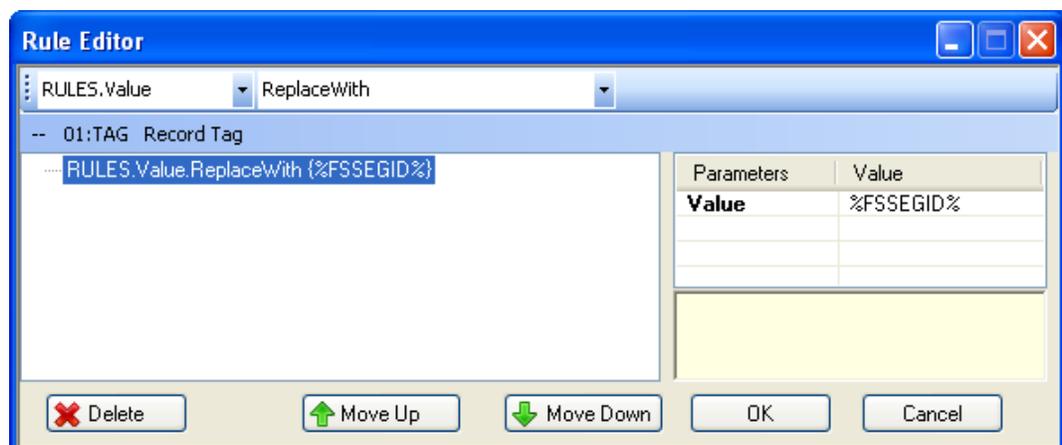
Represents the current target segment's ID.

Format (case sensitive)

FSSEGID

Example

This rule puts the ID of the current target segment into the current element.



FSSEGSEQUENCE

X12 EDI only.

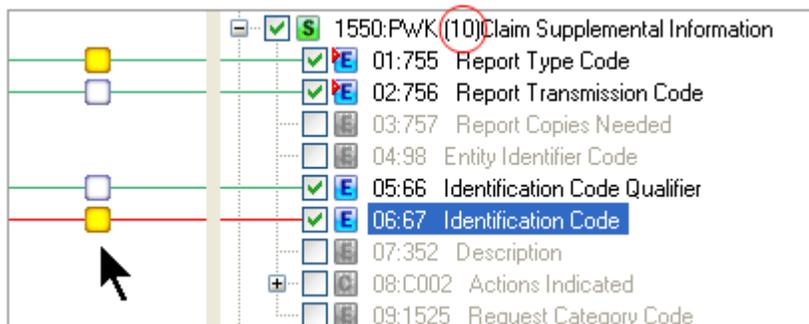
Represents the current iteration of the current target segment, starting with 1.

Format (case sensitive)

FSSEGSEQUENCE

The value resets at the start of the current loop, so that only segments that appear twice at the same location will have a value of more than 1.

This PWK segment has a repeat of 10.



If the output data actually contains 2 PWK segments here, FSSEGSEQUENCE within the PWK segment would result in:

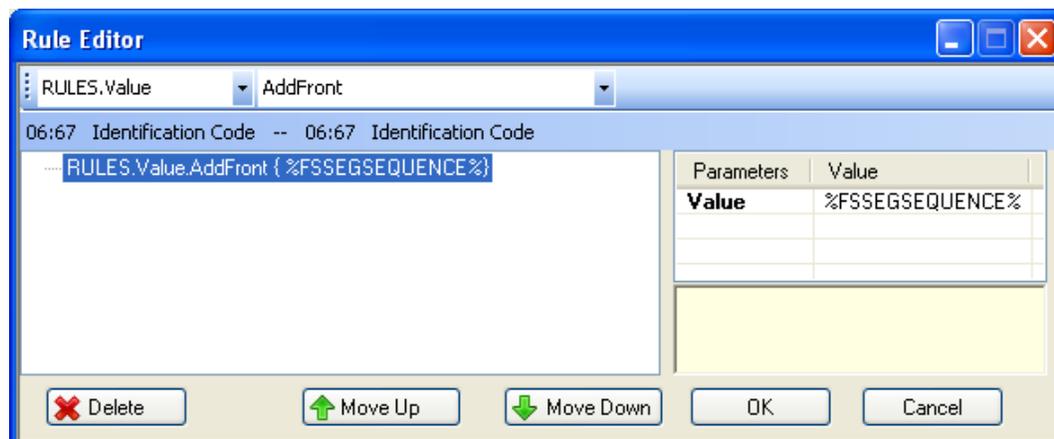
1st PWK FSSEGSEQUENCE = 1

2nd PWK FSSEGSEQUENCE = 2

If there were other iterations of the loops that contain the PWK, FSSEGSEQUENCE would start over with 1.

Example

This rule inserts the iteration of the current target segment at the front of the output value.



RECORDCOUNTER

Not available for XML targets.

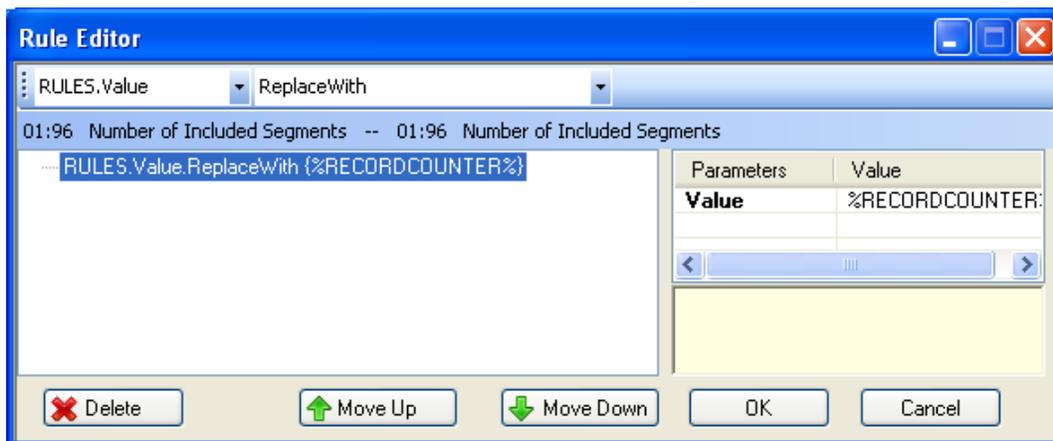
Represents the record's position in the file, starting with 1.

Format (case sensitive)

RECORDCOUNTER

Example 1

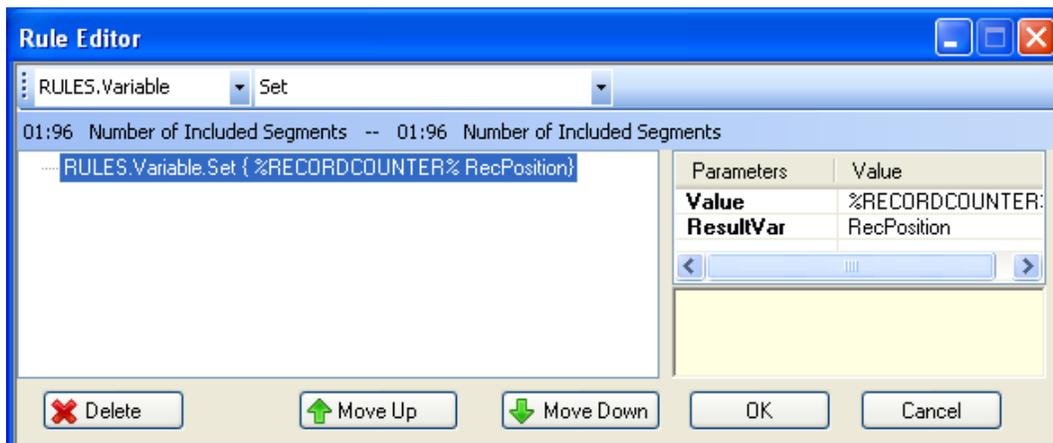
This rule puts the current record's number into the current target element or field.



1st record RECORDCOUNTER= 1
2nd record RECORDCOUNTER= 2
etc.

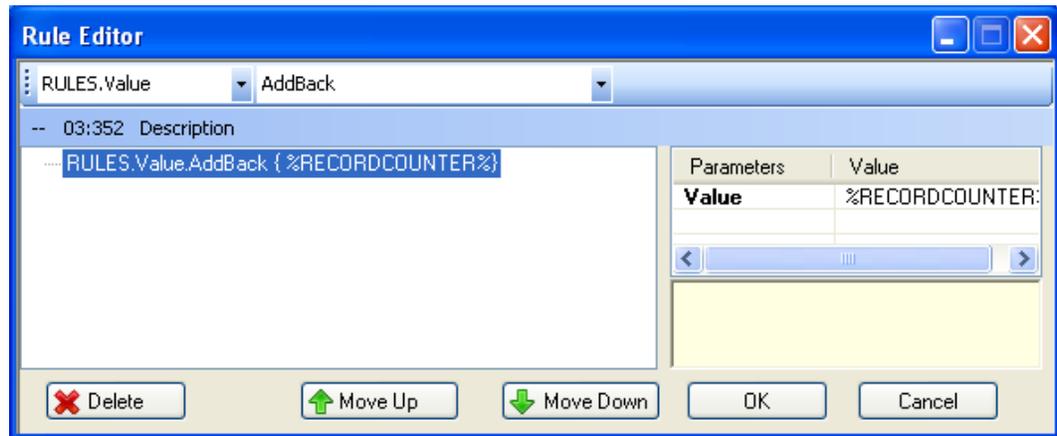
Example 2

This puts the current record's number in variable RecPosition.



Example 3

This appends the record number to the end of the value being output.



Target_Element

Not available for XML targets.

Represents the value in the target element or field.

Format (case sensitive)

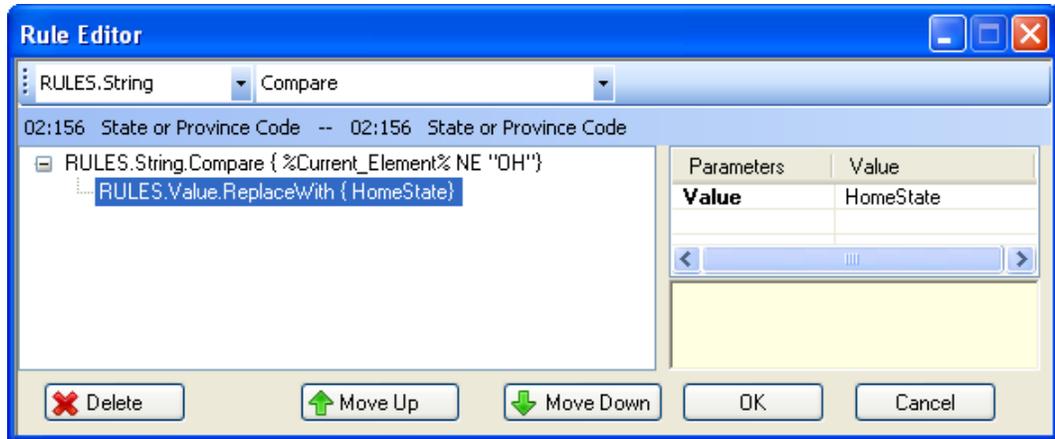
Target_Element

Variables

Using Variables

You can use variables to convey values. This example uses:

- A reserved variable surrounded with % as in %Current_Element%
- A literal surrounded with “ as in “OH”
- A variable with no surrounding punctuation as in HomeState



This rule puts whatever value has been loaded into HomeState into the output.

Populating Variables

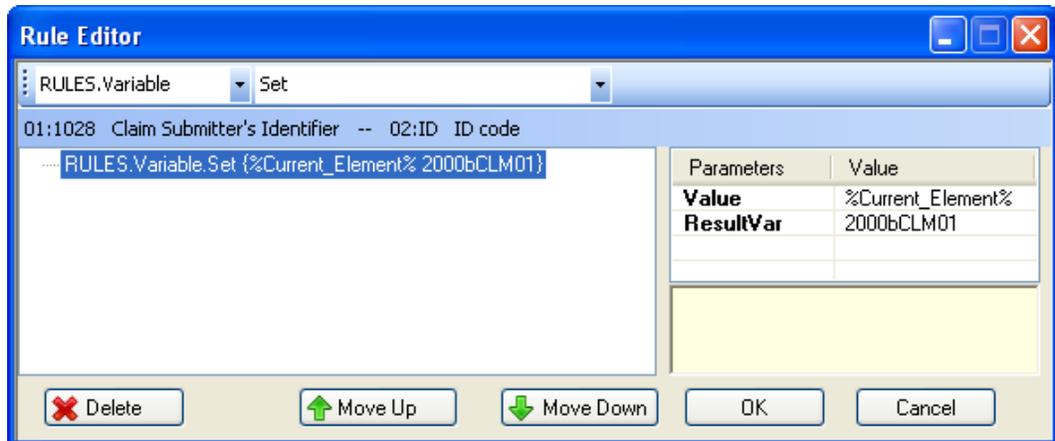
How does a value get into a variable? For example, how would we make HomeState contain "XX"?

You have two options:

- Use another business rule to load the value into HomeState.
- Set up an external file for some of your variables and set the value there. This is a good option for variables that contain specific values that don't originate in the input data, and they may change occasionally. You can simply edit a text file and change the value, rather than having to change the translation map.

Populating Variables with a Business Rule

Typically, this will be a **Variable.Set** rule like this, which puts the current value from the source data into variable 2000bCLM01:



When you want to put a value from the source file into a variable, use a business rule.

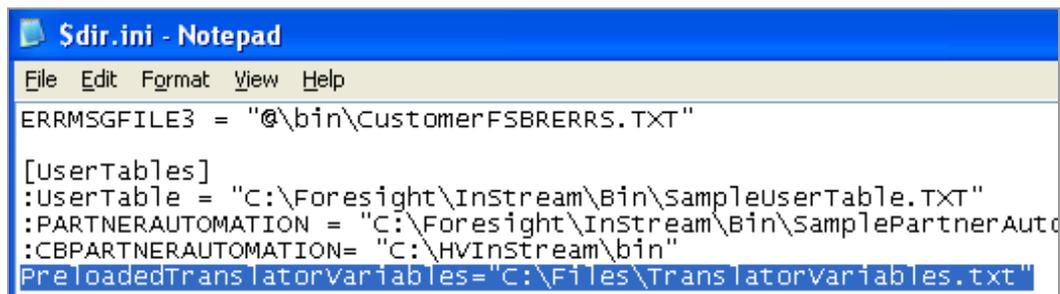
A variable in a business rule will override the same variable in an external text file (see below).

Populating Variables with an External Variables File

You can store variables and their values in an external file. This allows you to change the value without changing the translation map.

This file can have the filename and location of your choice, as long as it can be accessed by Foresight Translator.

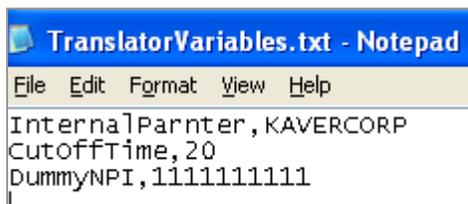
To tell Foresight Translator where to find your translator variables file, add a **PreloadedTranslatorVariables** line to \$dir.ini in Foresight Translator's \Bin directory:



Inside the variable file, each line contains:

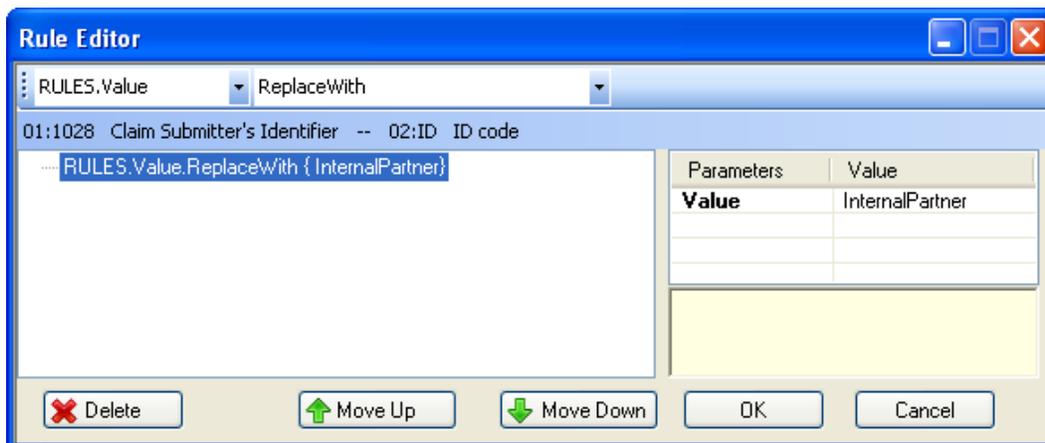
variable,value

Example:



```
InternalPartner, KAVERCORP
CutoffTime, 20
DummyNPI, 1111111111
```

This file contains three variables. **InternalPartner** contains the value “KAVERCORP.” If you use a business rule like this, the output for the element or field where this business rule is located will contain “KAVERCORP.”



InternalPartner must be spelled and capitalized exactly the same in the external file and in the business rule.

Save it with any filename and a location that is accessible to Foresight Translator. This file can contain other data also.

Literal Values

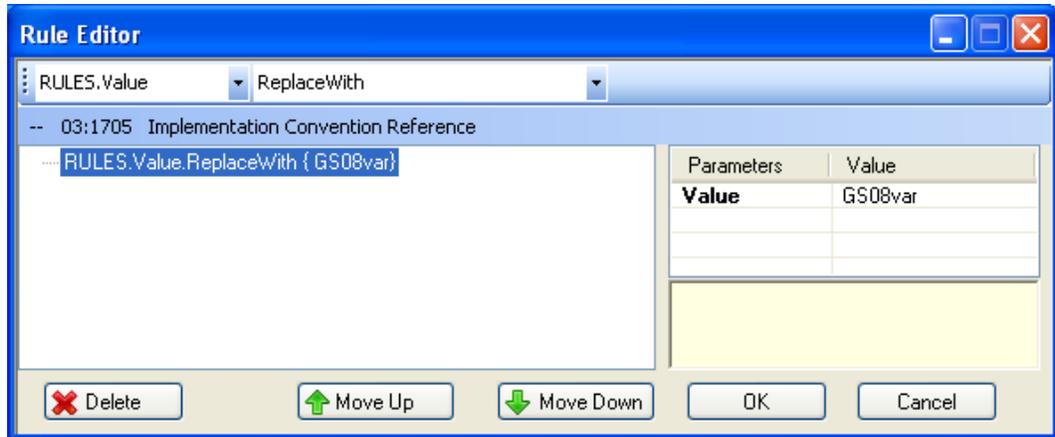
You can use literal values to insert specific values into rules. If there is possible confusion about whether the value is a literal or a variable, surround the literal with double quotes.

Good practice: Always surround literals with double quotes.

Example 1

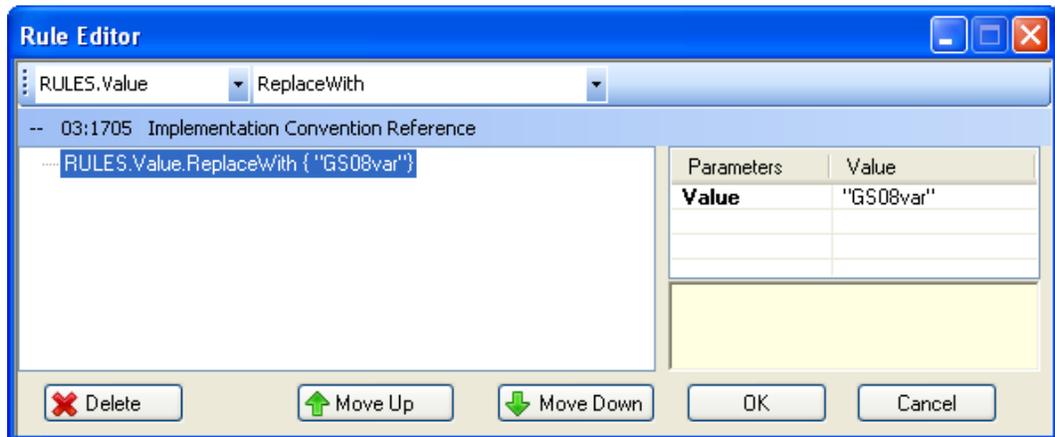
If GS08var exists as a variable, its contents will be used.

Otherwise, the literal value GS08var will be used in the output.



Example 2

Because of the double quotes, the literal value GS08var will be used in the output, regardless of the existence of a variable with the name GS08var.

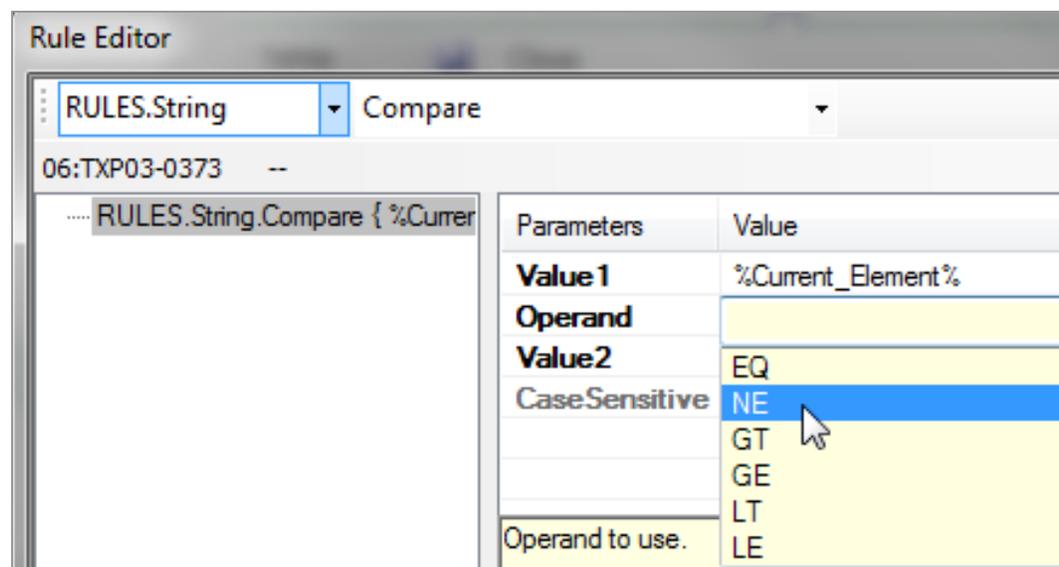


Operands

Common operands in Foresight Translator rules include:

Operand	Meaning
EQ	Equal to (=)
NE	Not equal to (\neq)
GT	Greater than ($>$)
GE	Greater than or equal to (\geq)
LT	Less than ($<$)
LE	Less than or equal to (\leq)

Example



Rules.2DArray

Array rows and columns

Array business rules refer to cells in arrays by row and column. Example: cell 0,1 means row 0 and column 1. The top row in an array is row 0 and the first column is column 0.

Row 0 →	0,0	0,1	0,2
Row 1 →	1,0	1,1	1,2
Row 2 →	2,0	2,1	2,2
	↑	↑	↑
	Column 0	Column 1	Column 2

CreateByStringList

This rule creates an array from a list of values.

CreateByStringList inserts values as columns, as shown here:

	0	1	2	3
0	aa	bb	cc	dd

Format of Parameters

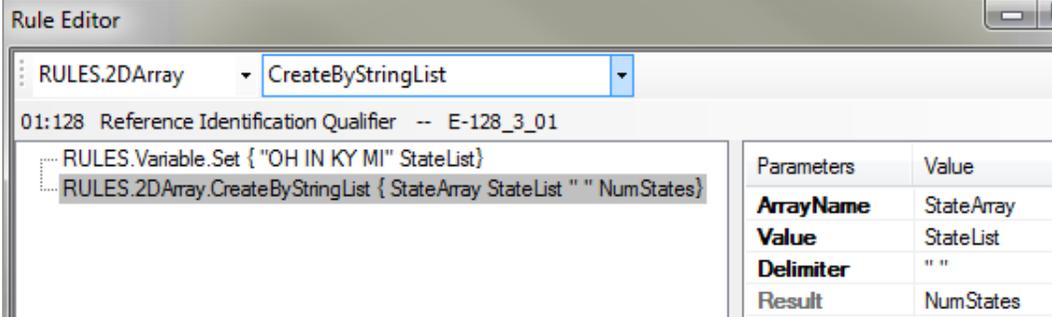
ArrayName Values Delimiter ResultVar RowIndex

Where:

<i>ArrayName</i>	Name to call the new array, which will contain the values. This must be a literal. If the array exists, it is overwritten.
<i>Values</i>	A variable containing a list of literal values to put in the array, separated by a delimiter that is specified in the next parameter.
<i>Delimiter</i>	Delimiter between the values in the input list. This value can be a literal or a variable.
<i>Result</i>	Optional variable to hold the number of columns in the output array.
<i>RowIndex</i>	Populate the array starting with this row. Unless otherwise specified, each value goes in row 0 starting with cell 0,0.

Example

This example creates an array called StateArray and loads the values OH, IN, KY, and MI into cells 0,0 through 0,3. The input list uses a space for a delimiter. The variable NumStates will contain four after this rule runs.



The array will populate as follows:

	0	1	2	3
0	OH	IN	KY	MI

CreateByStringListAsRow

This rule creates an array in rows from a list of values.

CreateByStringListAsRow insert values as rows, as shown here:

```
0
0 aa
1 bb
2 cc
3 dd
```

Format of Parameters

ArrayName Values Delimiter Sub-Delimiter ResultVar

Where:

ArrayName Name to call the new array. This must be a literal. If the array exists, it is overwritten.

Value A variable containing a list of literal values to put in the array, separated by a delimiter that is specified in the next parameter. Each value goes in row 0 starting with cell 0,0.

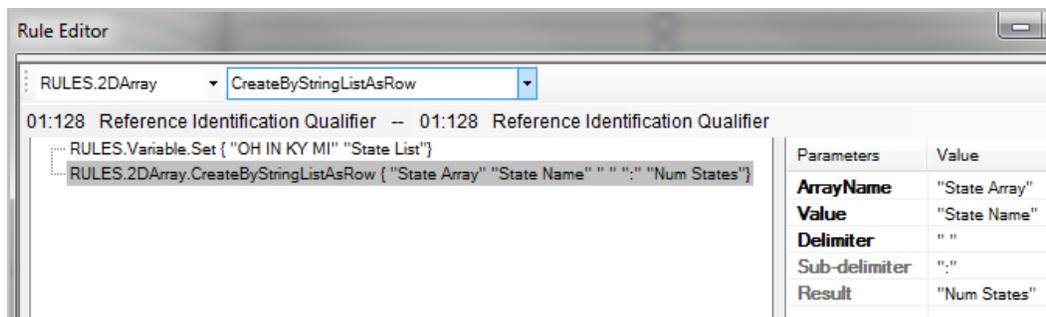
Delimiter Delimiter between the values in the input list.

Sub-Delimiter Sub-delimiter between the values in the input list.

Result Optional variable to hold the number of rows in the output array.

Example

This example creates an array called StateArray and loads the values OH, IN, KY, and MI into cells 0,0 through 3,0. The input list uses a space for a delimiter. The variable NumStates will contain four after this rule runs.



The array will populate as follows:

```
    0
0   OH
1   IN
2   KY
3   MI
```

CreateNewArray

This rule selects rows from an existing array to make new array. A variable shows how many rows were added.

Format of Parameters

ArrayName Rules NewArrayName ResultVar

Where:

ArrayName Name of an existing array that contains the values. This must be a literal.

Rules This is the selection criteria in this format:
Column="value",Column="value" ...
Example: 0="OH"

NewArrayName Name of the new array to be created.

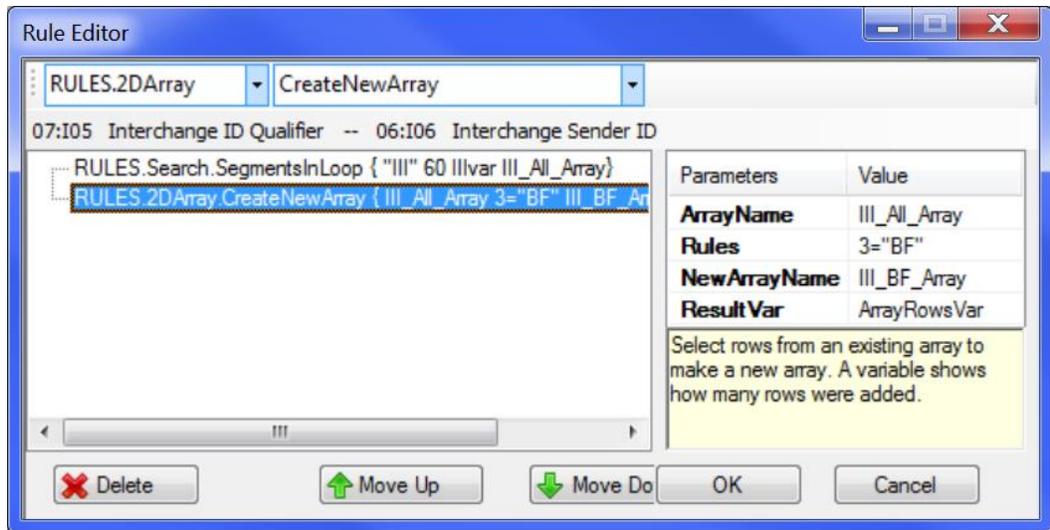
ResultVar Variable to hold the number of rows in the new array.

Example

Assume that the new array should contain only the rows that contain BK in column 3 in existing array III_All_Array.

```
1, 1, III, BK, 486
1, 2, III, BF, 555
1, 3, III, ZZ, 21
1, 4, III, BK, 666
```

This example will create a new array called III_BF_Array that contains the first and last rows from the data in III_All_Array. ArrayRowsVar will contain 2.



Delete

This rule deletes an array.

Format of Parameters

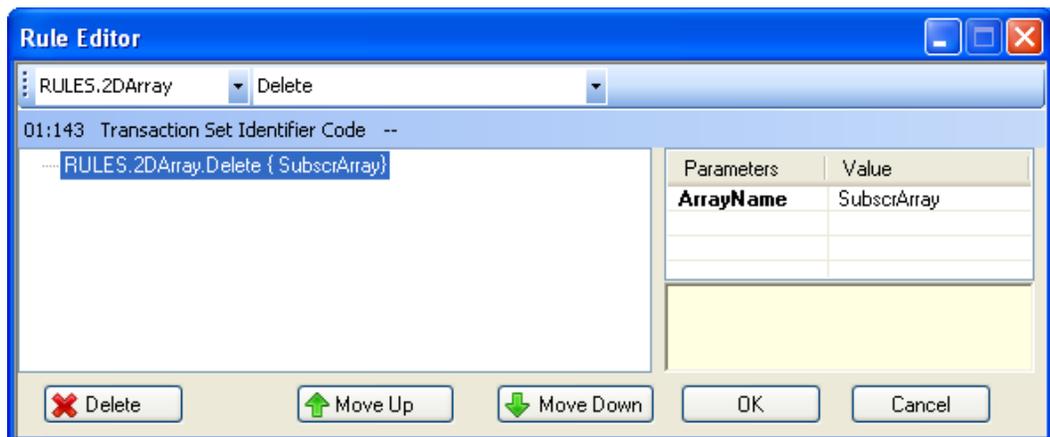
ArrayName

Where:

ArrayName Name of the array to delete. This must be a literal.

Example

This rule deletes the array SubscrArray.



DeleteAllArrays

This rule deletes all arrays.

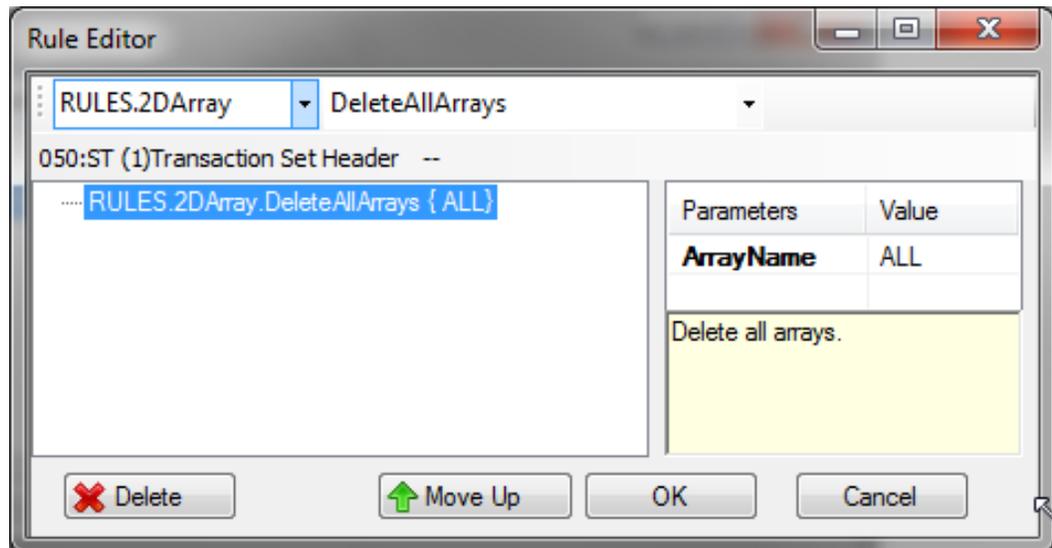
Format of Parameters

ArrayName

Where:

ArrayName This must be set to ALL.

Example



DeleteColumnsFromArray

This rule deletes a column from all rows in an array.

This can be used in any array and is especially handy for arrays created by [SegmentsInLoop](#) (see page 214), where the first two columns are not actual data.

Format of Parameters

ArrayName Column

Where:

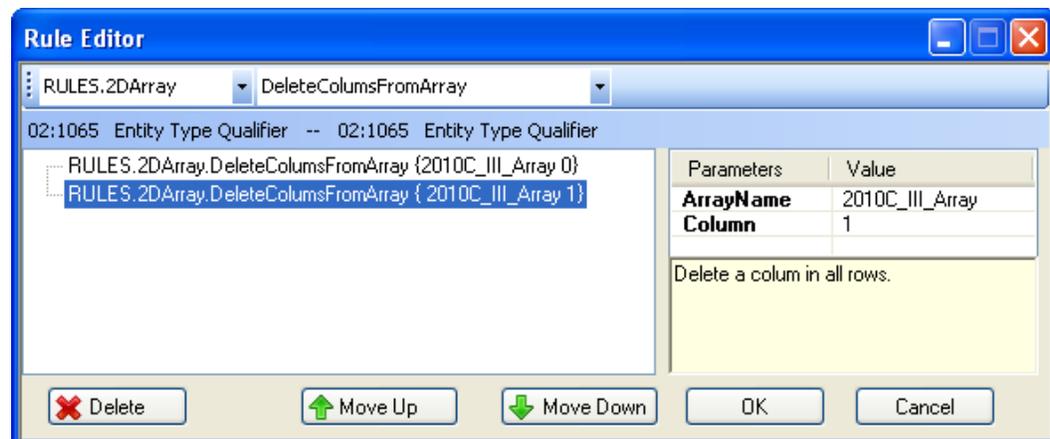
<i>ArrayName</i>	Name of the array that contains the column to delete. This must be a literal.
<i>Column</i>	The column to be deleted. This can be a variable or a literal in double quotes and the value can be a number, C for current column, or N for next column. Columns start with 0.

Example

Assume you do not want the first two columns in 2010C_III_Array, which contains:

```
1, 1, III, BK, 486
1, 2, III, BF, 555
1, 3, III, ZZ, 21
1, 4, III, BF, 666
1, 5, III, ZZ, 333
1, 6, III, BF, 777
1, 7, III, BF, 888
```

Use these rules to delete them:



DumpAllArrays

Important: This rule is for debugging purposes only. Do not use this rule in production maps.

This rule writes the contents of all arrays to the log or to a report generated by Foresight Translator.exe's **-r** command line parameter.

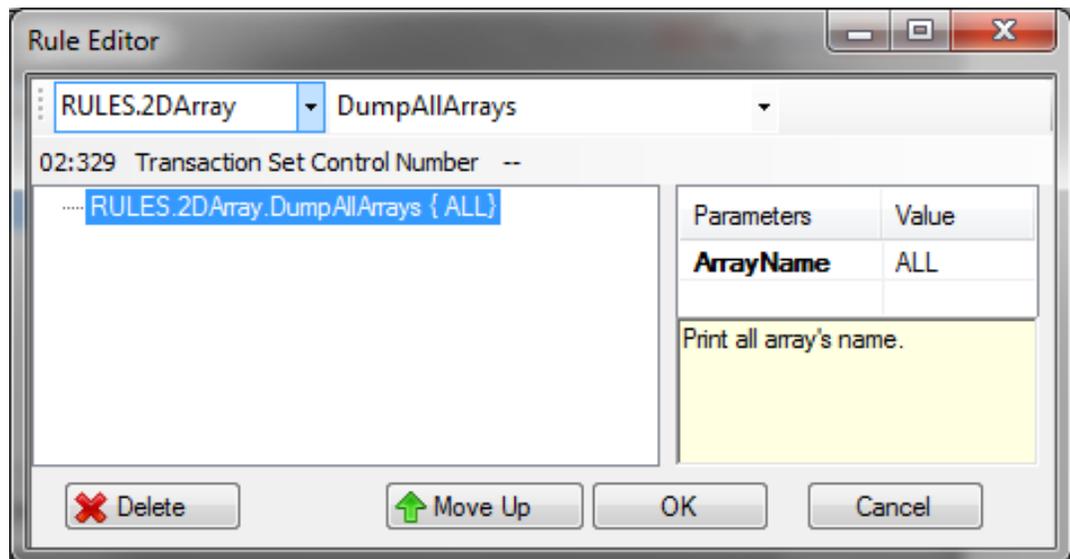
Format of Parameters

ArrayName

Where:

ArrayName Must be ALL.

Example



DumpArray

Important: This rule is for debugging purposes only. Do not use this rule in production maps.

This rule puts all values in an array into the report, which is generated with Foresight Translator's command-line **-r** parameter.

Format of Parameters

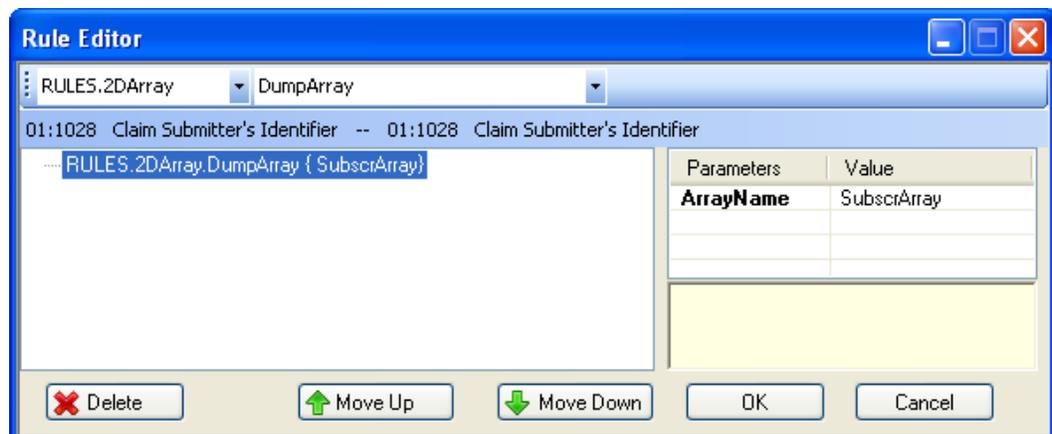
ArrayName

Where:

ArrayName Name of the array containing the data you want to see.
This must be a literal.

Example

This rule places all values in the array SubscrArray into the report.



DumpArrayToCSVFile

Important: This rule is for debugging purposes only. Do not use this rule in production maps.

This rule puts all values in an array into a .CSV file, which is generated with Foresight Translator's command-line `-r` parameter.

Format of Parameters

ArrayName

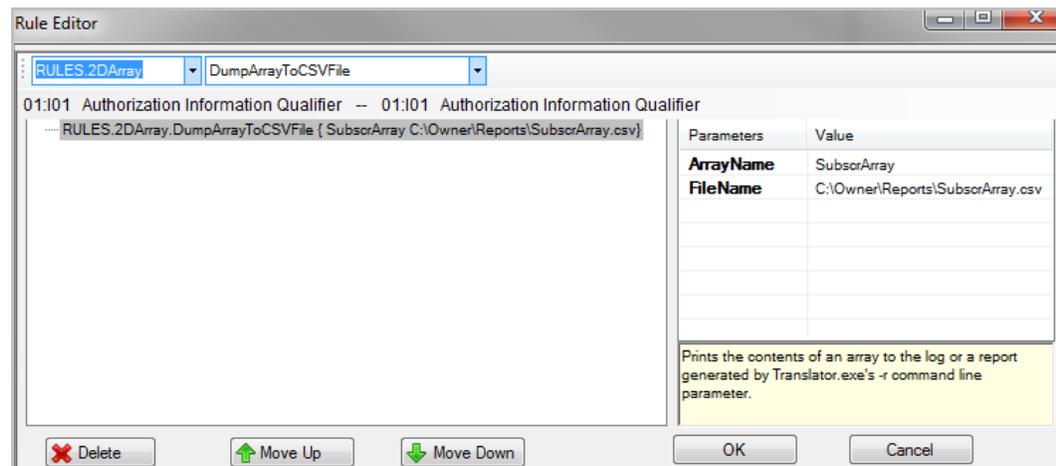
Where:

ArrayName Name of the array containing the data you want to see.
This must be a literal.

FileName Path and name of .CSV file.

Example

This rule places all values in the array `SubscrArray` into the file `C:\Owner\Reports\SubscrArray.csv`.



GetAllColumnValues

This rule populates a variable with all values from an array column.

Format of Parameters

ArrayName Column ResultVar

Where:

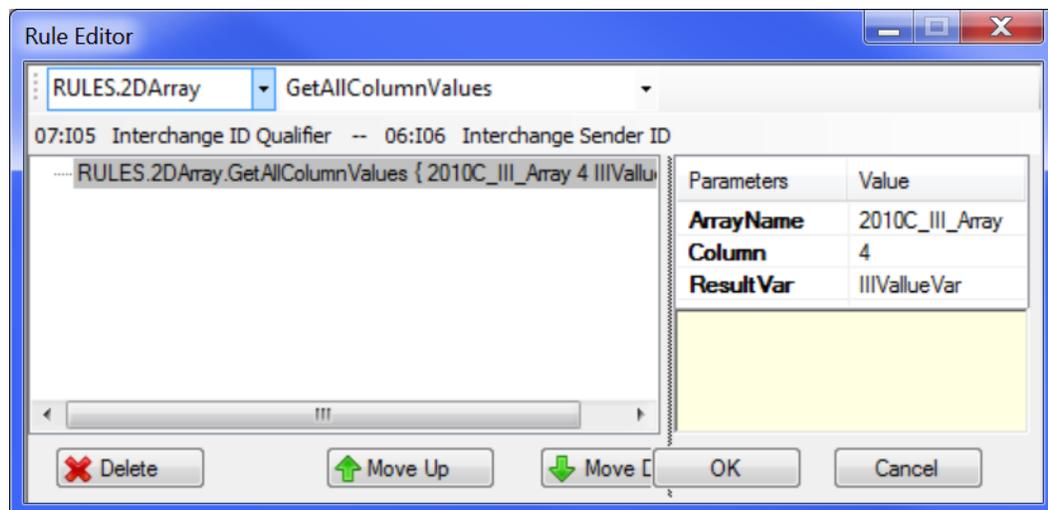
<i>ArrayName</i>	Name of the array that contains the values. This must be a literal.
<i>Column</i>	The column to return. This can be a variable or a literal in double quotes and the value can be a number, C for current column, or N for next column. Indexes start with 0. To specify multiple columns, use GetAllColumnValuesAsArray on page 114.
<i>ResultVar</i>	Variable that is to receive the returned values. This is a list of variables, separated by commas.

Example

Assume you want the values in the last column in 2010C_III_Array, which contains:

```
1, 1, III, BK, 486
1, 2, III, BF, 555
1, 3, III, ZZ, 21
1, 4, III, BF, 666
```

Use this rule:



IIIValueVar will contain: "486", "555", "21", "666"

GetAllColumnValuesAsArray

This rule populates a variable with all values from one or more array columns. It is similar to [GetAllColumnValues](#) on page 113, and contains the same parameters. GetAllColumnValuesAsArray, however, allows multiple columns to be specified, separated by commas.

GetColumnValue

This rule finds the first value in an array that matches specified conditions and returns it in a variable. It also reports the row index in which the value was found.

Format of Parameters

ArrayName Rules Column RowIndex ResultVar

Where:

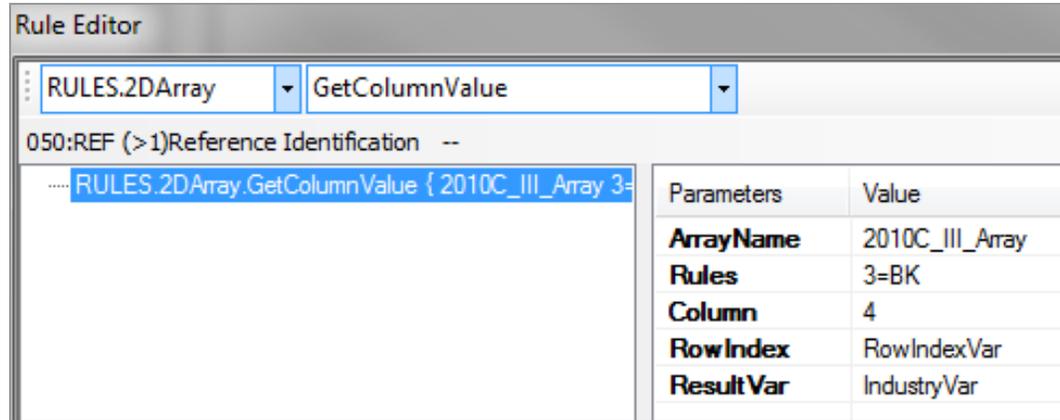
<i>ArrayName</i>	Name of the array to be searched. This must be a literal.
<i>Rules</i>	This is the selection criteria in this format: Column=value,Column=value,... Example: 0=OH, 3=COLUMBUS (Search for a row that contains OH in column 0 and COLUMBUS in column 3.)
<i>Column</i>	The column identifier to return if the selection criteria finds a match in the array. This can be a variable or a literal in double quotes and the value can be a number, C for current column, or N for next column. Example: 4 (Building on the example given in <i>Rules</i> : Search for a row that contains OH in column 0 and COLUMBUS in column 3. When you find it, return the value found in column 4 of that row.)
<i>RowIndex</i>	The row index (or row identifier) where matching data was found in the array. This can be a variable or a literal in double quotes.
<i>ResultVar</i>	Output variable to hold the returned value found in the column and row.

Example

Assume you want to search for BK column 3 in this array. If it is found, you want to return the value in column 4. The array contains:

```
1, 1, III, BK, 486
1, 2, III, BF, 555
1, 3, III, ZZ, 21
1, 4, III, BK, 666
```

Use this rule.



RowIndexVar will contain 0 (the first row) and **IndustryVar** will contain 486. The rule stops searching the array after it finds the first match. The last row, which also contains BK, has no effect on the output.

GetRowCounter

This rule populates a variable with the number of rows in the specified array.

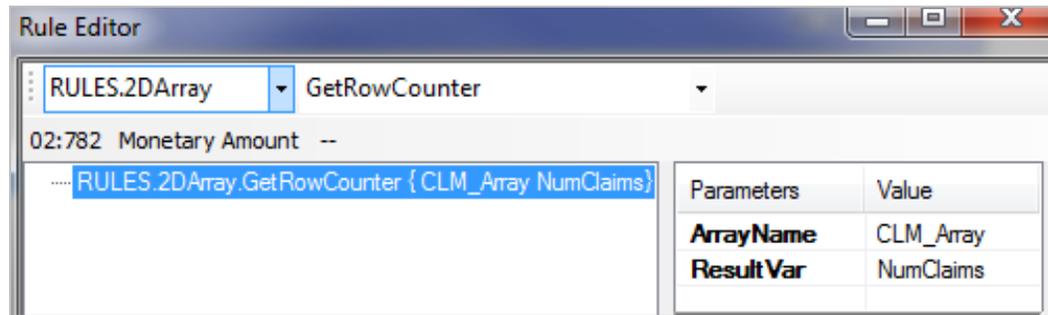
ArrayName ResultVar

Where:

ArrayName Name of the array. This must be a literal, optionally surrounded with double quotes.

ResultVar Variable to hold the number of rows.

Example



GetSubElementValue

This rule populates a variable from a component element in an array.

Format of Parameters

ArrayName Row Column SubElmIndex ResultVar

Where:

ArrayName Name of the array that contains the value. This must be a literal, optionally surrounded with double quotes.

Row The row where the data is located. This can be a variable or a literal in double quotes and the value can be a number, C for current row, or N for next row. Row numbers start with 0.

Column The column containing the subelement that is to be returned. This can be a variable or a literal in double quotes and the value can be a number, C for current column, or N for next column. Column indices start with 0.

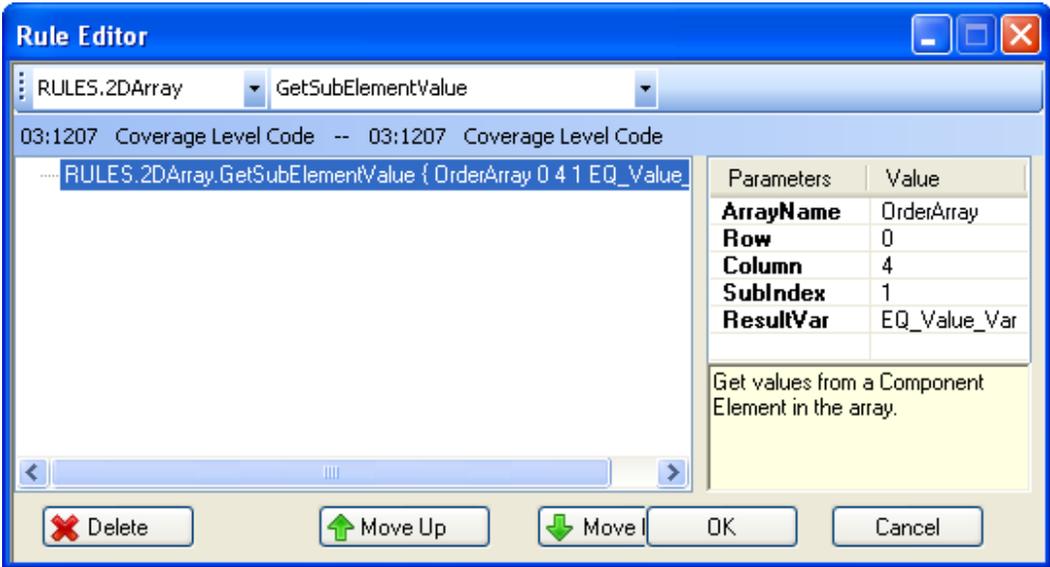
SubElmIndex Index of component element. Indices start with 0.

ResultVar Variable that is to receive the value that was found in the subelement. This can be a literal (in double quotes), or a variable containing the name.

Example

Assume that the following values are in array OrderArray. We want to get the second component element value from column 3 in row 1 and put it in variable EQ_Value_Var.

1	1	AAA	OH	ER:A0170:A1:AA:A2:A1:DOORKNOBS	330.00
2	1	BBB	MI	ER:A0177:A3:AA:A3:A3:DOORBELLS	330.00



GetValueFromArray

This rule populates a variable from a cell in an array.

Format of Parameters

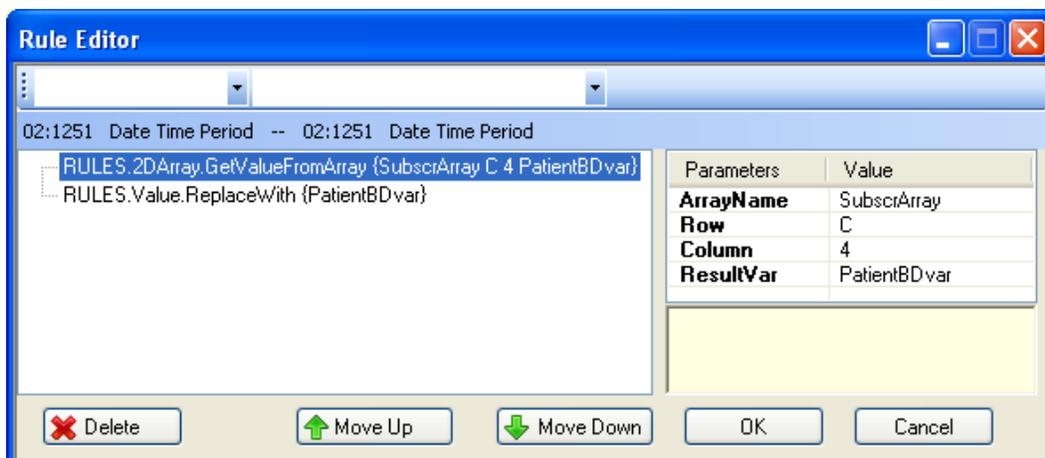
ArrayName Row Column ResultVar

Where:

<i>ArrayName</i>	Name of the array that contains the value. This must be a literal.
<i>Row</i>	The row where the cell is located. This can be a variable or a literal in double quotes and the value can be a number, C for current row, or N for next row.
<i>Column</i>	The column where the cell is located. This can be a variable or a literal in double quotes and the value can be a number, C for current column, or N for next column.
<i>ResultVar</i>	Variable that is to receive the value that was found in the array cell. This can be a variable containing a variable or a literal (in double quotes) containing a variable name.

Example

This rule gets the value from column 4 of the current row of SubscrArray and places it in a variable called PatientBDvar. It then uses that value in the output.



HasValue

This rule returns a row index that contains the values for which you are searching. It requires a sub-rule to act if this information is found.

Format of Parameters

ArrayName Rules ResultVar

Where:

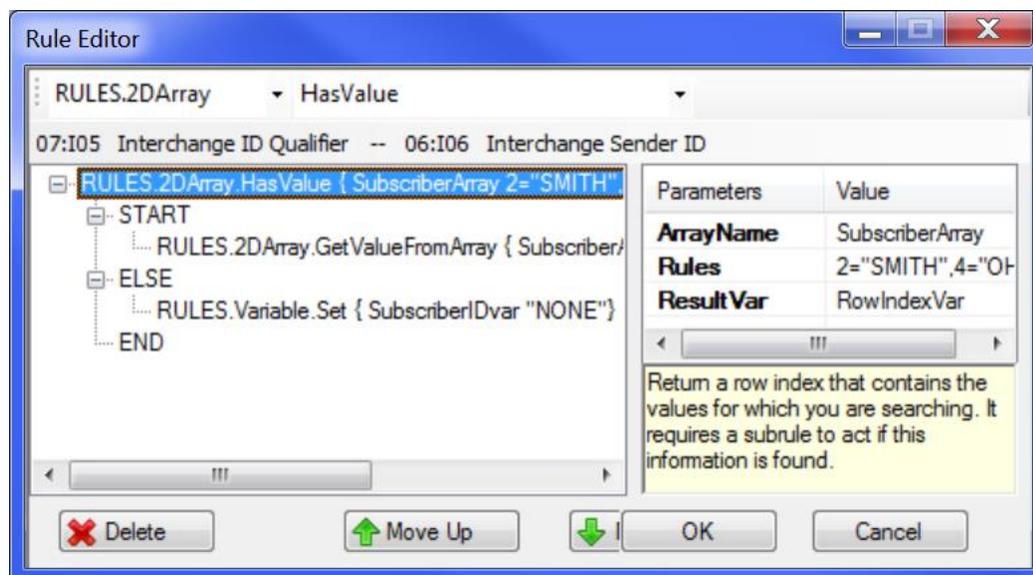
<i>ArrayName</i>	The array to be searched.
<i>Rules</i>	The column number (a literal), an equal sign, and a value. You can have multiple values, each separated with a comma.
<i>ResultVar</i>	A variable containing the row number, if the values were found.

Example

This rule checks to see if a row in SubscriberArray has value SMITH in column 2 and OH in column 4.

- If true, the GetValueFromArray rule executes.
- If false, SubscriberIDvar is set to "NONE".

At least one element in the segment is mapped so that the segment will be in the output.



Please see [Conditional Rules](#) on page 274 for directions on how to set up conditional rules like this one.

InsertARow

This rule populates a row in an array.

Format of Parameters

ArrayName Row Value

Where:

ArrayName Name of the array you are populating. This must be a literal. If it doesn't exist, the array is created.

Row The row where the cell is located. This can be a variable or a literal in double quotes and the value can be a number, C for current row, or N for next row.

Any data currently in the row is deleted and replaced with the new data until the last specified column is reached. After that, data that was in the cells will remain.

Example:

Original row 1, 2, 3, 4, 5

Rule specifies *A**C*D

Row after rule executes: A, , C, D, 5

Value Source of value used to populate the cells. These can be variables, "literals" in quotes, or reserved variables like Current_Element.

The first item must be the delimiter you are using to separate the values in the rule (this is not output).

Example

This inserts the next row into REFarray.

Parameters	Value
ArrayName	REFarray
Row	N
Value	*REF01var*Current_Element*

The value `*REF01var*Current_Element*REFcountVar**"END"` says that the separator is an asterisk and the contents of the row should be:

Column 0 = contents of variable REF01var

Column 1 = contents of the current element

Column 2 = contents of variable REFcountVar

Column 3 is empty (if data was in this cell from a previous rule, it is erased)

Column 4 = the literal END

The resulting row might contain: FH, 123456789, 3,, END

InsertToArray

This rule populates a cell in an array.

Note: InsertToArray and InsertToArryF can be used in the same translation.

Format of Parameters

ArrayName Row Column Value

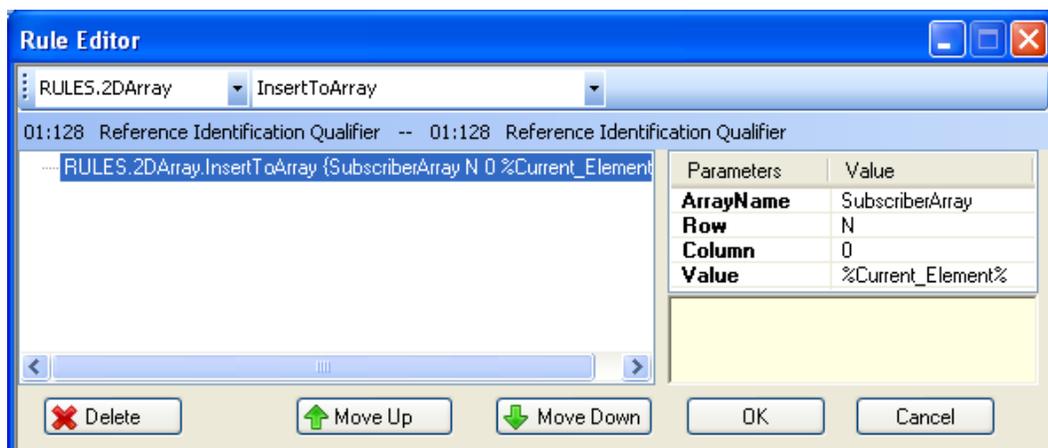
Where:

<i>ArrayName</i>	Name of the array you are populating. This must be a literal. If it doesn't exist, the array is created.
<i>Row</i>	The row where the cell is located. This can be a variable or a literal in double quotes and the value can be a number, C for current row, or N for next row.
<i>Column</i>	The column where the cell is located. This can be a variable or a literal in double quotes and the value can be a number, C for current column, or N for next column.
<i>Value</i>	Source of value used to populate the cell. This can be a variable, a "literal" in quotes, or a reserved variable like Current_Element.

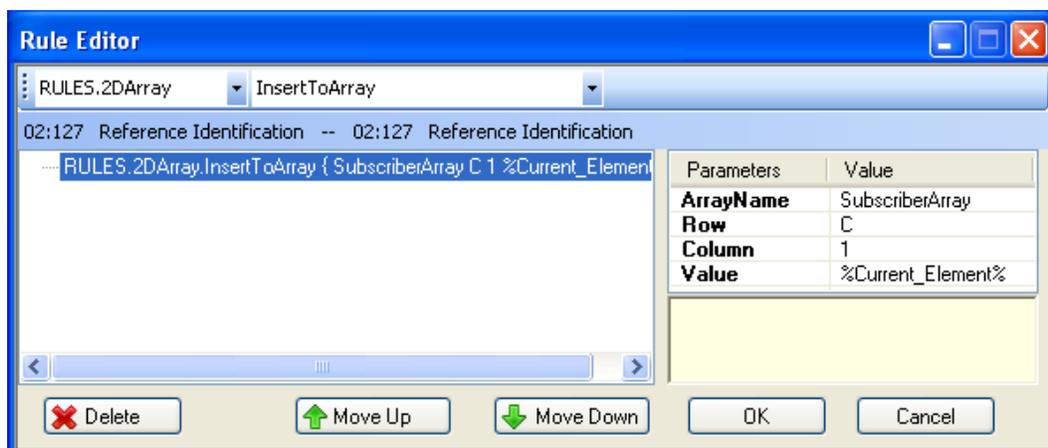
Example 1

This rule:

1. Moves the row pointer down one row (N) in array SubscrArray.
2. Inserts the current element's value into column 0 (the first column)



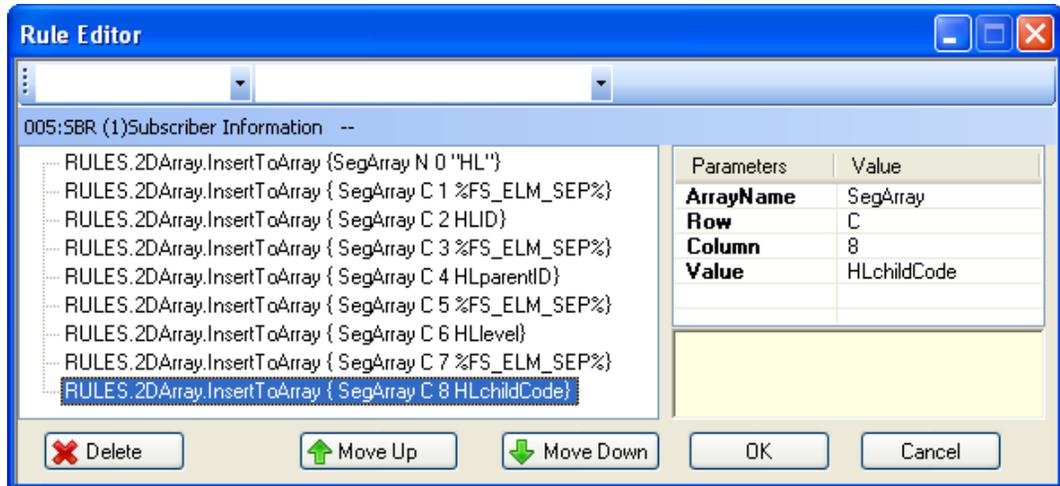
This rule, on another element, adds a corresponding value in the next column. Notice the column is now “1.” Because the Row parameter contains C (for current), the row remains the same.



Example 2

This array assembles HL segments into rows in an array. Each time it executes, it:

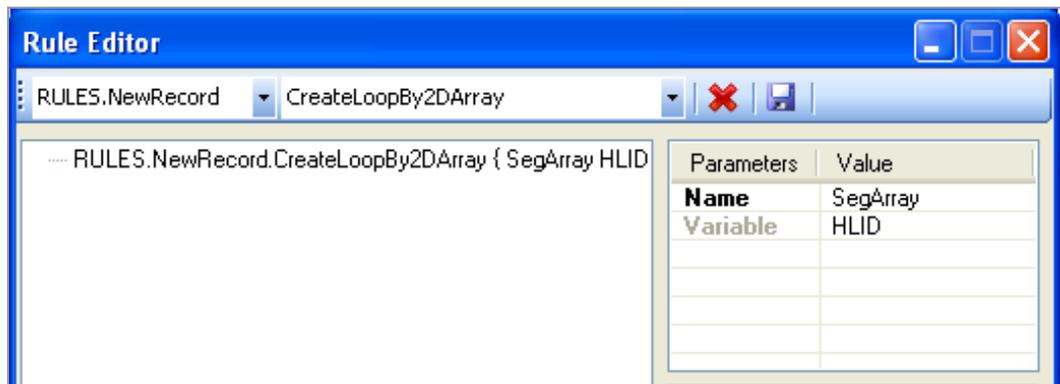
- Goes to the next row (N) and puts the literal HL into the first column
- Goes to column 2 and inserts the value in the variable FS_ELM_SEP, which is the output file's element separator
- Goes to column 3 and puts in the value in the variable HLID into the third column.
- HLID might be incremented in another rule such as Counter.Increment
- Continues putting variables, which have all been predefined to contain values from the HL, into the row, separated by element separators
- You do not need to add the segment terminator



You can accomplish the same result by using InsertToArray rules on each HL element and using Current_Element rather than a variable.

You can then output the records in the array at the location of your choice with a NewRecord.CreateLoopBy2DArray rule like this.

It increments the value in variable HLID after creating each record.



InsertToArrayF

This rule populates a cell in an array and formats its value to a fixed length with padding and justification.

Note: InsertToArray and InsertToArrayF can be used in the same translation.

Format of Parameters

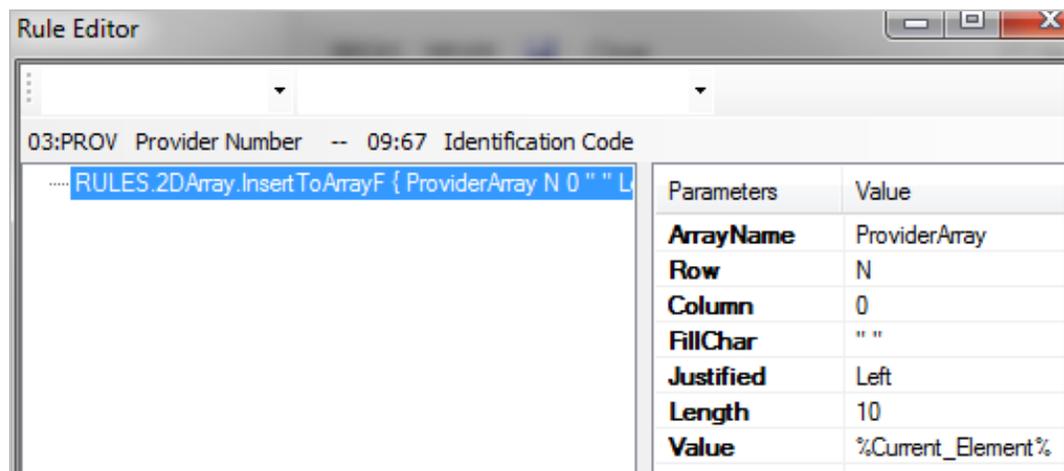
ArrayName Row Column Value FillChar Justified Length

Where:

<i>ArrayName</i>	Name of the array you are populating. This must be a literal with or without double quotes surrounding it. If it doesn't exist, the array is created.
<i>Row</i>	The row where the array cell is located. This can be a variable or a literal in double quotes and the value can be an integer, C for current row, or N for next row. Row indexes start with 0.
<i>Column</i>	The column where the array cell is located. This can be a variable or a literal in double quotes and the value can be a number, C for current column, or N for next column. Column indexes start with 0.
<i>FillChar</i>	Fill character to use if needed to meet the minimum length.
<i>Justified</i>	Left =value is left-justified with padding at end. Right =value is right-justified with padding in front.
<i>Length</i>	Value length after padding is added.
<i>Value</i>	Source of value used to populate the cell. This can be a variable, a "literal" in quotes, or a reserved variable like <code>Current_Element</code> .

Example

This rule takes the value in the current element, pads it with two trailing spaces, and puts it in the first column of the next row of `ProviderArray`.



Loop

This rule repeatedly executes a list of rules until the end of the row is reached. This rule requires one or more sub-rules.

Format of Parameters

ArrayName StartPos Index

Where:

ArrayName Name of the array you are searching. This must be a literal with or without double quotes surrounding it. If it doesn't exist, the array is created.

StartPos Start index. Default is 0.

Index Variable to hold row index (literal with/without double quotes)

MoveToNextRow

This rule moves the row pointer to the next row in the specified array.

Format of Parameters

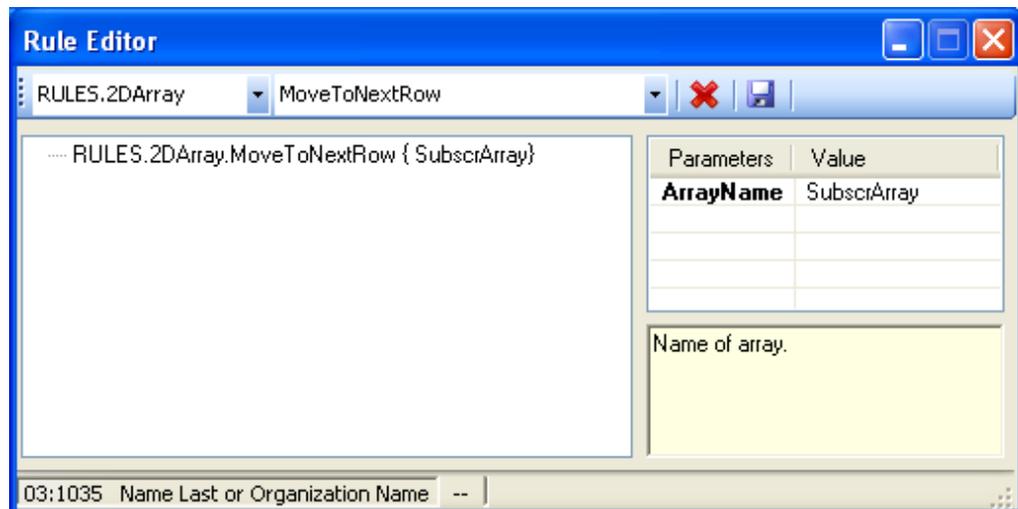
ArrayName

Where:

ArrayName Name of the array. This must be a literal.

Example

This rule moves the row pointer to the next row in the array SubscrArray.



See also [CreateNewArray](#), which lets you move to the next row while inserting values in an array.

OutputMemoryLayout

Future rule. Do not use.

Rules.Counter

Increment

This rule sets up a counter which increments by a specified interval. This rule can be placed on an element/field or a segment/record.

Format of Parameters

CounterName Increment

Where:

CounterName Select RECORDCOUNTER or USERDEFINEDVAR, where USERDEFINEDVAR is a variable to hold the count. If it doesn't exist, the counter is created and contains the value in Increment.

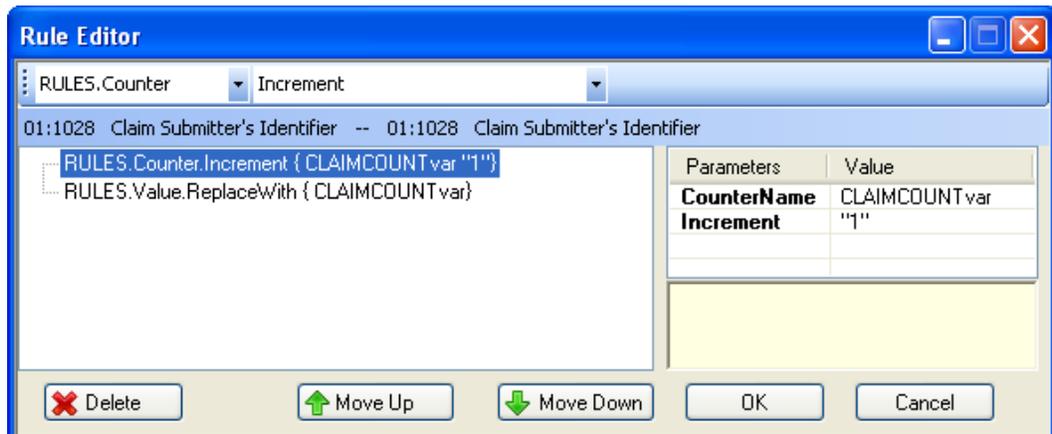
Increment Increment the current contents of CounterName by this amount.

Example

This example puts a count into variable CLMCOUNTvar. Each time the rule executes, it increases the count by 1.

The second rule uses the count as the value in the current element.

Earlier in the file, we would have used Counter.Reset to initialize CLMCOUNT.



Reset

This rule lets you set a counter to an integer that you specify. It can go on an element/field or a segment/record.

Format of Parameters

CounterName Value

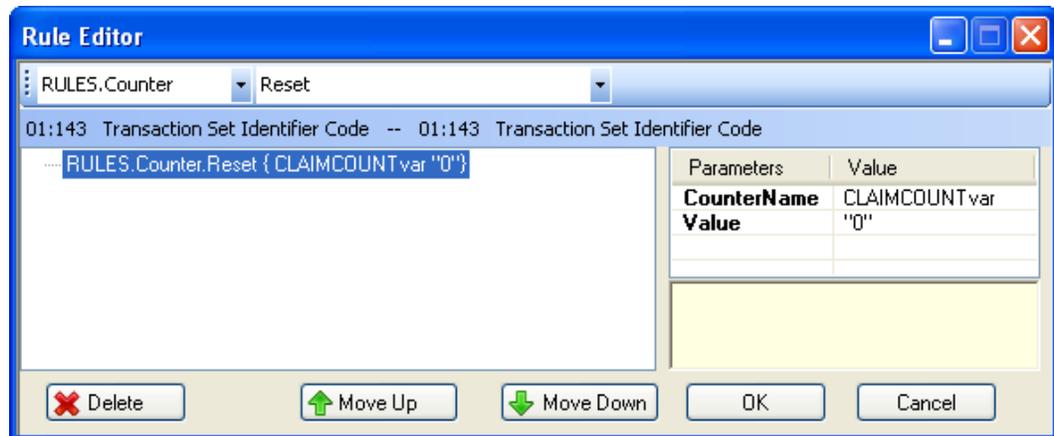
Where:

CounterName Select RECORDCOUNTER or USERDEFINEDVAR, where USERDEFINEDVAR is a variable set to hold the count.

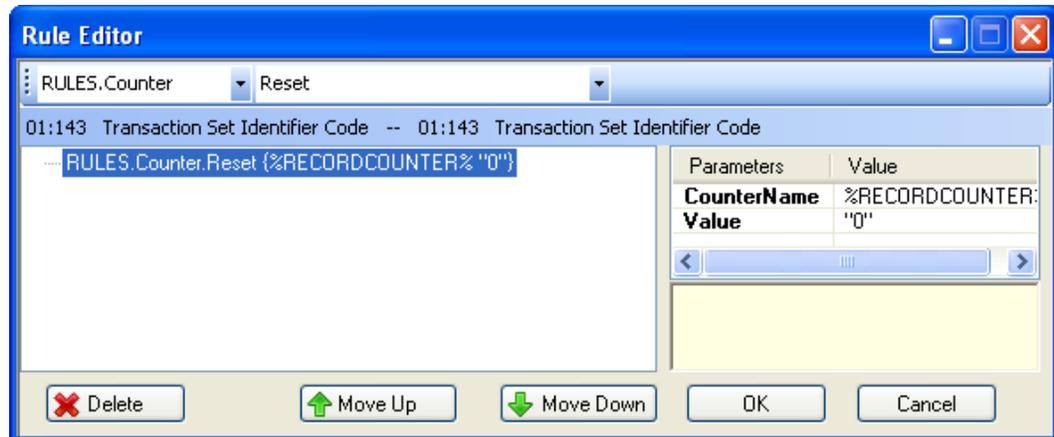
Value Set the counter to this integer. This can be a literal value or a variable.

Examples

This example resets **CLMCOUNTvar** to 0.



This rule on the ST01 initializes the [RECORDCOUNTER](#) to 0. See page 96.

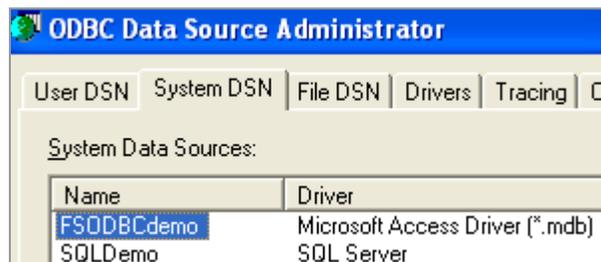


Rules.Database

Before using Database Rules

Before using Foresight Translator Database rules:

- Be sure your database has a DSN name in Control Panel | Administrative Tools | Data Sources (ODBC) | System DSN:



- Edit \$Dir.ini in Foresight Translator's \Bin directory and give the DSN a name:

```
[Database]
DBRef="DSN=SQLDemo"
DBRef1="DRIVER={SQL
Server};SERVER=(FCSUPP10\SQLEXPRESS);DATABASE=TTIDemo;UID=sa;PWD=zxasqw12;"
ProvDB="DSN=FSODBCdemo"
```

RunQuery

This rule lets you query any database that is accessible via ODBC and receive the results in one or more variables. Please read [Before using Database Rules](#) on page 129 before starting. This rule is for Windows platforms only.

For databases that are not accessible via ODBC, use the ISIServer to make the connection instead. See [ISIServerDB](#).

Format of Parameters

DBRef SQL ResultVars

Where:

DBRef Pointer to the database's DSN name, defined in Foresight Translator's \$Dir.ini (see above).

SQL The SQL statement surrounded by double quotes.

ResultVars Variables to hold returned values in the format variable=n, where n is the position of the value that is to be returned.

Example results of query

PartnerID	Name	Description	IsInternal	CreateDate
2	HORIZONE		1	2009-12-02 14:24:05.920

If the Parameters were these:

Parameters	Value
DBRef	ProvDB
SQL	"Select * from Master WHERE (Name='%Current_Element%')"
ResultVars	"ReturnNameVar=2 CreateDateVar=5"

Then:

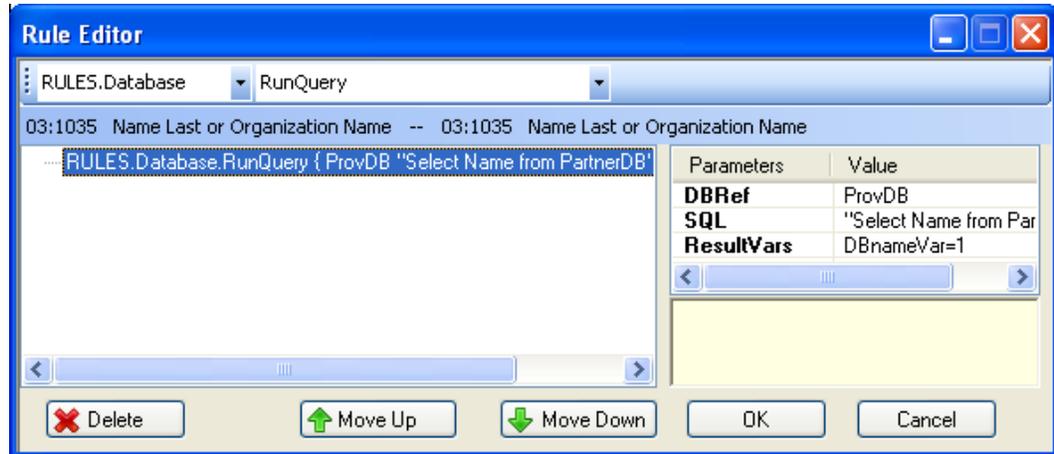
- ReturnNameVar would contain the 2nd value - HORIZONE
- CreateDateVar would contain the 5th value - 2009-12-02 14:24:05.920

Demo: Please see [_readme_ODBC.txt](#) in Translator's \DemoData\ODBC directory.

Example 1

This example shows how to return one value. It queries a database whose DSN name is contained in **ProvDB**, like the example in [Before using Database Rules](#) on page 129.

It executes a SQL query that returns one value, which is placed in variable **DBnameVar**.

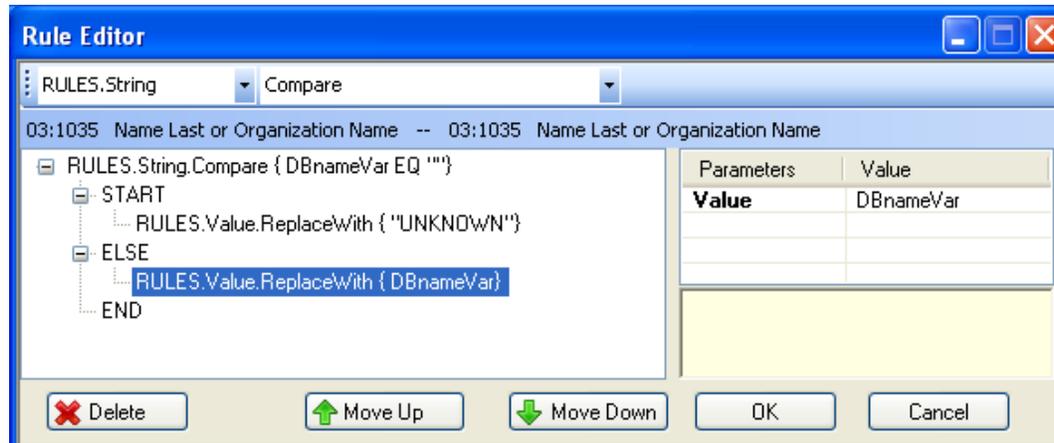


The Parameters area:

Param...	Value
DBRef	ProvDB
SQL	"Select Name from Master 'WHERE (ID=%Current_Element%)"
ResultVar:	"DBnameVar=1"

On another element, a rule checks the contents of **DBnameVar**.

- If it is empty, it uses the literal UNKNOWN in this element.
- Otherwise, it uses the value in DBnameVar.

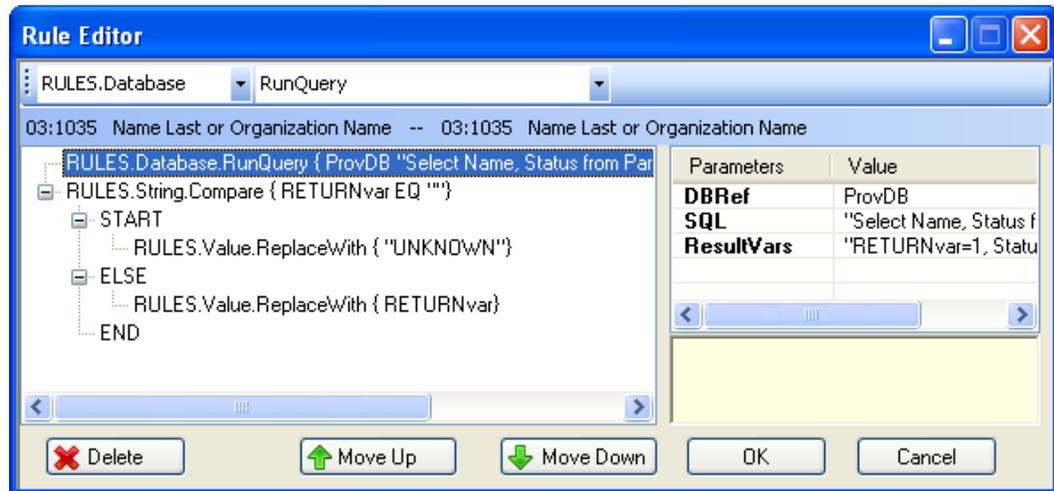


Example 2

This example shows how to return two values. It queries a database with DSN name **ProvDB**. It executes a SQL query and returns the first two values from the query.

The first returned value goes into variable **RETURNvar** and the second goes into **StatusVar**.

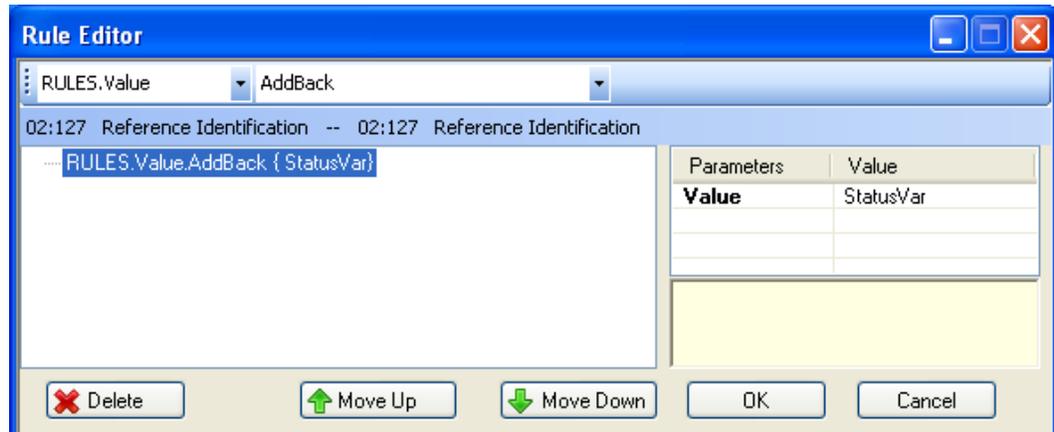
The StringCompare rule checks RETURNvar. If it is empty, it places the literal UNKNOWN in this element's output. Otherwise, it places the contents of RETURNvar in the output.



The parameters area:

Param...	Value
DBRef	ProvDB
SQL	"Select Name, Status from Master WHERE (ID=%Current_Element%)"
ResultVar	"RETURNvar=1 StatusVar=2"

This rule then appends the contents of StatusVar to the output:



RunStoredProcedure

This rule lets you run a stored procedure and receive the results in one or more variables. Please read [Before using Database Rules](#) on page 129 before starting.

Format of Parameters

DBRef Procedure ResultVars

Where:

<i>DBRef</i>	Pointer to the database's DSN name, defined in Foresight Translator's \$Dir.ini (see Before using Database Rules on page 129).
<i>Procedure</i>	The procedure name and parameters needed by the procedure. There are no limits on the number of parameters, but the procedure and variables must be less than 2K.
<i>ResultVars</i>	Variables to hold returned values in the format variable=n, where n is the position of the value that is to be returned. If you have multiple variables, separate them with single spaces.

Demo: Please see _readme_ODBC.txt in Foresight Translator's \DemoData\ODBC directory.

Example

This example runs a stored procedure called **npi_Lookup** in a database with DSN name **MYFSdemo**.

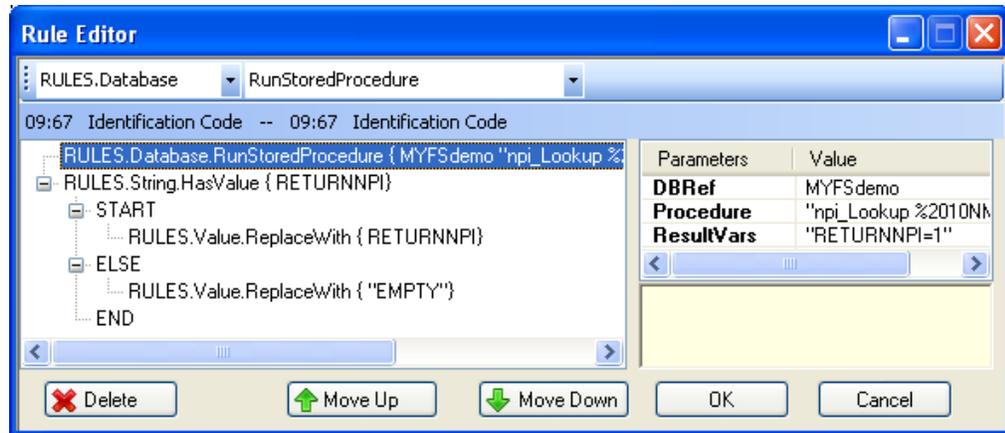
The procedure has two input parameters:

- The first will be the contents of variable 2009.
- The second will be the contents of the current element.

The 1st returned value goes into variable **RETURNNPI**.

The **StringHasValue** rule checks RETURNNPI:

- If it has a value, the value will be used in the current element.
- If not, the current element will get the literal value "EMPTY."



The parameters area:

Parameters	Value
DBRef	MYFSdemo
Procedure	"npi_Lookup %2010NM108% %Current_Element%"
ResultVars	"RETURNNPI=1"

Rules.DateTime

CompareDate

This rule compares two dates in D6 (YYMMDD) or D8 (CCYYMMDD) format.

If the date in Value2 matches the comparison operator, the comparison is considered “true.” If desired, you can then specify additional sub-rules based on the true comparison.

Format of Parameters

DateFormat1 Value1 Operator DateFormat2 Value2

Where:

DateFormat1 Format of first date to compare. Select from:
YYMMDD
CCYYMMDD

Value1 Value of first date to compare.

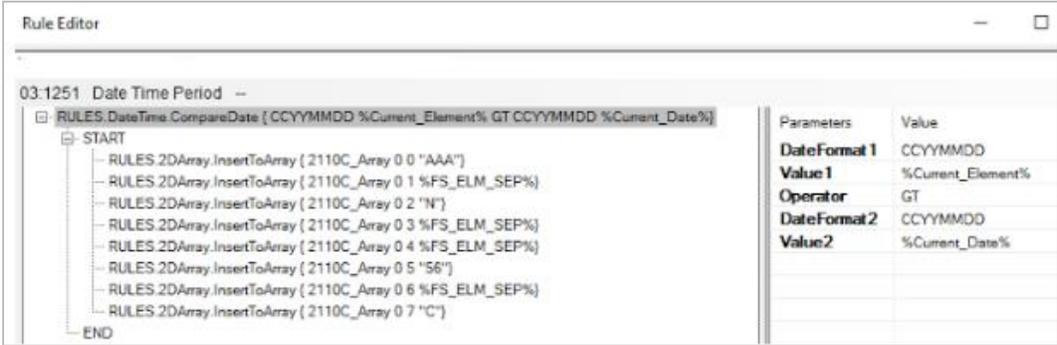
Operator Relational operator for comparison. Select from:
EQ Equal to
NE Not equal to
GT Greater than
GE Greater than or equal to
LT Less than
LE Less than or equal to

DateFormat2 Format of second date to compare. Select from:
YYMMDD
CCYYMMDD

Value2 Value of second date to compare.

Example

This rule compares the date format for Value1 (Current_Element) with the date format for Value2 (Current_Date). If the date in Value2 is Greater Than Value1, the comparison is considered “true” and START END rules are used to build a segment (AAA) into an Array.



Conversion

This rule converts a date to another date format.

Format of Parameters

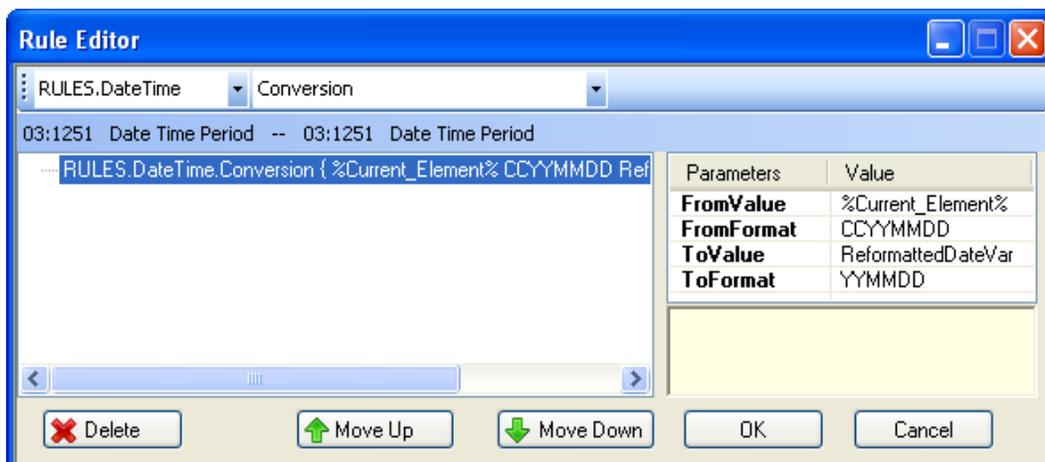
FromValue FromFormat ToValue ToFormat

Where:

<i>FromValue</i>	Contains the value to be converted into the new date format.
<i>FromFormat</i>	The format of the original date before it is converted. Select the format from the list in the parameters area.
<i>ToValue</i>	Variable to contain the reformatted date.
<i>ToFormat</i>	The new date format. Select it from the list in the parameters area. If data is missing (for example, the time is not present in the FromValue), zeros are used.

Example

This rule converts the current value from CCYYMMDD to YYMMDD format and puts the result in variable ReformattedDateVar.



GetGMTDateTime

This rule places the current Greenwich Mean Time (GMT) into a variable.

Format of Parameters

Format ResultVar

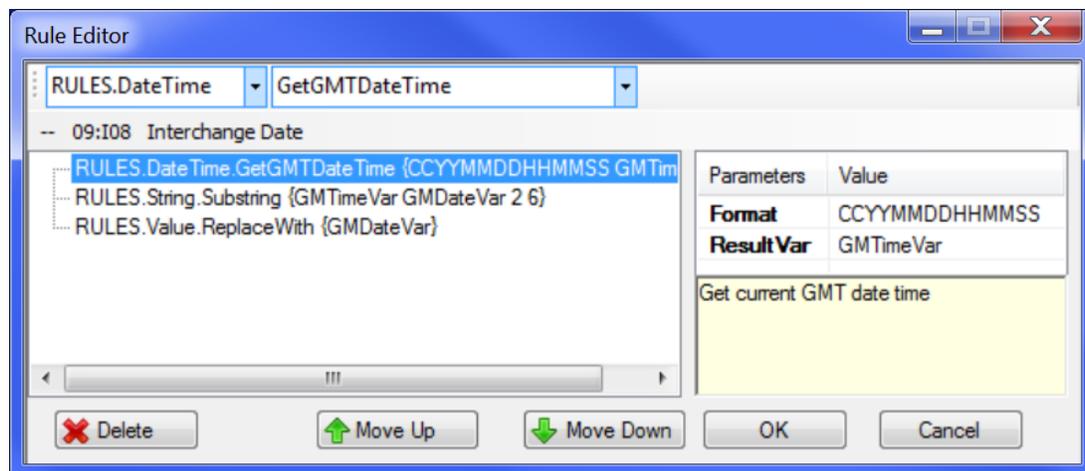
Where:

Format Type a date-time format or choose one from the drop-down list.

ResultVar A variable to hold the result.

Example

This example puts the GMT into variable GMTTimeVar. It then substrings the YYMMDD value into variable GMDateVar, which it uses for the output value on the current element.



GetLocalDateTime

This rule places the current local date and time into a variable.

Format of Parameters

Format ResultVar

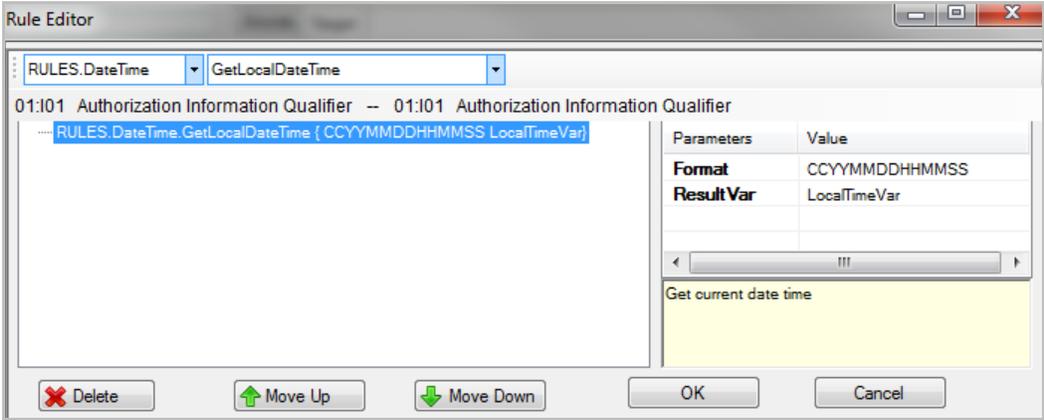
Where:

Format Type a date-time format or choose one from the drop-down list.

ResultVar A variable to hold the result.

Example

This example puts the local date and time into variable LocalTimeVar using the format CCYYMMDDHHMMSS.



ProcessingDate

This rule can go on an unmapped element. It lets you determine the “processing date” of the translation according to criteria that you set up, and put it in a variable.

It starts with the current date and current time and has nothing to do with source data. It then adjusts the date in two ways:

1. Is the current time after a “cutoff” time?

Look at the current time and see if it is after a cutoff time that you have set up. If so, the processing date rolls over to tomorrow. If not, it remains as today.

2. Is the current date in a file of non-processing dates?

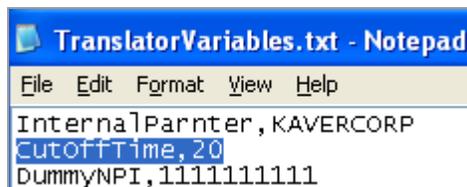
Next, look at the processing date after possible adjustment for cutoff time. See if it is listed in a “rolled forward table” of weekend days, holidays, and other dates that you don’t want used as processing dates. If it is listed in the table, move the processing date forward to the corresponding date in the table.

Setup

Define the cutoff time

The cutoff time is the time when the processing date should move to the next day. For example, if a file is translated after 8 p.m., you may want to consider it processed the next day. The cutoff time is specified with a digit from 1-24, using the 24-hour format, so that 9 means 9 a.m., and 22 means 10 p.m. Use 24 if you don’t want a cutoff time.

If you want the cutoff time to be in an external file, rather than hard-coded into the business rule, create an external variables file as described in [Populating Variables with an External Variables File](#) on page 99. Add an entry like this, which says processing dates should be tomorrow for files translated after 8 p.m.:



```
TranslatorVariables.txt - Notepad
File Edit Format View Help
InternalParnter, KAVERCORP
CutoffTime, 20
DummyNPI, 1111111111
```

Create a “Rolled Forward Table”

In this file, enter weekend days, holidays, and other dates that you don’t want to have considered as processing dates.

The format is non-processing-date, corresponding-processing-date.

In this example, the top line means files translated on October 2, 2010 will have a processing date of October 4, 2010. Also, files processed after the cutoff time on October 1 will have a processing date of October 4.

```

Weekends_Holidays.txt - Notepad
File Edit Format View Help
20101002, 20101004
20101003, 20101004
20101009, 20101012
20101010, 20101012
20101011, 20101012
20101016, 20101018
20101017, 20101018
20101023, 20101025
20101024, 20101025
  
```

Use the filename of your choice and save it to Foresight Translator’s \Bin directory.

Write down the filename. You will need it in the ProcessingDate business rule.

This chart shows what happens when cutoff=8 p.m. and the rolled forward table above is used.

Date/time translated	“Processing date”	Explanation
October 1, 9 a.m.	October 1	Before cutoff, not in rolled forward table, so current date is used as the processing date.
October 1, 10 p.m.	October 4	After cutoff, so date moves to the next day, which is in rolled forward table (first column). Corresponding second column shows October 4, so that is the processing date.
October 5, 9 a.m.	October 5	Before cutoff, not in rolled forward table, so current date is used as the processing date.
October 5, 10 p.m.	October 6	After cutoff, so date moves to the next day, which is not in rolled forward table (first column). Processing date rolls over from the cutoff time but not from the rolled forward table.
October 9, any time	October 12	File is translated on date that is in the rolled forward table, so it rolls to the corresponding date in the file. Cutoff time is not a factor.

Format of Parameters

CutoffTime TimeZone RolledForwardTable DateFormat ResultVar

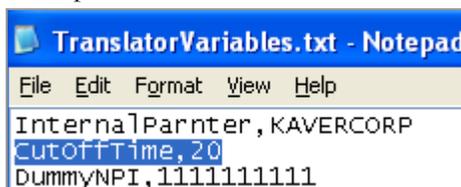
Where:

CutoffTime

Time after which the date advances to the next day.

This can be:

- A literal in quotes. This is a whole number from 1-24.
- A variable. If the variable is populated from an external variables file (see page 99), this name must match the one in the file exactly, including capitalization.



If you do not want a cutoff time, use 24.

TimeZone

Select a time zone if desired. Otherwise the local time zone is used.

FromInputFileName starts with a date and time in the filename itself rather than the date and time that the data was translated. This gives you the option of renaming incoming files with a date-time stamp, and thus processing date can be based in when the file arrived, not when it was translated.

If **FromInputFileName** is selected, the rule looks at the filename to find the date and time. The filename must have this format, and include a file extension:

anyname.CCYMMDD.HHMMSS .extension

Examples:

Myfile.20110430.201555.txt

Myfile.837P.5010.20110430.201555.txt

FromInputFileName does not use the current time and current date at all. Instead, it picks the time from the filename and sees if it is after the cutoff time. If so, it adds a day to the date in the filename. Next, the rule checks to see if the adjusted date is in the rolled forward file. If so, it uses the corresponding date listed to populate the variable.

RolledForwardTable A file that contains dates to roll forward. Please see [Create a “Rolled Forward Table”](#) on page 141. This file is required to make the rule work. If you have no rolled forward dates, you can leave the file empty.

DateFormat Format for the new date. Select from the drop-down list.

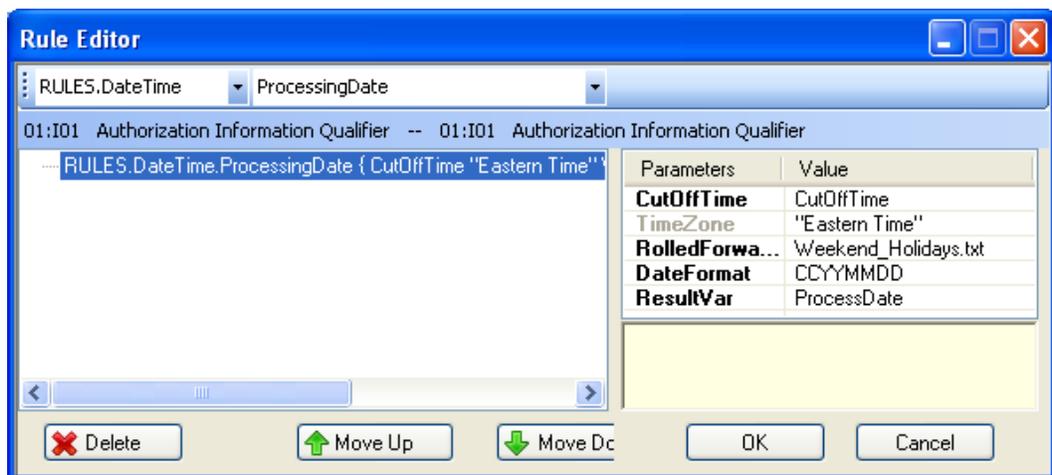
ResultVar Variable to hold the processing date.

Example

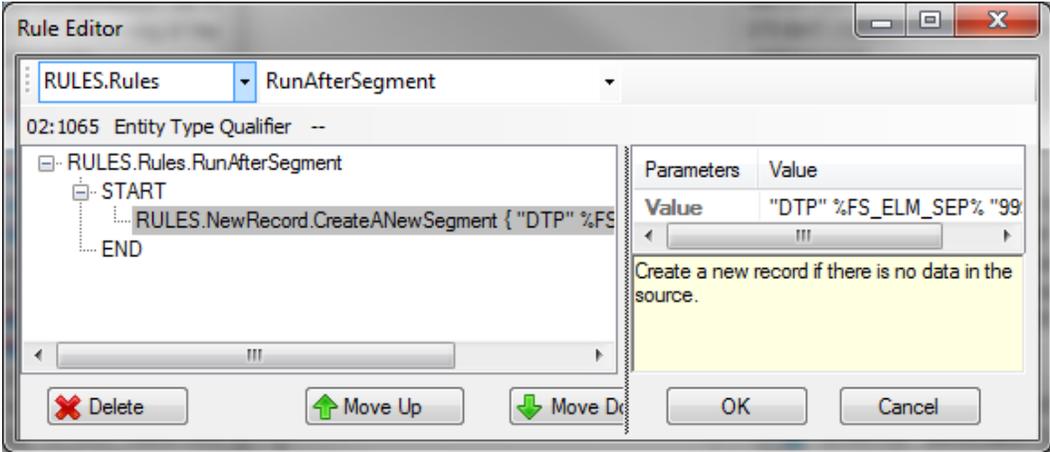
This rule:

- Checks to see if the current time is after the time in the CutOffTime variable.
If true, the date is rolled forward until tomorrow.
- Checks Weekend_Holidays.txt to see if the date (or next day) is listed.
If true, the date is rolled forward to the corresponding date in Weekend_Holidays.txt.
- The date, whether rolled forward or not, is now placed in variable ProcessDate in CCYYMMDD format.

This rule can be on any element, source or target, as long as it appears before the variable ProcessDate is used.



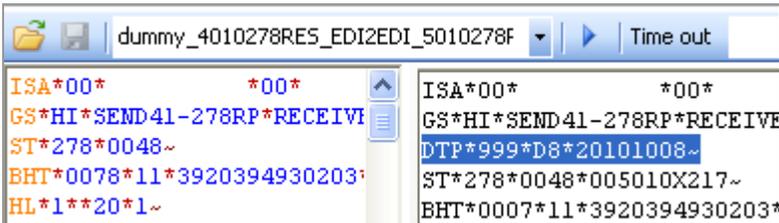
This rule uses ProcessDate in a new DTP segment that is to be inserted in the output:



The Parameters area includes ProcessDate:

Parameters	Value
Value	"DTP" %FS_ELM_SEP% "999" %FS_ELM_SEP% "D8" %FS_ELM_SEP% ProcessDate"

The output shows a new segment with the value in ProcessDate included:



ToRDX

This rule converts a date/time format to date range date/time format and places the updated data in a variable.

Format of Parameters

Date *Format* *Date* *Format* *ResultVar* *ToFormat*

Where:

Date The starting or date/time value to be converted into the new format. It can be a literal in double quotes, a variable or Current_Element.

Format Select the current format of the StartDate, or type a format that is not on the list.

Date The ending date/time to be converted. If this does not exist, the same value as StartDate will be used.

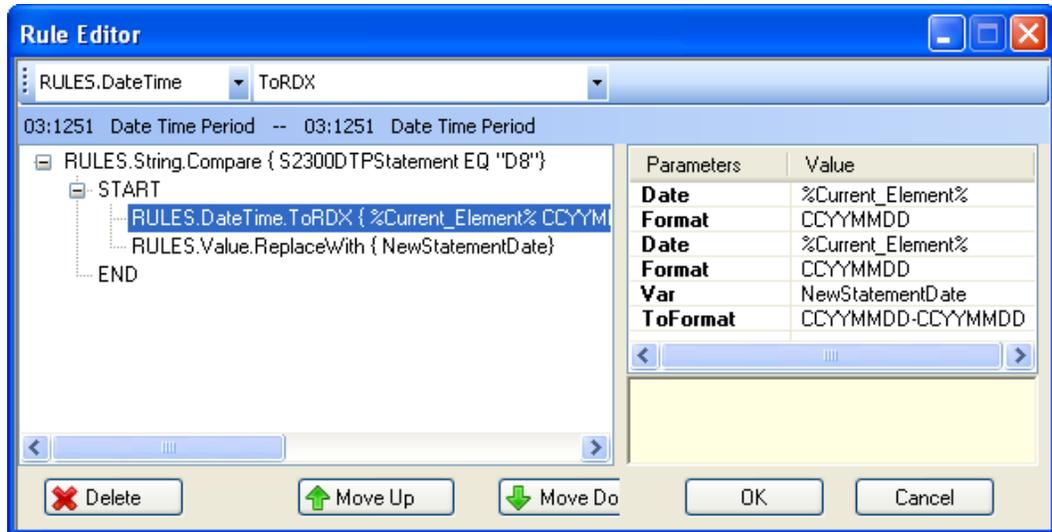
Format Current format of EndDate. For example: 20120812. (See *Format* above.)

ResultVar Variable in which to capture the reformatted data.

ToFormat Format for output.
- CCYY-CCYY
- CCYMM-CCYMM
- YYMMDD-YYMMDD
- CCYMMDD-CCYMMDD

Example

This rule checks a variable containing the date format. If it is D8, it converts the date in the current element from CCYMMDD format to CCYMMDD-CCYMMDD, places the result in variable NewStatementDate, and then uses that in the output.



If the source contained 20040419, then the target would contain 20040419-20040419.

ToXmlDateTimeType

For XML targets only. This rule converts a D6 or D8 date to XML format : *YYYY-MM-DDTHH:MM:SS*.

Format of Parameters

Value *Format*

Where:

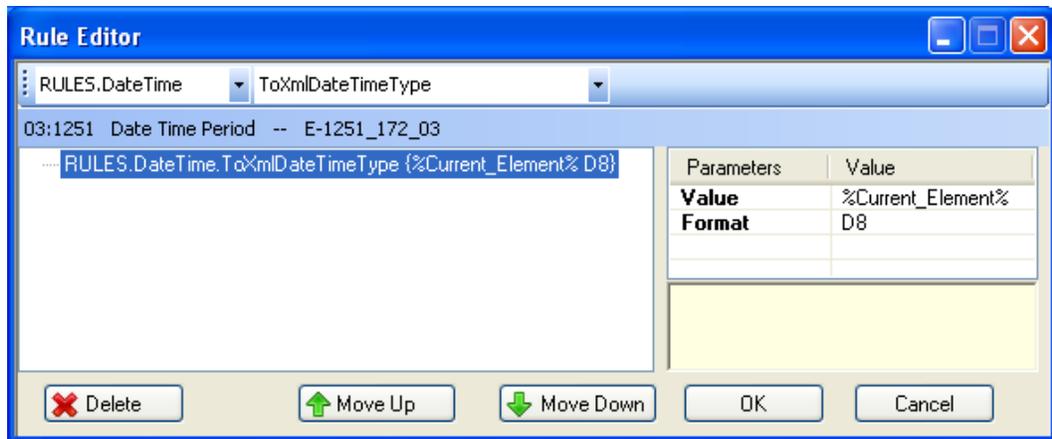
Value Contains the value to be converted into the XML date format.

Format The format of the original date before it is converted:

D6 YYMMDD
D8 YYYYMMDD

Example

This rule converts the date in **ISAdate** from D8 format into the XML datetime format.



Example results:

<pre>REF*2U*12345678900987654321768958~ NM1*QD*1*JOHNSON*BARBARA*T**PH.D~ N3*103 NORTH MAIN STREET*PO BOX 1~ N4*COLUMBUS*OH*43017*US~ CLM*1*100.00***11:A:1*N*A*N*A****~ DTP*096*D8*19971225~ DTP*434*RD8*20030212-20030213~ DTP*435*DT*200206201002~ CL1*1*1*01~</pre>	<pre>- <S-DTP_172> <E-374_172_01>096</E- 374_172_01> <E-1250_172_02>D8</E- 1250_172_02> <E-1251_172_03>1997-12- 25T00:00:00</E-1251_172_03> </S-DTP_172></pre>
---	--

Rules.Encoding

Base64decoding

This rule translates a Base64 value to the equivalent text value.

Format of Parameters

InputValue *Result*

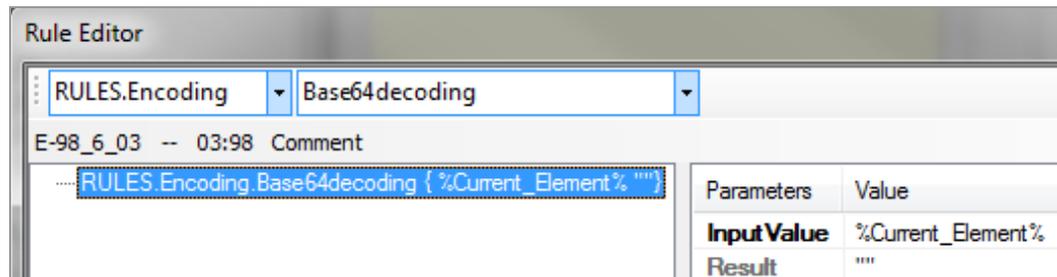
Where:

InputValue A Base64 value – a variable or `Current_Element`.

Result A variable to hold the decoded text value. If this is left blank, the result replaces the current target value.

Example

This target rule converts the Base64 source value to its text equivalent and places the result in the target.



Please see DEMO20 in [Appendix F: Foresight Translator Demos](#) on page 341.

Base64encoding

This rule translates a text value to the equivalent Base64 encoded value.

Format of Parameters

InputValue *Result*

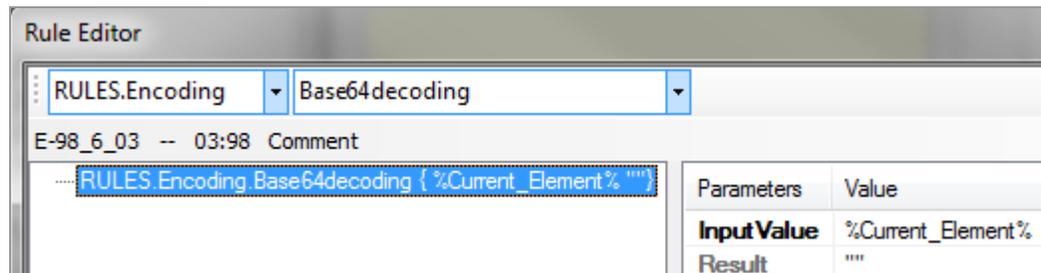
Where:

InputValue The text value – a variable or Current_Element.

Result A variable to hold the Base64 encoded value. If this is left blank, the result replaces the current target value.

Example

This target rule converts a text value in the current element to its Base64 equivalent and places the result in the target.



Please see DEMO19 in [Appendix F: Foresight Translator Demos](#) on page 341.

Rules.External

ISIServerDB

This rule requests that the SQL statement in the translation rule be run via Instream Integration Server. For more information see **ISIServer.pdf** provided with the user documentation for Foresight Translator.

Format of Parameters

Reference FunctionType Statement ReturnCode ResultVars

Where:

<i>Reference</i>	Connection information as set in the ISIServer.config file.
<i>FunctionType</i>	Select “Query” or “Procedure” from the drop down list.
<i>Statement</i>	A stored procedure statement surrounded by double quotes.
<i>Returncode</i>	A variable set to 1 for success, 0 for failed.
<i>ResultVars</i>	Variable(s) to contain the return value(s) separated by spaces and surrounded by double quotes. The format is VAR= <i>n</i> where <i>n</i> is the position of the data to be returned.

ISIServerWS

This rule calls a Web Service through the Instream Integration Server (ISIServer).

ISIServer can be used when Foresight Translator is on Unix or Windows and you cannot directly connect to a database or web service using ODBC. For more information see **ISIServer.pdf** provided with the user documentation for Foresight Translator.

Format of Parameters

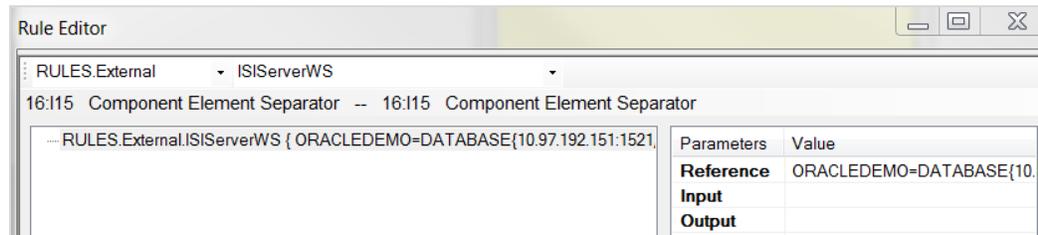
Reference Input Output

Where:

<i>Reference</i>	Connection information as set in the ISIServer.config file.
<i>Input</i>	A 2DArray object that contains input values.
<i>Output</i>	A 2DArray object that contains output values.

Example

This target rule specifies connection information for an Oracle Database.



UserAPIExit

This rule must be built by TIBCO Foresight and makes use of specific requirements provided by the user. Contact TIBCO Foresight Technical Support for information about having this rule customized for your environment.

Rules.File

CreateFileFromArray

This rule creates a file using data in the specified array as input.

Format of Parameters

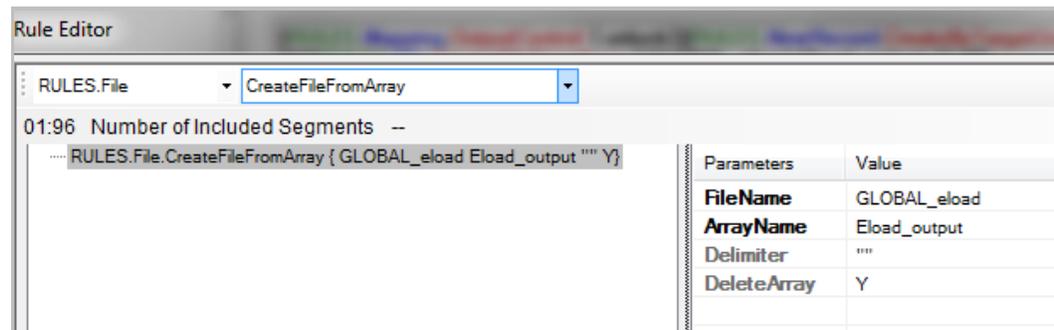
fileName *ArrayName* *Delimiter* *DeleteArray*

Where:

- fileName* Name of file to be created, including path. This can be a variable or a literal without double quotes.
- ArrayName* Name of the array from which the target file will be created. This can be a variable or a literal with double quotes.
- Delimiter* (Optional) Delimiter that separates the columns found in the input data (the array). This value can be a literal or a variable.
- DeleteArray* (Optional) Specify if the array should be deleted after it has been written to the file. (Default is Y.)

Example

This rule specifies a file named GLOBAL_eload should be created using data from the array Eload_output. No delimiters should be used and, after creating the file, the array Eload_output should be deleted.



CreateSearchTable

This rule creates a search table using data in the specified file as input.

Format of Parameters

TableName *FileName* *FileFormat* *Delimiter* *RemoveSpaces*

Where:

TableName Name of table to be created. This can be a variable or a literal without double quotes.

FileName Input file name, including path. This can be a variable or a literal without double quotes.

FileFormat File type; can be DELIMITED or FIXEDLENGTH.

Delimiter Specifies column separation in the table.

For delimited files: This can be a variable or literal with double quotes. Example: “*”

For fixed length files: Use the following format

startposition-fieldlength;startposition-fieldlength

Note a semicolon is used to separate columns.

Example : 3-5;8-6;15-7;27-10;39-9

RemoveSpaces (Optional) Specifies if spaces should be deleted from the data when it is written to the table. (Default is Y.)

Example

This rule specifies a table named GLOBAL_eload_input should be created using data from the file GLOBAL_eload. The file being used to create the table is fixed length, with columns as follows:

The first column begins at position 3 and can have 5 characters.

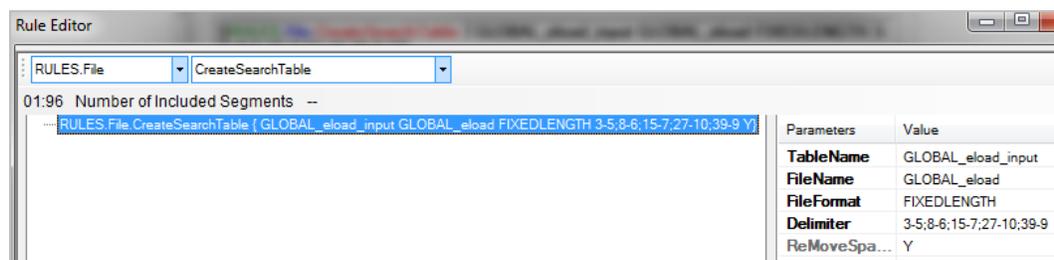
The second column begins at position 8 and can have 6 characters.

The third column begins at position 15 and can have 7 characters.

The fourth column begins at position 27 and can have 10 characters.

The fifth column begins at position 39 and can have 9 characters.

After the table has been created, any extra spaces data should be deleted.



RunTime

This debugging rule can show how long it takes to translate a file or parts of a file. It displays the elapsed time to the console when running Translator.exe.

Format of Parameters

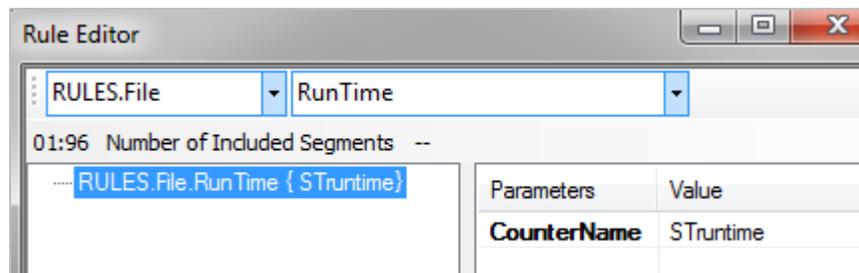
CounterName

Where:

CounterName A name for the counter.

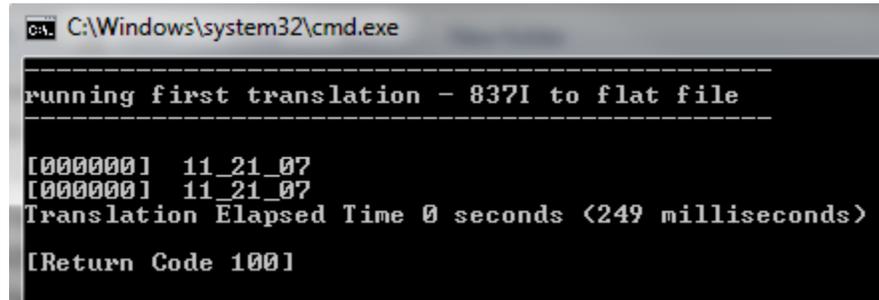
Example

This rule on the SE01 will display the time that each transaction set took to run.



Example output

The file contained two transaction sets and each took 0 seconds.



```
C:\Windows\system32\cmd.exe
-----
running first translation - 837I to flat file
-----
[0000000] 11_21_07
[0000000] 11_21_07
Translation Elapsed Time 0 seconds <249 milliseconds>
[Return Code 100]
```

SwapFile

Future rule. Do not use.

WriteStringToFile

This rule writes the data in the specified string to a file.

Format of Parameters

fileName StringName Newline

Where:

fileName Name of file to be written to, including path. This can be a variable or a literal without double quotes.

StringName This can be a variable or a literal with double quotes.

Newline (Optional) Specifies if the string should be written on a new line. (Default is Y.)

Example

This rule specifies the literal string “This is Test Data only” should be written to the file named GLOBAL_eload_TEST. The content of the string should appear in the file on a new line.

Rule Editor

RULES.File WriteStringToFile

01:96 Number of Included Segments --

....RULES.File.WriteStringToFile { GLOBAL_eload_TEST "This is Test Data only" Y }

Parameters	Value
FileName	GLOBAL_eload_TEST
StringName	"This is Test Data only"
Newline	Y

Rules.Format

COBOL_PIC

This rule lets you convert a value between a decimal and a COBOL Picture. This rule can be used to truncate an EDI value if the EDI value exceeds the length assigned to the COBOL PIC.

Format of Parameters

InputValue *Format* *Option* *ResultVar*

Where:

InputValue Value to be packed.

Format The COBOL PIC value can be:

9(2)V9 Two numbers to the left and no more
 than one number to the right of the
 decimal.

9(2)V99 Two numbers to the left and no more
 than two numbers to the right of the
 decimal.

9(3)V9 Three numbers to the left and no more
 than one number to the right of the
 decimal.

9(3)V99 Three numbers to the left and no more
 than two numbers to the right of the
 decimal.

<*other*> Type in a valid value of your choice (e.g.
 9(2)).

Option Type of conversion.

From From COBOL Picture

To To COBOL Picture

ResultVar Variable to hold the resulting packed value (literal
 with/without double quotes).

Example 1

For this example:

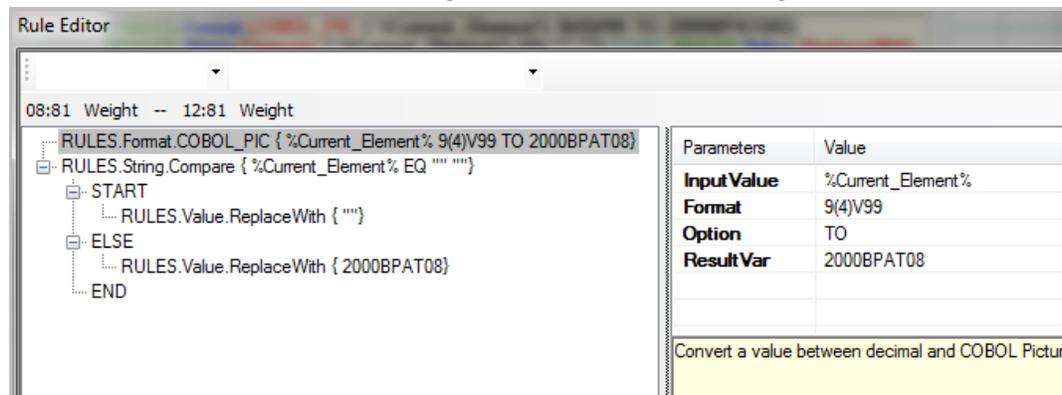
- The COBOL PIC value for the 5010 837P 2000B PAT08 is 9(4)V99, which means there can be no more than 4 numbers to the left of decimal and no more than 2 number to the right of the decimal.
- Our EDI has a value of 798.078 for the field.

If we do nothing, the entire value (798.078) is mapped, exceeding the expected COBOL PIC length (9(4)V99) and causing the outbound Flat File to be 'Not Well Formed'.

We could use the FloatingDecimal rule to convert the value to the COBOL PIC length, but the value would be rounded up from 7894.078 to 789408. **We do not want the value rounded.**

If we use the COBOL_PIC rule, as shown below, the value will truncated in accordance with the COBOL_PIC length (no more than 2 number to the right of the decimal) with no rounding, giving us the value 789407.

This shows the rule written on the Target, when the Source and Target are linked.



The screenshot shows the 'Rule Editor' window. The main area displays a rule tree with the following structure:

```
08:81 Weight -- 12:81 Weight
├── RULES.Format_COBOL_PIC { %Current_Element% 9(4)V99 TO 2000BPAT08 }
├── RULES.String.Compare { %Current_Element% EQ "" "" }
│   ├── START
│   │   └── RULES.Value.ReplaceWith { "" }
│   └── ELSE
│       └── RULES.Value.ReplaceWith { 2000BPAT08 }
└── END
```

On the right side, there is a 'Parameters' table:

Parameters	Value
Input Value	%Current_Element%
Format	9(4)V99
Option	TO
Result Var	2000BPAT08

Below the table, there is a description: 'Convert a value between decimal and COBOL Pictur'.

Example 2

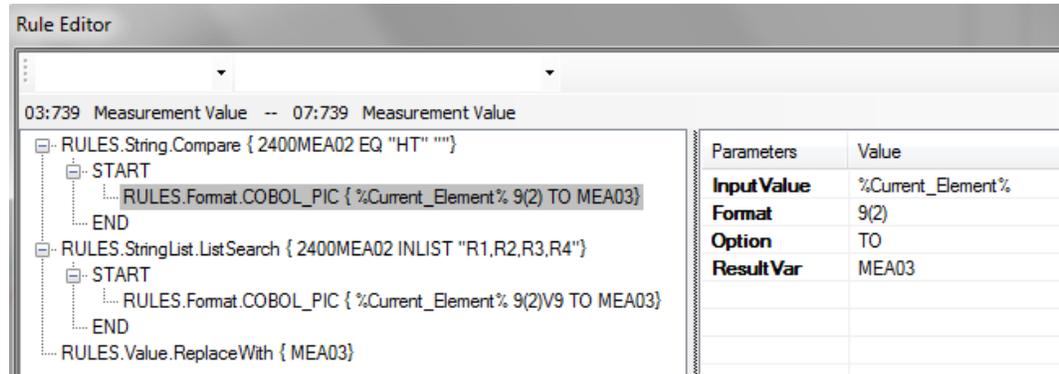
For this example:

- The COBOL PIC value for the 5010 837P 2400 MEA03 is:
 - 9(2) when the MEA02 = HT, and
 - 9(2)V9 when the MEA02 = R1, R2, R3, or R4.
- Our EDI has a value of 10.08 for the 2400 MEA03. **We do not want the value rounded.**

When the EDI data has a value of 2400 MEA02 = HT then the 2400 MEA03 value of 10.08 should map as 10. (9(2) specifies no more than two numbers to the left of the decimal and zero numbers to the right of the decimal. The decimal is dropped from the value as there are no numbers after it.)

When the EDI data has a value of 2400 MEA02 = R1 then the 2400 MEA03 value of 10.08 should map as 10.0. (9(2)V9 specifies no more than two numbers to the left of the decimal and one number to the right of the decimal.)

This shows the rule written on the Target, when the Source and Target are linked.



FloatingDecimal

This rule allows you to specify if a decimal should be included and where it should go. It corresponds to the printf function in several programming languages.

Format of Parameters

InputValue *Format* *ResultVar*

Where:

InputValue Contains the number to be formatted. This is a number containing a decimal. If no decimal is included, a trailing decimal is assumed.

Format The format. See **Formats** below.

ResultVar Variable to contain the newly formatted value.

Formats

Use these settings, in order:

Y or N	Show decimal point or not in output.
%	Literal
<i>TotalLength</i>	Total string length, counting the decimal.
<i>DecimalLength</i>	Number of digits after the decimal, rounded if necessary.
<i>DataFormat</i>	f – floating decimal output d – signed integer output s – string output

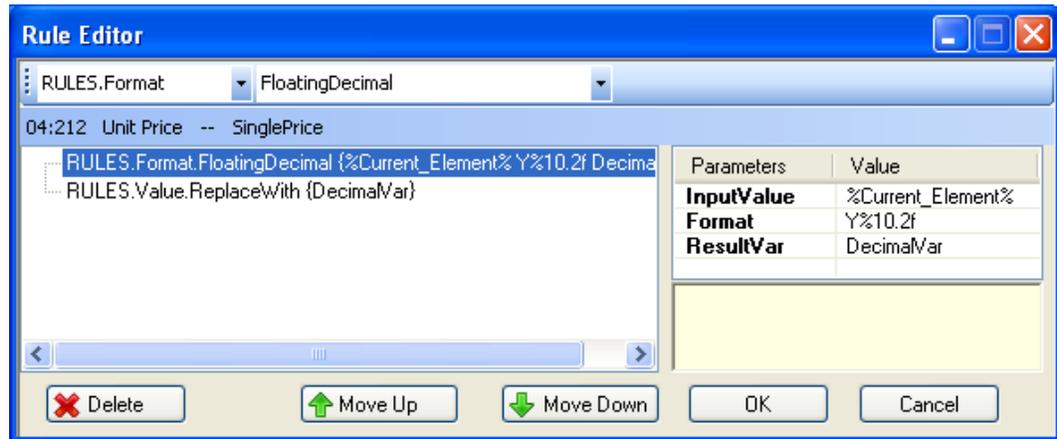
Example

The first rule formats the value in the current element this way:

- Y Display a decimal in the output
- 10 Up to 10 digit total length, counting the decimal
- 2 Place two digits after the decimal
- f Is a floating point value, so put a decimal in the output

It places the reformatted value in variable **DecimalVar**.

The second rule uses the contents of **DecimalVar** for the current element's output.



Example results for Y%10.2f example above:

Source Data	Target Data
12345	12345.00
12345.	12345.00
1234.5	1234.50
123.45	123.45
12.345	12.35 (rounded up)

Example formats		
Contents of varA	Value wanted in varB	Format parameter in rule
123456	123456.00	Y%10.2f Decimal is assumed after 6; format specifies two decimal places, so zeros are added.
1234.56	1234.56	Y%10.2f
123456	123456.0	Y%10.1f
1234.56	1234.6	Y%10.1f Format specifies one decimal place so decimal digits are rounded.
123456	1234560	N%10.1f Decimal is assumed after 6; format specifies one decimal place. N specifies that the decimal not be put in output.
1234.56	12346	N%10.1f Format specifies one decimal place so .56 is rounded to .6. The N specifies that the decimal not be included in the output.
123456	123456	Y%10.2d or Y%10.1d or N%10.1d
1234.56	1234	Y%10.2d or Y%10.1d or N%10.1d

Alignment of Output

If an output value does not meet the field's minimum length, you have a choice for alignment:

Default Output is right-justified and padded with leading spaces.

Pad with 0 Put a 0 (zero) before the total length: Y%010.1d.

Left-justify Put a minus sign before the total length: Y%-10.1d.

LeftJustified

This rule lets you left-justify a value and specify a fill character if the value does not meet the field's minimum length.

Format of Parameters

FillChar InputValue ResultVar

Where:

FillChar A character to use to fill after the value if it doesn't meet the field's minimum length.

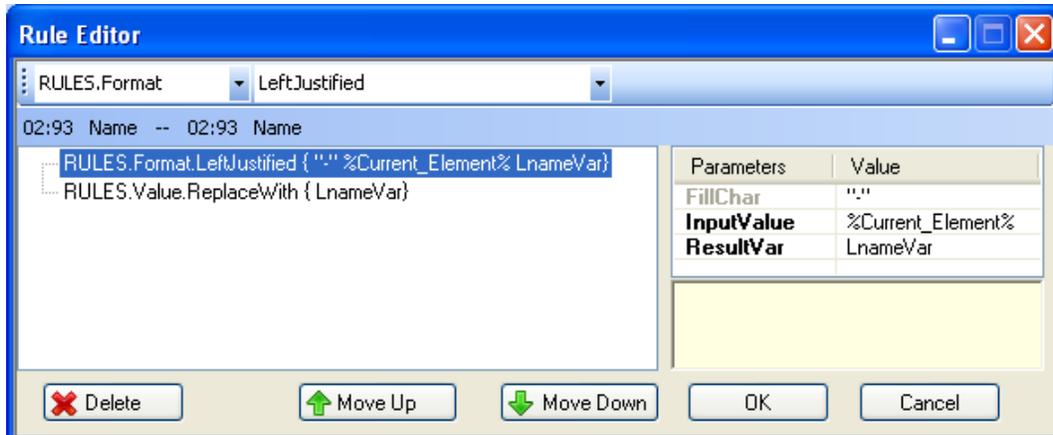
InputValue Current_Element or a variable containing the value to be left-justified.

ResultVar A variable to contain the output.

Example

The first rule places the current element's value in variable **LnameVar** and specifies that it be left-justified and filled to the minimum output field length with x.

The second rule places the contents of LnameVar in the output.



MonetaryAmount

This rule removes leading or trailing zeros in real numbers.

Format of Parameters

InputValue NewValue

Where:

InputValue Value to be changed.

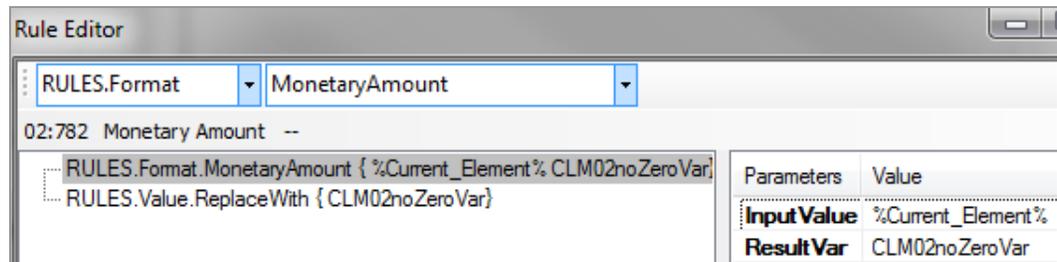
ResultVar A variable to contain the newly-formatted value.

Example Conversions

Input	Output
-0.87	-.87
8.30	8.3
1.00	1
0.3	.3

Example Rule

In this example, the rule removes insignificant zeros from the current element.



Packeddecimal

This rule lets you convert a value to a packed-decimal format. Packed decimal format identifier is designated with an S (signed) and specifies the number of characters, decimals, and if the value is positive or negative.

See also [IsUnPacked](#) on page 258.

Format of Parameters

InputValue *Format* *ResultVar*

Where:

InputValue Value to be packed.

Format Packed decimal format. See the format examples below. Can be a literal that you type, or select from the drop-down list.

ResultVar A variable to contain the newly-formatted value.

Format Examples

The concept and formatting of packed-decimals is widely available via the Internet and other reference sources. A few examples are shown here:

- S98V99 = 8 characters with 2 decimals
- S97V999 = 7 characters with 3 decimals

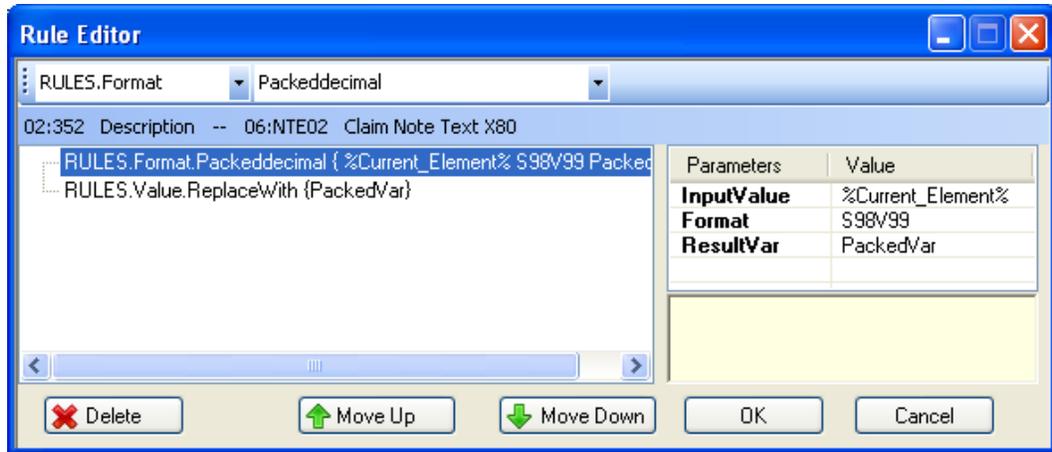
Examples of a flat file definition are:

- S98V99: 000000451{ = +45.10
- S999V999: 00100} = -1

The last character ({ or }) indicates if the number is positive or negative.

Example

In this example, the rule converts the current element to packed-decimal format S98V99, saves the result to a variable called **PackedVar**, and replaces the current element with the contents of **PackedVar**. S98V99 means 8 characters to the left of decimal and 2 to the right, so the value will be the format 00000000.00.



Pad

Pads a value to meet the minimum length of the element or field.

Format of Parameters

InputValue FillChar Justified Length ResultVar

Where:

InputValue Current_Element or variable containing the value to be padded.

FillChar One keyboard character to use for padding, surrounded by double quotes.

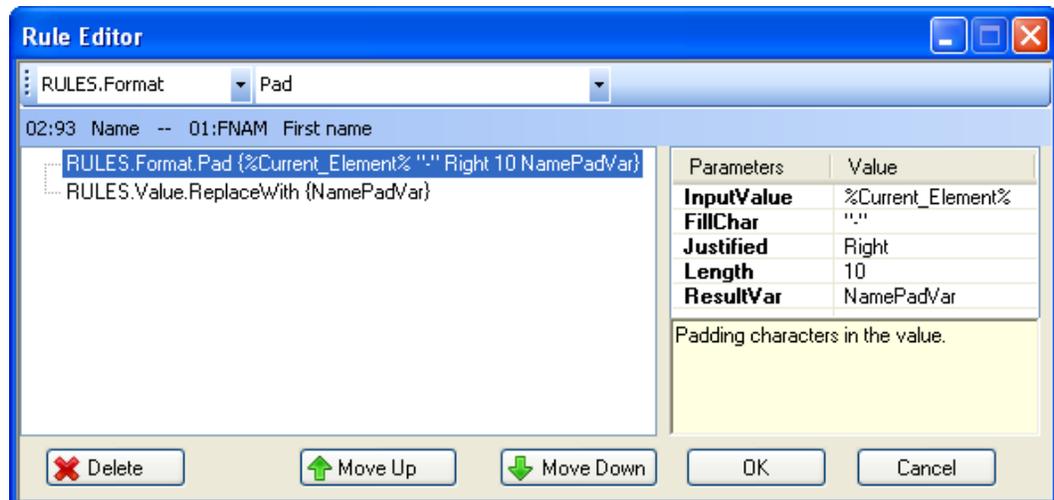
Justified Choose how to justify the value:
Left = padding goes after the value
Right = padding goes before the value

Length The minimum length of the element or field.

ResultVar Variable to hold the resulting value.

Example

We want to pad this element with leading dashes if necessary to meet the minimum length of 10:



RightJustified

This rule lets you right-justify a value and specify a fill character if the value does not meet the field's minimum length. It is similar to [LeftJustified](#). (See page 162.)

SetDecimal

This rule inserts a decimal point into an integer.

Format of Parameters

InputValue Position ResultVar

Where:

InputValue The integer to be formatted.

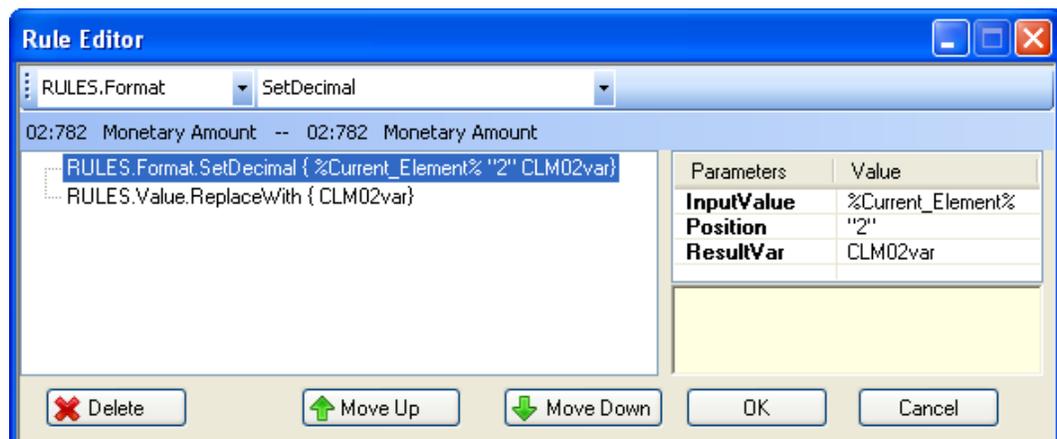
Position The position where the decimal point should be inserted, counting from the end of the value.

ResultVar A variable to contain the newly-formatted value.

Example

The first rule takes the current value, inserts a decimal two digits from the right, and puts the resulting value in variable **CLM02var**.

The second rule uses the contents of CLM02var in the output field.



Example results:

Input	Output
123456	123.456
123	0.123
12	0.012

ToCDATA

This rule creates a CDATA (character data) string. CDATA is used in SGML markup language and XML.

Format of Parameters

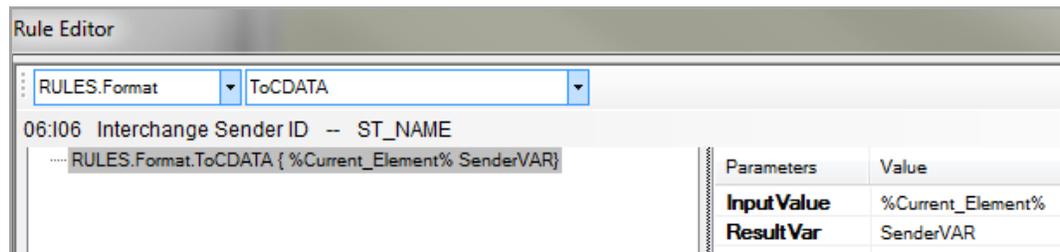
InputValue *ResultVar*

Where:

InputValue The value to be formatted.

ResultVar A variable to contain the newly-formatted value.

Example



Source formatted like this:

```
<sender>John Smith</sender>
```

Creates a CDATA string in the target like this:

```
<![CDATA[<sender>John Smith</sender>]]>
```

ToNxType

This rule converts a value to an N (implied decimal) value.

Format of Parameters

InputValue *Position* *ResultVar*

Where:

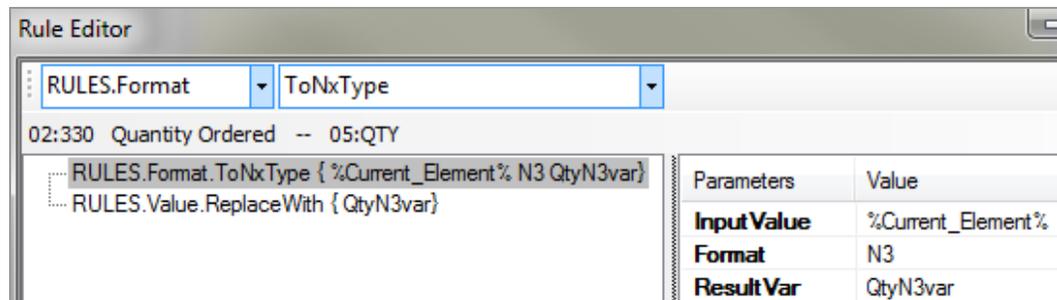
InputValue The number to be formatted.

Format The output value's format: N, N1, N2, N3, N4.

ResultVar A variable to contain the newly-formatted value.

Example

The first rule takes the current value, adds three zeros to its end and puts the result in **QtyN3var**. The second rule puts the new implied decimal value in the output.



Example conversions

Input value	Format	Output value
15	N1	150
15	N3	15000
15.0	N1	150
15.0	N3	15000
15.01	N1	15010
15.01	N3	150

Trim

This rule removes the specified character(s) from the leading and trailing (left and right) sides of a value and places the remaining value into a variable.

Format of Parameters

InputValue TrimChar ResultVar

Where:

InputValue Current_Element or variable containing the value to be trimmed.

TrimChar Optional; default is spaces.

One or more characters to trim, surrounded by double quotes.

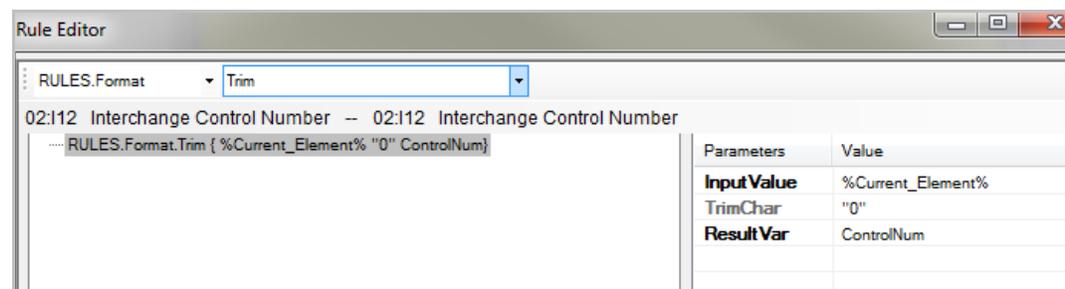
If you supply one character, all occurrences of that character will be trimmed from the left and right of the original value. Example: "0" removes all leading and trailing zeros.

If you supply multiple characters, all sets of them will be trimmed from the left and right of the original value. Example: "12" removes all leading and trailing sets of 12.

ResultVar A variable to hold the trimmed result.

Example

We want to trim leading and trailing zeros. If the original value is 000011100, we want the value 111 to be stored in variable **ControlNum**.



TrimLeft

Removes specified characters from the left of a value and places the remaining value into a variable.

Format of Parameters

InputValue TrimChar ResultVar

Where:

InputValue Current_Element or variable containing the value to be trimmed.

TrimChar Optional; default is spaces.

One or more characters to trim, surrounded by double quotes.

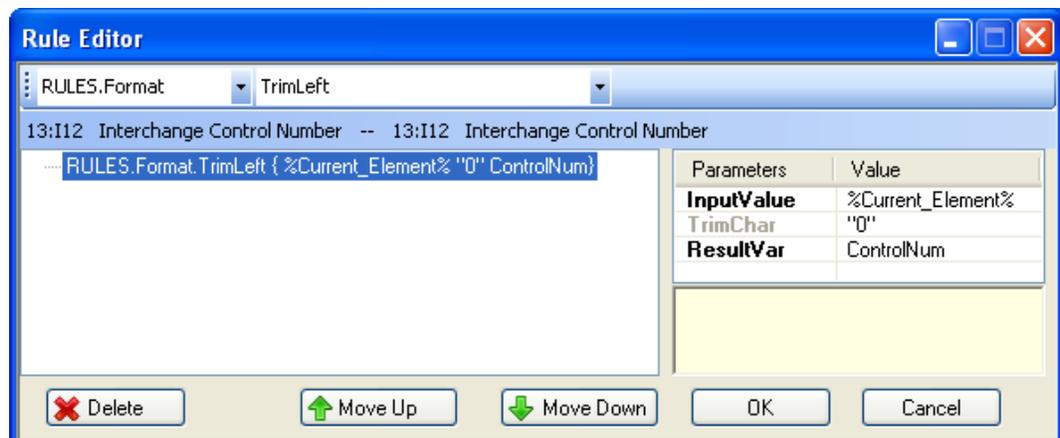
If you supply one character, all occurrences of that character will be trimmed from the front of the original value. Example: "0" removes all leading zeros.

If you supply multiple characters, all sets of them will be trimmed from the location you chose. Example: "12" removes all leading sets of 12.

ResultVar A variable to hold the trimmed result.

Example

We want to trim leading zeros. If the original value is 0000111, we want the value 111 to be stored in variable **ControlNum**.



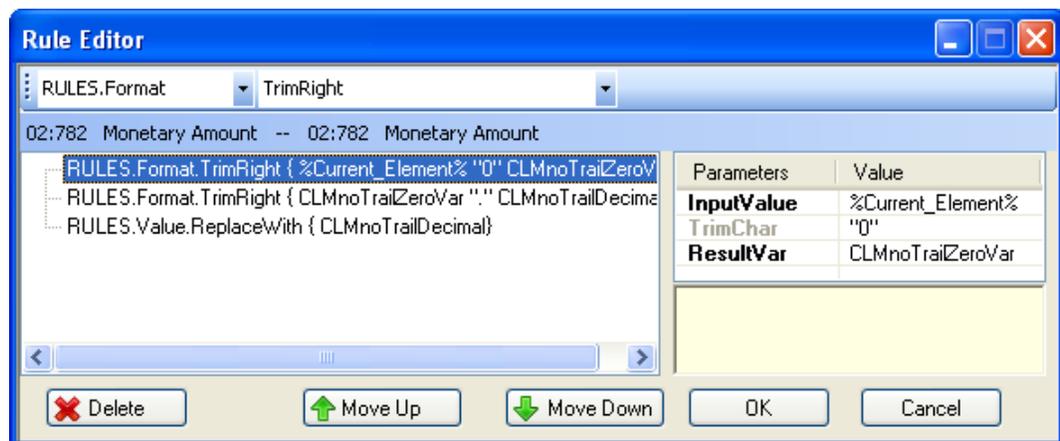
TrimRight

Trims the characters of your choice from the right of value and places the remaining value into a variable. This is just like TrimLeft except it removes trailing characters.

Example

```
- RULES.Format.TrimRight { %Current_Element% "0" CLMnoTrailZeroVar}  
- RULES.Format.TrimRight { CLMnoTrailZeroVar "." CLMnoTrailDecimal}  
- RULES.Value.ReplaceWith { CLMnoTrailDecimal}
```

- The first rule trims any trailing zeros from the end of the current element and puts it in variable CLMnoTrailZeroVar.
- The second rule further trims the value by removing trailing decimals, if any.
- The third rule uses the result in the output.



Rules.HL7

AddEscape

When converting from XML to HL7, Foresight Translator automatically escapes delimiters embedded in the data. For example, if the data contains:

```
<Facility_Name>HILL~SPCS</Facility_Name>
```

...the ~ will be converted to the proper HL7 escape character:

```
HILL\R\SPCS
```

The AddEscape rule lets you convert other values to their HL7 escape equivalents.

Format of Parameters

Input Option Result

Where:

Input Current_Element or variable containing the value to be escaped.

Option One of these:

Xdd Converts the input value to hexadecimal, preceded with X.
Example: 56 becomes X3536

Cxxyy future

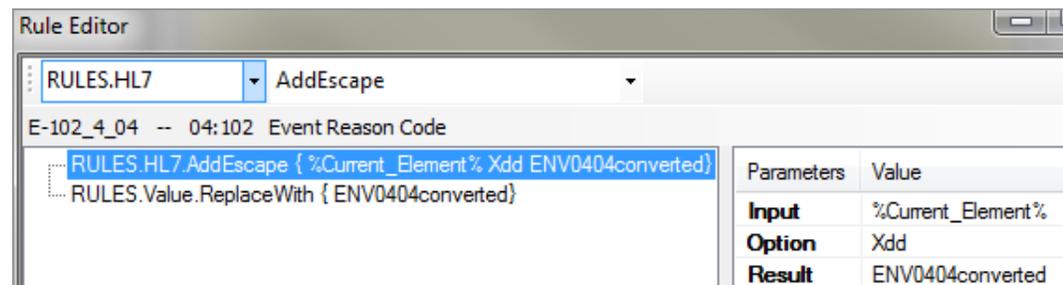
Mxxyyzz future

Zdd future

Result Variable to hold the converted value.

Example

This takes the XML source value and converts it to hexadecimal, preceded with X. It then uses the converted value in the output.



Rules.ICDCodes

The ICDCodes rules convert diagnosis or procedure codes between ICD-9 and ICD-10. These require TIBCO Foresight® ICD-10 Conversion Adapter, which is a separate product that extends the capabilities of Foresight Translator to handle ICD-9 to ICD-10 conversions.

You have two ways of doing this in Foresight Translator:

1. Input one code and get out another. Use a ConvertOne rule. In Demo_ICD_9to10_837I.map, see the rules on the first HI segment's two composites.
2. Input multiple codes plus other related data. The code plus accompanying data are all considered when VIPS chooses codes to return. Use BuildVariable rules and a Convert rule.

See **ICD_at_Foresight.pdf** for details on the following rules:

- Convert
- ConvertOne
- InsertToArrayWithType

Rules.JavaAPI

GetValue

This rule uses the Java API to get a value from the Java program. Foresight Translator must be invoked via Java API for this rule to work.

See also [Appendix B: Java API](#) on page 321.

Format of Parameters

Request ReturnVar

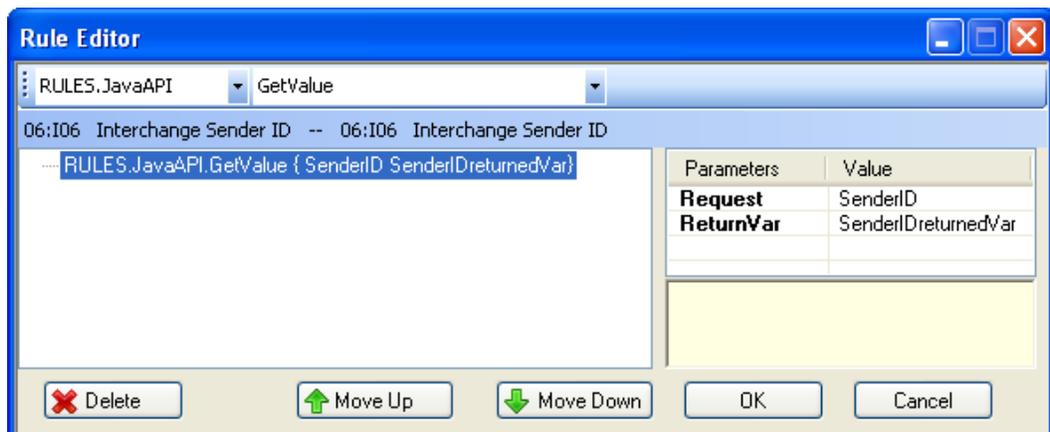
Where:

Request Information that Foresight Translator wants from the calling program.

ReturnVar Variable returned to Foresight Translator.

Example

This rule asks for the contents of the calling program's SenderID. It will be returned to Foresight Translator in the variable SenderIDreturnedVar.



UpdateValue

This rule sends a variable back to the Java program that called Foresight Translator. Foresight Translator must be invoked via Java API for this rule to work.

Format of Parameters

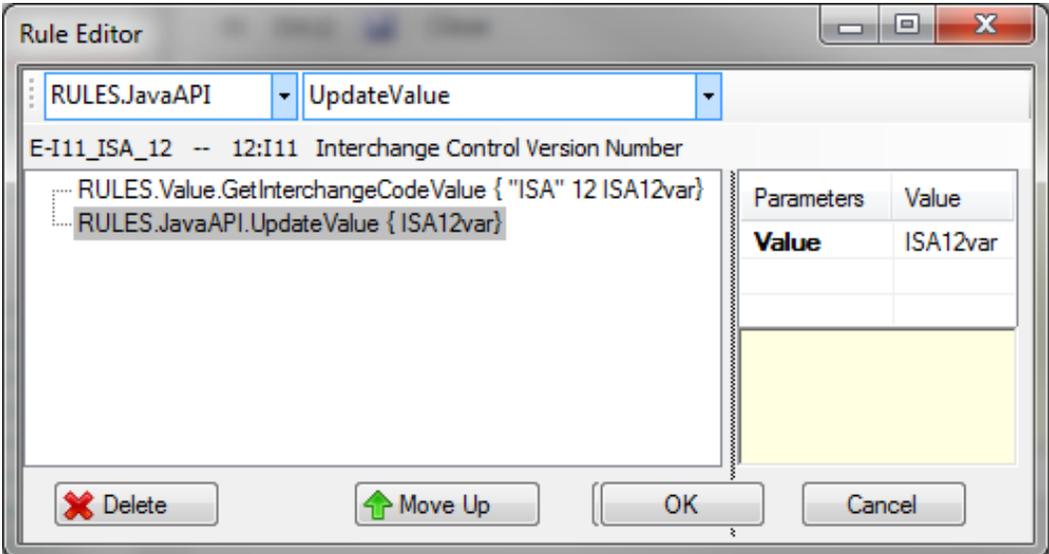
Value

Where:

Value Variable containing information that the calling program wants from Foresight Translator.

Example

This example captures the target's ISA12 (Interchange Control Version Number) into a variable ISA12var. The next rule sends the value to the Java program that called Foresight Translator. This information would be useful for TIBCO BusinessConnect™, for example, when it is compiling enveloping for the target document.



Rules.Mapping

Mapping rules specify when to map.

Cancel

This rule cancels mapping for the current element or segment if certain values are found in an element in the same segment/record.

Format of Parameters

Option

Where:

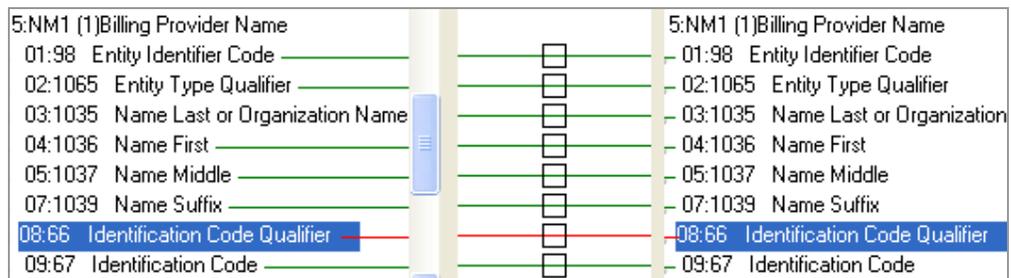
Option *position=value*

The target's element position that controls whether mapping is cancelled, followed by an equal sign, followed by the value in that element that causes mapping to be cancelled.

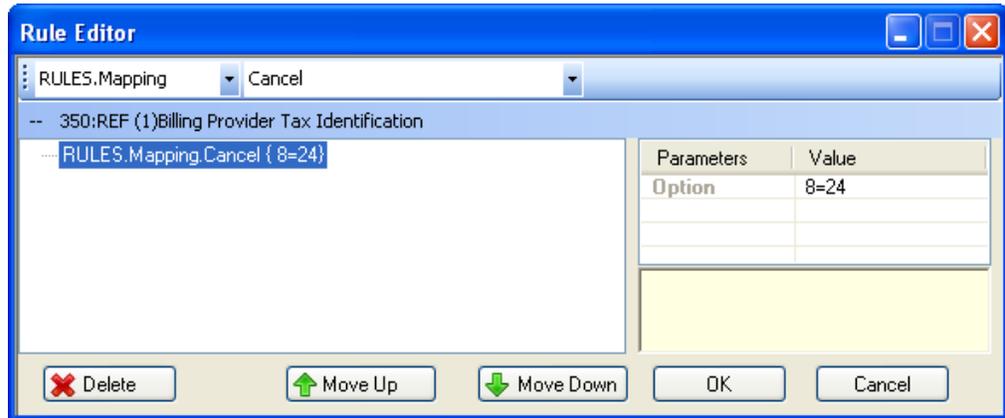
Example 1

If the **NM108** contains **24**, we do not want to map the **NM108** and **NM109**.

1. Map both elements:



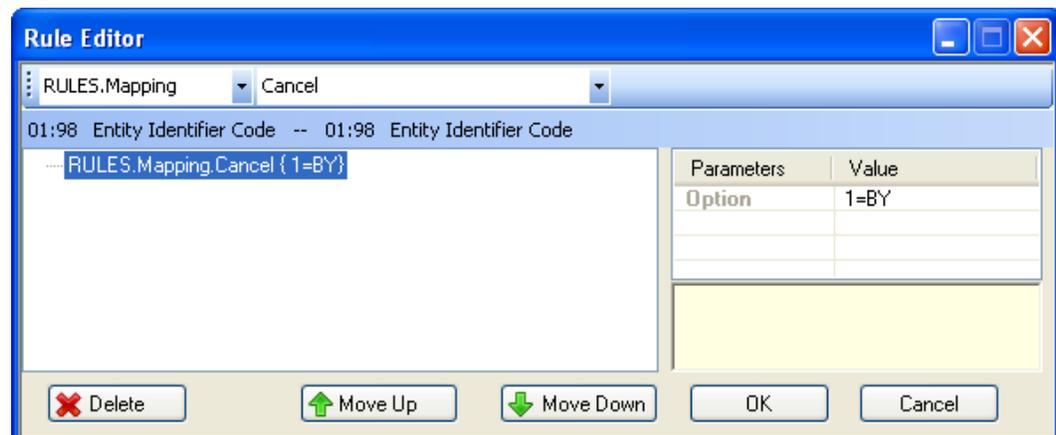
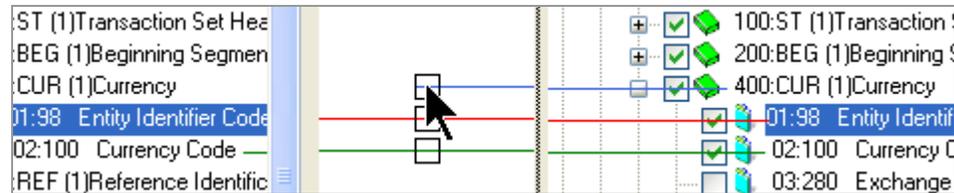
- Put this target rule on the NM108 and NM109:



Example 2

If the **CUR01** contains **BY**, we do not want to map any part of the CUR segment.

If it contains other values, we do. Put this rule on the segment.



CancelWithCondition

This rule specifies conditions that cancel mapping for a segment. The conditions are more complex than with the Mapping.Cancel rule.

Format of Parameters

Value1 Operand Value2 Condition Value3 Operand Value4

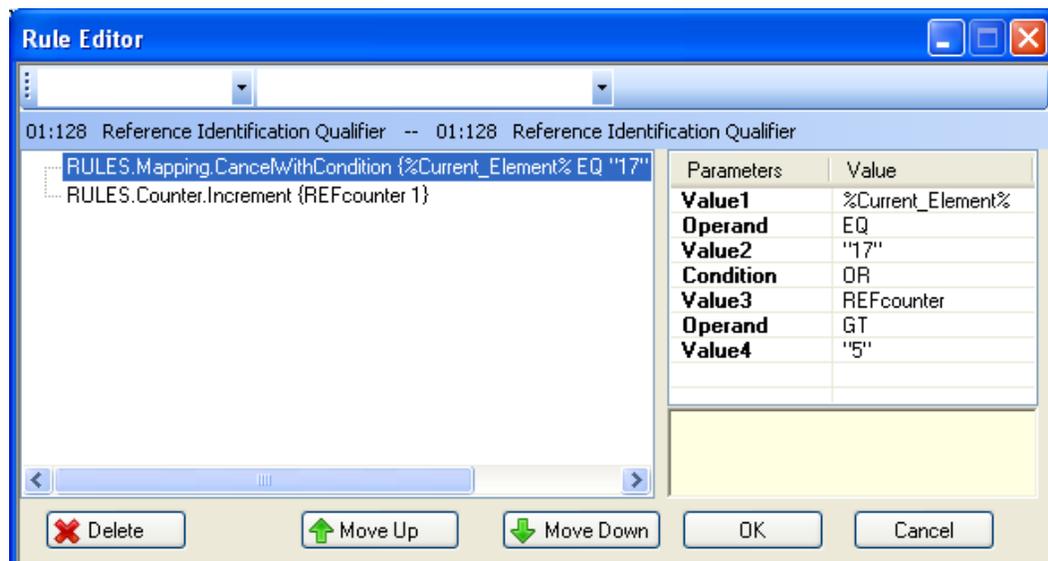
Where:

<i>Value1</i>	The first value to compare.
<i>Operand</i>	Relational operand to use for comparison EQ, NE, GT, GE, LT, or LE.
<i>Value2</i>	The second value to compare.
<i>Condition</i>	Logical operand to use for comparison AND/OR
<i>Value3</i>	The third value to compare.
<i>Operand</i>	Relational operand to use for comparison EQ, NE, GT, GE, LT, or LE
<i>Value4</i>	The fourth value to compare.

Example 1

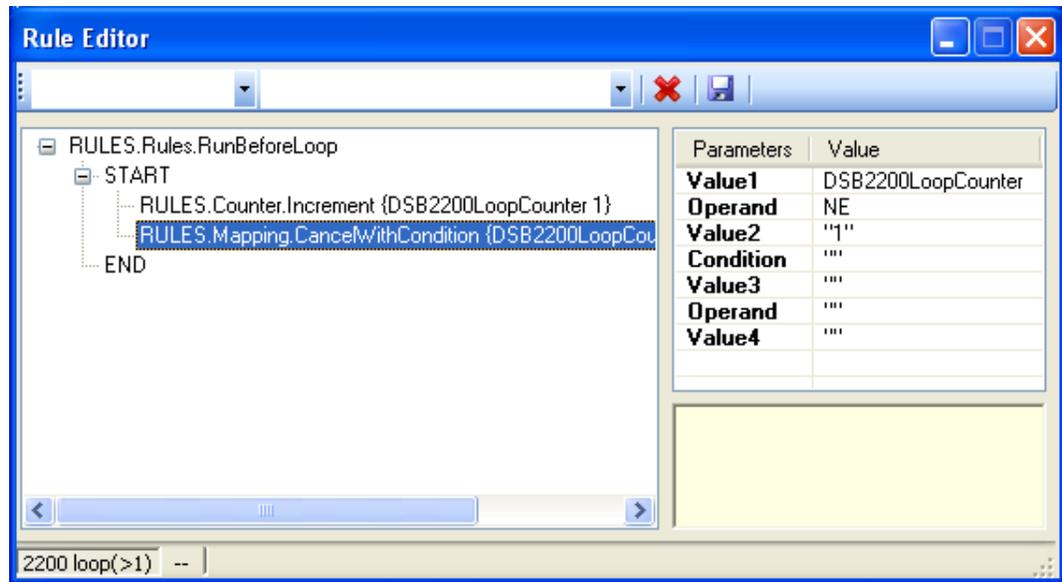
If any of the conditions are true, cancel the mapping for the segment.

The next rule increments a counter.



Example 4

This rule is on a loop header. Increment a counter and then cancel the loop mapping if the counter does not contain 1.



CodeLookup

Future rule. Do not use.

GetVariableSize

This debugging rule for Translator.exe prints out how many variables have been created.

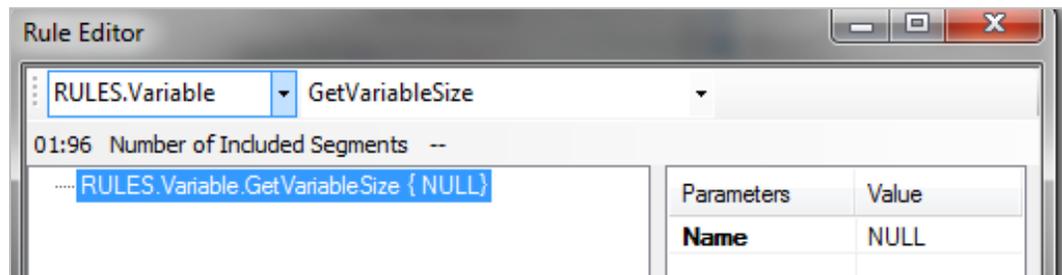
Format of Parameters

Name

Where:

Name Must be set to NULL.

Example



Example output

```
-----  
running first translation - 837I to flat file  
-----  
Variable size = 1  
Translation Elapsed Time 0 seconds (237 milliseconds)  
[Return Code 100]
```

MappingSegentWithCondition

Future rule. Do not use.

MappingWithCondition

This rule lets you map source and target only if a specified condition is met.

It is designed for situations where a source item can map to one of a number of target fields, depending on a comparison between two values.

Format of Parameters

Expression

Where:

Expression A numerical comparison in this format:

ValueA Operator ValueB

ValueA and *ValueB* can be variables, literals in quotes, or reserved words like *Current_Element*.

Operator can be:

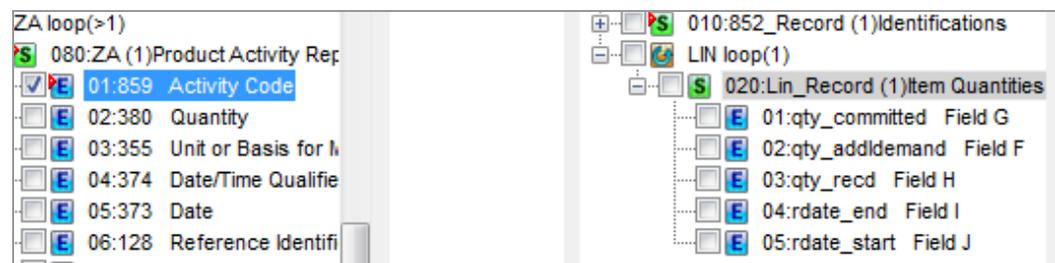
EQ	Equal to
NE	Not equal to
GT	Greater than
GE	Greater than or equal to
LT	Less than
LE	Less than or equal to

The comparison can be more complex with the addition of AND or OR plus parentheses, as in this example format:

Expression1 AND (Expression2 OR Expression3)

Example

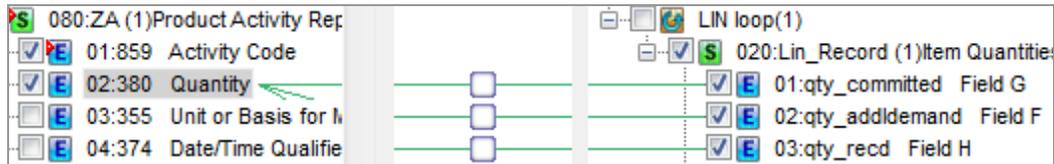
The code value in the ZA-01 determines where the value in the ZA-02 (Quantity) goes.



Therefore, we capture the ZA-01 value in a variable:

RULES.Variable	Set
01:859 Activity Code --	
.....RULES.Variable.Set { %Current_Element% ZA01var}	
Parameters	Value
Value	%Current_Element%
ResultVar	ZA01var

We then map the ZA-02 to each of the three target qty fields:



We add **MappingWithCondition** rules to each target element to be sure they only get the quantity when it is appropriate.

On the 01:qty_committed:

RULES.Mapping	MappingWithCondition
02:380 Quantity -- 09:qty_pos_adj Field 15	
.....RULES.Mapping.MappingWithCondition { "ZA01 EQ "QC" }	
Parameters	Value
Expression	"ZA01 EQ "QC" ""

On the 02:qty_addldemand:

RULES.Mapping	MappingWithCondition
02:380 Quantity -- 10:qty_recv Field 16	
.....RULES.Mapping.MappingWithCondition { "ZA01 EQ "QD" "" }	
Parameters	Value
Expression	"ZA01 EQ "QD" ""

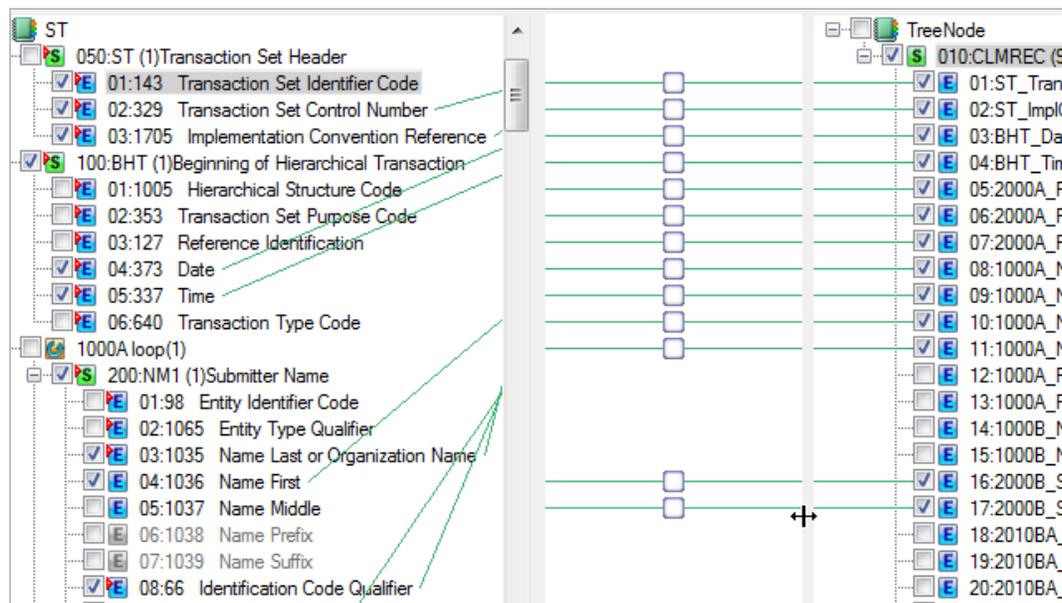
On the 03:qty_recd:

RULES.Mapping	MappingWithCondition
02:380 Quantity -- 12:qty_sold Field 18	
.....RULES.Mapping.MappingWithCondition { "ZA01 EQ "QR" "" }	
Parameters	Value
Expression	"ZA01 EQ "QR" ""

OutputControl

OutputControl lets you map multiple source segments to one target segment or record without having to use 2 pass. It saves up all the data in the designated range and then writes it out when it encounters a WriteTargetSegment rule.

It is designed for situations like this, where the ST, BHT, and NM1 all map to one big target segment. Without OutputControl or 2 pass, you will get one output record for each source segment that is mapped to the output record CLMREC.



To avoid having to use 2 pass, use OutputControl.

Format of Parameters

Value

Where:

Value

Choose one:

lock

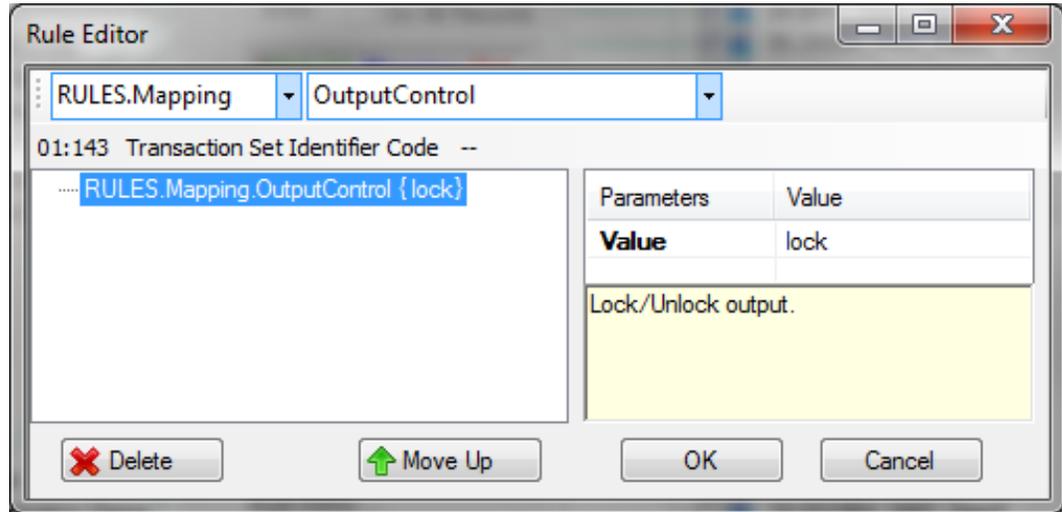
Mark the beginning of the first source segment that is mapped to the single output record.

unlock

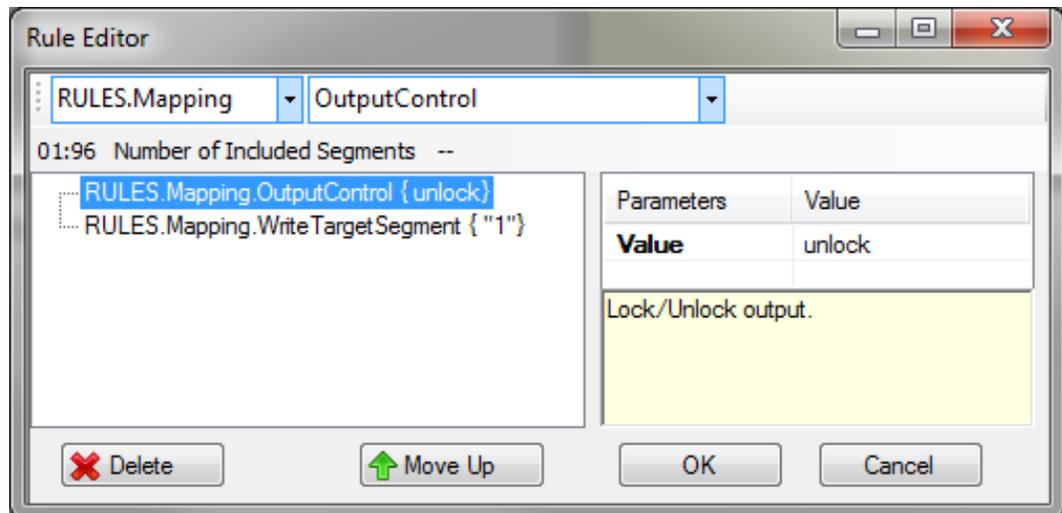
Mark last mapped source segment to the single output record.

Example

In the example above, put this rule the ST or one of its elements:



Put these rules on the SE:



The first one unlocks the range of data that is being saved, and the second writes it out to the target.

SetElementUsage

This rule sets the Current Element to Used or Not Used.

Format of Parameters

Option ElmID

Where:

Option One of these:

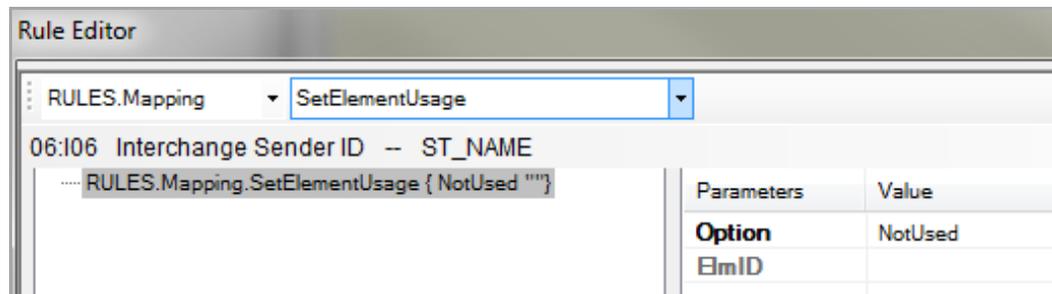
Used

Not Used

ElmID Element ID. Defaults to the current element.

Example

Put this rule on the ST or one of its elements.



SetQualifier

Future rule. Do not use.

WriteTargetSegment

After data has been saved up with OutputControl, WriteOutputSegment writes it out to the target. This is used when two or more source segments are mapped to one output segment.

Format of Parameters

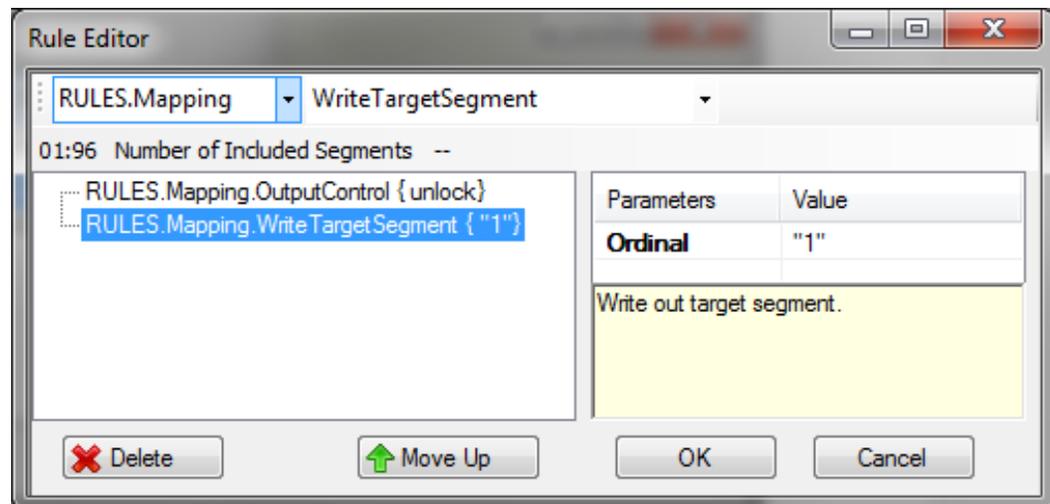
Ordinal

Where:

Ordinal The ordinal number of the target record. You can find the ordinal by clicking on the target record and looking in the Properties pane.

Example

This rule writes out all information that has been mapped in the OutputControl range.

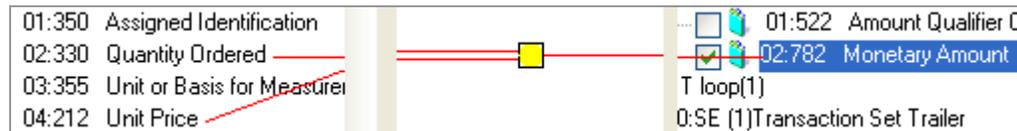


Rules.MultipleValues

MultipleValues

This rule applies to 2 Pass Mapping. See [One Pass and Two Pass Mapping](#) on page 24.

It performs an operation on two or more source values and places the result in a source element or field. This rule is only available if multiple source fields are mapped to one target field:



This rule requires 2 Pass Mapping. See [One Pass and Two Pass Mapping](#) on page 24.

Format of Parameters

Variable Variable Operand ResultVar

Where:

Variable The first variable.

Variable The second variable.

Operand What to do with the two variables (see Operand Table below).

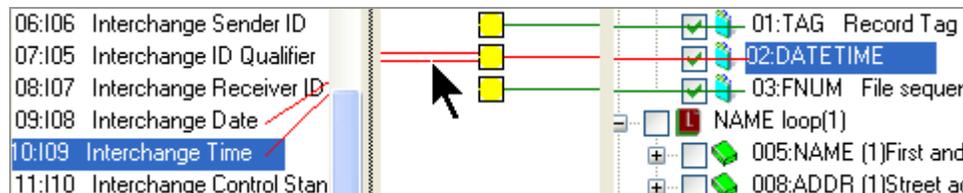
ResultVar Variable to hold the results.

Operand Table	
Operand	Explanation
ToEdiDateTime	Combines a date and a time from two source fields and places them in one variable in <i>yyyy/mm/dd – hh:mm:ss</i> format.
Multiply	Performs mathematical operation on two or more numbers in the source and places the result in one variable. Multiply and Add can work with two or more values. Divide and Subtract can work with two values.
Divide	
Add	
Subtract	
ToXmlDateTime	Puts a date and a time from two source fields into one variable in <i>yyyy-mm-ddT\mathbf{T}hh:mm:ss</i> format. T is a literal. This requires an XML target.

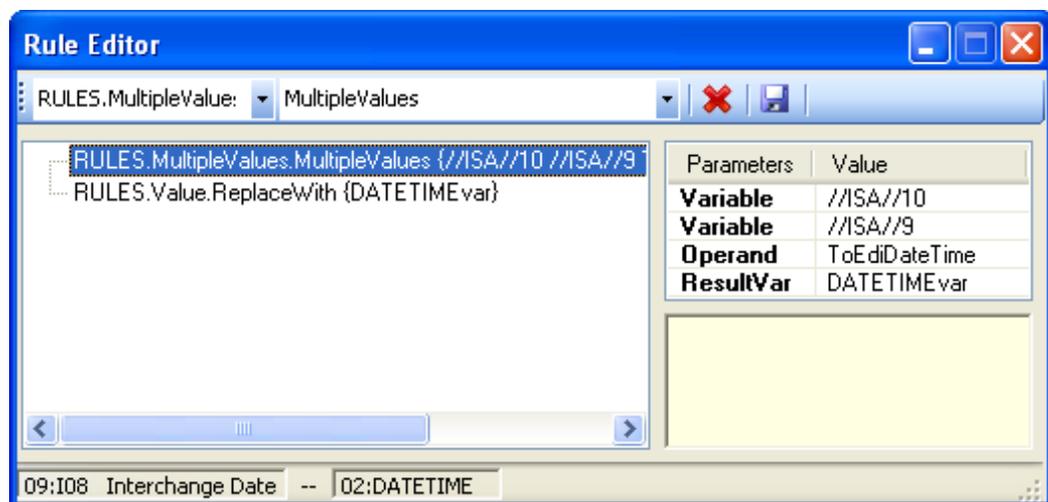
Example 1 – ToEdiDateTime

This combines the source's Date and Time elements into one target field.

First, map the two source elements into the one target field:



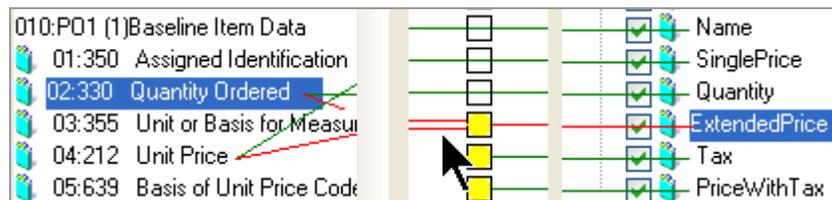
Then set up this MultipleValues rule to combine them into the **ToEdiDateTime** format and place the result in variable **DATETIMEvar**. The first two parameters are pre-filled based on the mapping. The target is XML data.



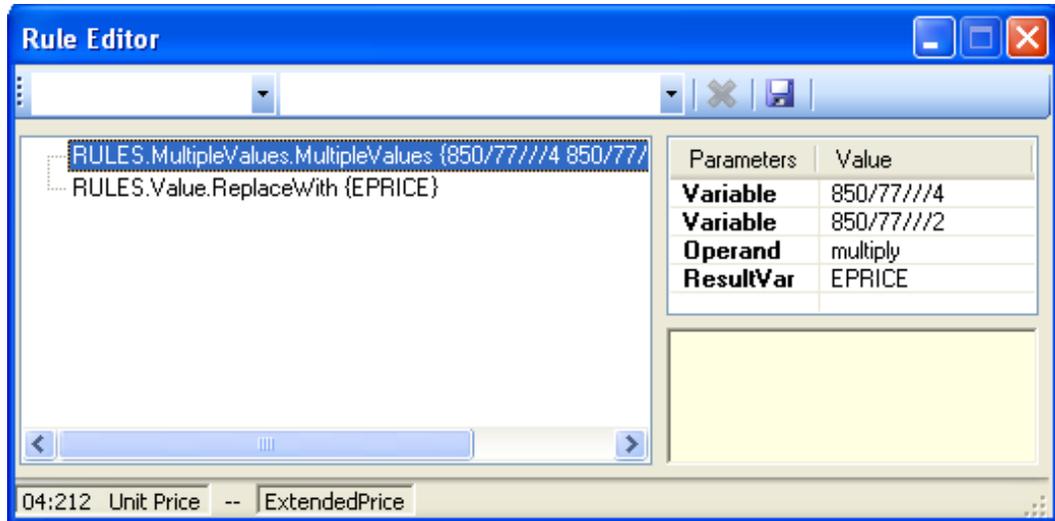
Example 2 – Multiply

This multiplies two source values and places the result in variable **EPRICE**, which is then used for the target value.

First, map the **Quantity Ordered** and **Unit Price** from source into target's **ExtendedPrice** field.

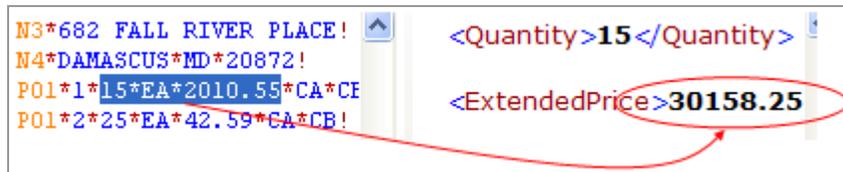


Then set up this MultipleValues rule to multiply the source values and place the result in variable **EPRICE**. The first two parameters are pre-filled based on the mapping.



Finally, choose Settings | Target Guideline Settings ... | 2 Pass Mapping.

Example result:



Rules.NewRecord

This rule creates a new record when using these translations:

- EDI – EDI
- EDI – Flat File

Create

This rule creates a new target EDI or flat file record.

The target record/segment must:

- Have a NewRecord Create rule.
- Be mandatory in the target.
- Not be the last segment/record in the output.

Format of Parameters

Value

Where:

Value The contents of the target record/segment without a segment or record terminator. If it contains spaces, surround it with double quotes. This is a literal, not a variable.

Example

The rule is on the segment itself, the segment is not mapped, and other items are mapped below it.

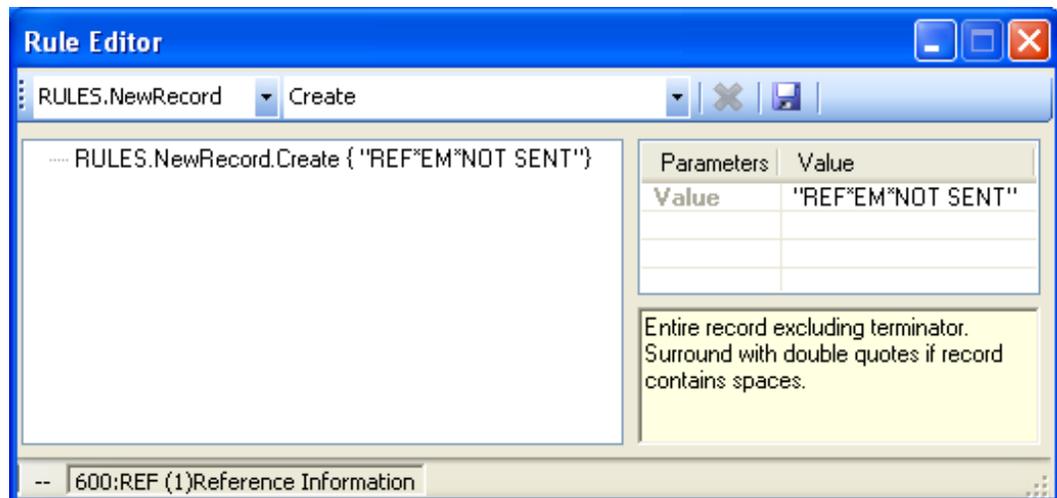
Mapping option 1: This always creates the segment specified in the rule.



Mapping option 2: This uses the mapped data from the source, if the segment is present. Otherwise, it uses the segment specified in the rule.



For example, this rule creates the segment `REF*EM*NOT SENT`:



CreateANewSegment

This rule creates a new segment if there is no data in the source. It does not have to exist in the target.

Format of Parameters

Value

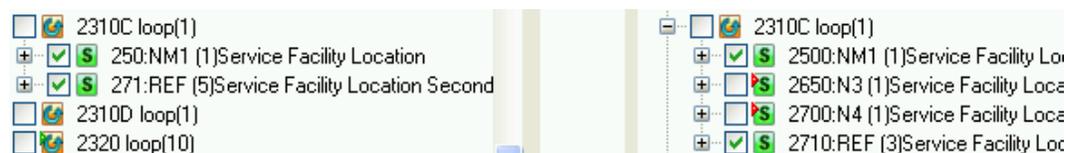
Where:

Value Literals surrounded with double quotes, reserved words surrounded with percent signs, and variables. Each item is separated with a space.

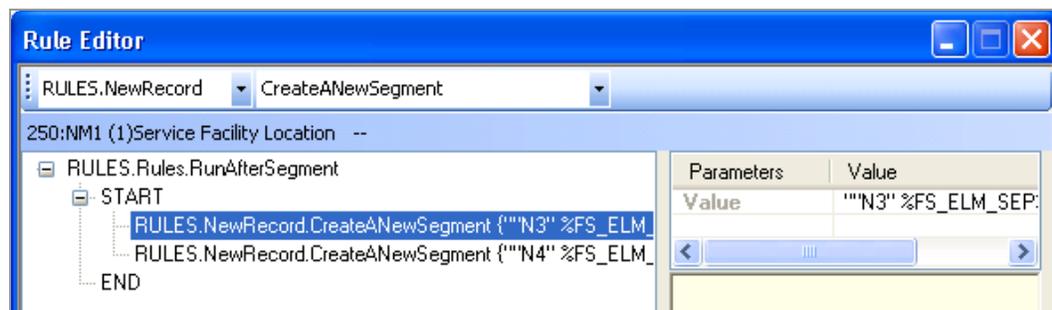
The entire line is surrounded with double quotes. This means that there will be two consecutive double quotes at the beginning if the first value is a literal, and there will be two consecutive double quotes at the end if the last value is a literal.

Example

The target has mandatory N3 and N4 segments that do not exist in the source. We will use rules to create them.



This is a source rule on the NM1 segment. It creates two segments after the current segment.



The Parameters area defines the new segment with a combination of literals, variables, and reserved words. Each is separated by exactly one space:

Parameters	Value
Value	""N4" %FS_ELM_SEP% "CITY" %FS_ELM_SEP% "ZZ" %FS_ELM_SEP% "11111""

Output will contain the two new segments, inserted after the NM1. In this example, another rule converted the FA at NM1 to 77.

NM1*FA*1*JOHNSON-1-20-22A*BARBARA	NM1*77*1*JOHNSON-1
LX*1~	N3*ADDRESS~
SV3*AD*DO120:12:AA:A2:13*120.00*J	N4*CI*ZZ*11111~
HL*7*1*22*0~	LX*1~

CreateANewSegmentWithCondition

This source or target rule on a segment specifies that, if certain conditions are met, a new segment is created after the current one. This is similar to [CreateANewSegment](#) on page 193.

Format of Parameters

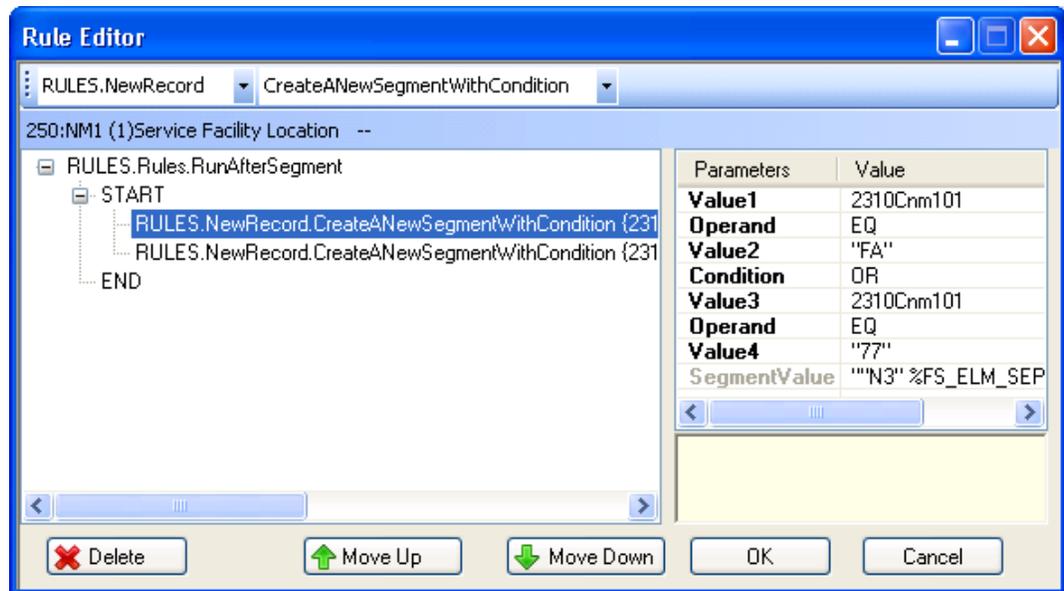
Value1 *Operand* *Value2* *Condition* *Value3* *Operand* *Value4*
SegmentValue

Where:

<i>Value1</i>	First value to compare.
<i>Operand</i>	Relational operand to use: EQ, NE, GT, GE, LT, LE.
<i>Value2</i>	Second value to compare.
<i>Condition</i>	Logical operand to use AND/OR
<i>Value3</i>	First value to compare.
<i>Operand</i>	Relational operand to use. EQ, NE, GT, GE, LT, LE.
<i>Value4</i>	Second value to compare.
<i>SegmentValue</i>	Literals surrounded with double quotes, reserved words surrounded with percent signs, and variables. Each item is separated with a space.

Example

If either condition is true, the new segment is created after the one on which this rule is placed.



Parameters area:

Parameters	Value
Value1	HL04
Operand	EQ
Value2	0
Condition	OR
Value3	HL04
Operand	EQ
Value4	""
Segment...	""DMG" %FS_ELM_SEP% "CITY" %FS_ELM_SEP% "STATE" %FS_ELM_SEP% "ZIP" ""

CreateByTargetOrdinal

This rule lets you create a new record containing the values that you have assigned to its elements or fields. You use the ordinal number of the target segment to specify which segment or record you want to create. If the segment to be created is an enveloping segment, it will not have an ordinal, so you can use the segment tag instead.

Format of Parameters

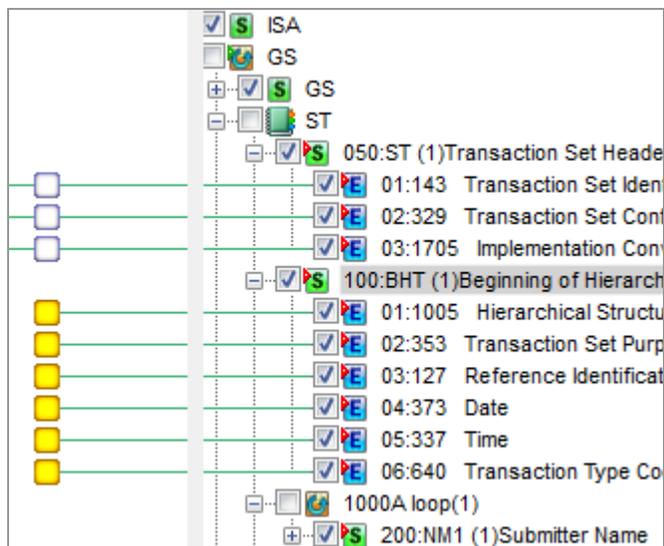
Value

Where:

Value Ordinal number of segment/record to be created in the output. There are no ordinal numbers for interchange and group enveloping but you can use the segment tag instead.

Example

Assume that this target BHT segment does not have any source data that can map to it.



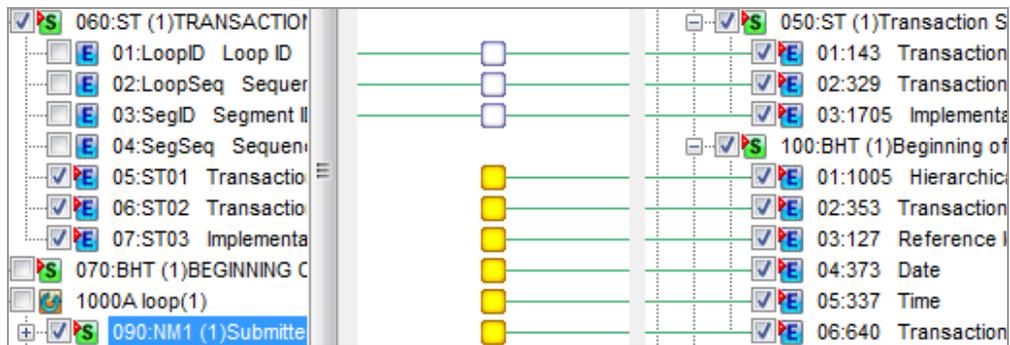
You can assign data to its elements with ReplaceWith rules but it still will not appear in the output because none of its elements have been mapped.

To get around this:

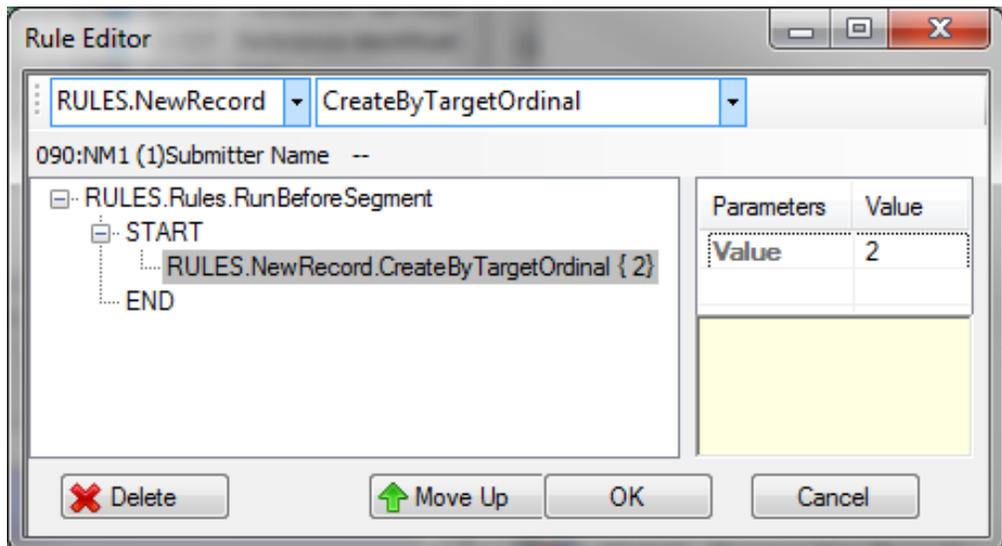
1. We use ReplaceWith rules on the elements that we want (in this case, all of them).
2. We look up the BHT's ordinal in its properties pane:

Segment ID [BHT]	
Description	Beginning of Hi
Purpose	To define the b transaction set purpose and ref time
Ordinal	2
Repeat	1

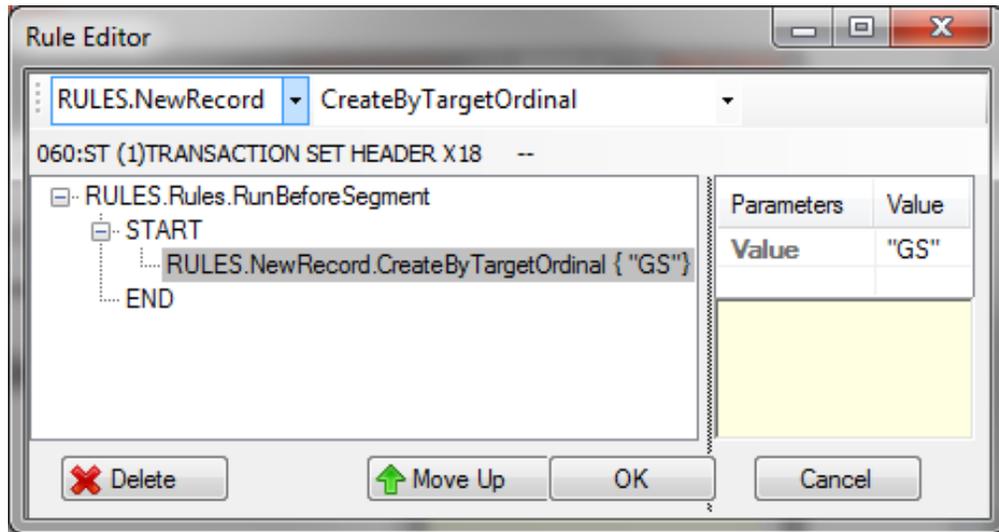
3. We create a source rule on the next segment/record, the NM1:



Here is the rule:



If we had been trying to create the GS segment, using a similar strategy, the source ST segment would have a rule like this, where the ordinal is replaced with the segment tag:



CreateInMemoryByTargetOrdinal

This rule creates a new record in memory using target ordinal number. This is similar to [CreateByTargetOrdinal](#) on page 196 but allows for the creation of empty segments and entry of a memory reference name.

Format of Parameters

Value *Option* *Name*

Where:

Value Target ordinal number or multiple target ordinal numbers separated by a comma.

Option Should empty segments be created? Select Y or N.

Name Name of memory reference.

CreateLoopBy2DArray

This source rule on a source segment creates a new target segment for each row in the specified array. It then clears out all data from the array.

Format of Parameters

Name Variable

Where:

Name The array's name.

Variable List of variables that are to be incremented after use. Separate variables with commas and surround the entire list with double quotes.

Separator Separator between data values in the new segment. Only use if all separators are the same and the array does not have separators added. Please see Example 2 below.

Example 1

This example creates a new target segment for each row in the PriorAuth array. No variables are incremented by the rule.

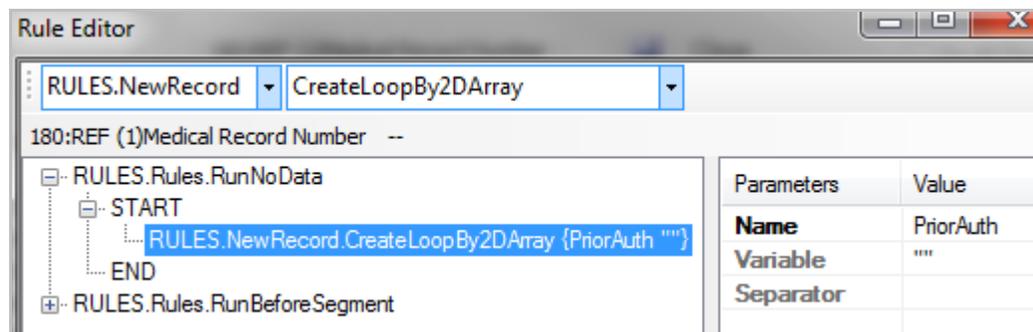
An array is created earlier with InsertToArray.

```

... RULES.Variable.Set { %Current_Element% 2300REF01PriorAuth }
... RULES.String.Compare { %Current_Element% EQ "9F" }
... START
... RULES.2DArray.InsertToArray { PriorAuth 0 0 "REF" }
... RULES.2DArray.InsertToArray { PriorAuth 0 1 %FS_ELM_SEP% }
... RULES.2DArray.InsertToArray { PriorAuth 0 2 %Current_Element% }
... END
... RULES.String.Compare { %Current_Element% EQ "G1" }
... START
... RULES.2DArray.InsertToArray { PriorAuth 1 0 "REF" }
... RULES.2DArray.InsertToArray { PriorAuth 1 1 %FS_ELM_SEP% }
... RULES.2DArray.InsertToArray { PriorAuth 1 2 %Current_Element% }
... END

```

It is then written to the output:



Example 2

Since the segment created with the PriorAuth array uses the same delimiters throughout, we can omit them from the array:

```
... RULES.Variable.Set { %Current_Element% 2300REF01PriorAuth}
- RULES.String.Compare { %Current_Element% EQ "9F"}
  - START
    - RULES.2DArray.InsertToArray { PriorAuth 0 0 "REF"}
    - RULES.2DArray.InsertToArray { PriorAuth 0 1 %Current_Element%}
  - END
- RULES.String.Compare { %Current_Element% EQ "G1"}
  - START
    - RULES.2DArray.InsertToArray { PriorAuth 1 0 "REF"}
    - RULES.2DArray.InsertToArray { PriorAuth 1 1 %Current_Element%}
  - END
```

... and use the Separator parameter with CreateLoopBy2DArray:

Parameters	Value
Name	PriorAuth
Variable	""
Separator	.

Example 3

This example creates a new target segment for each row in the PriorAuth array, whether the current segment is present in the data or not. Two variables are incremented by the rule. Since a separator is not specified, delimiters must be included in the array.

Parameters	Value
Name	PriorAuth
Variable	"RefVar, ClmRefVar"
Separator	

Please see the example in [RunNoData](#) on page 207 for a more complex example.

Example 4

If all delimiters are the same in the segment being written, you do not have to add them to the array. Use the Separator parameter:

```
[RULES.String.Compare { %Current_Element% EQ "BK"} START
RULES.2DArray.InsertToArray {2300_CLM 0 0 "HI"}
RULES.2DArray.InsertToArray {2300_CLM 0 2 %Current_Element%}END]
```

Example 5

This rule on the first source element inserts a record before the ISA:

```
[RULES.2DArray.InsertARow { DDD 0 "This is a header row"}]
```

CreateMemoryLayout

Future rule. Do not use.

CreateXmlTargetBy2DArray

This rule creates XML output and then clears all data from the array. For more information on this rule, contact TIBCO Foresight Support.

Format of Parameters

Name *Repeat Position*

Where:

Name The name of the array.

Repeat Position Position of repeating point (separated by commas).

DeleteSegmentValues

This rule deletes all element values from a segment.

Format of Parameters

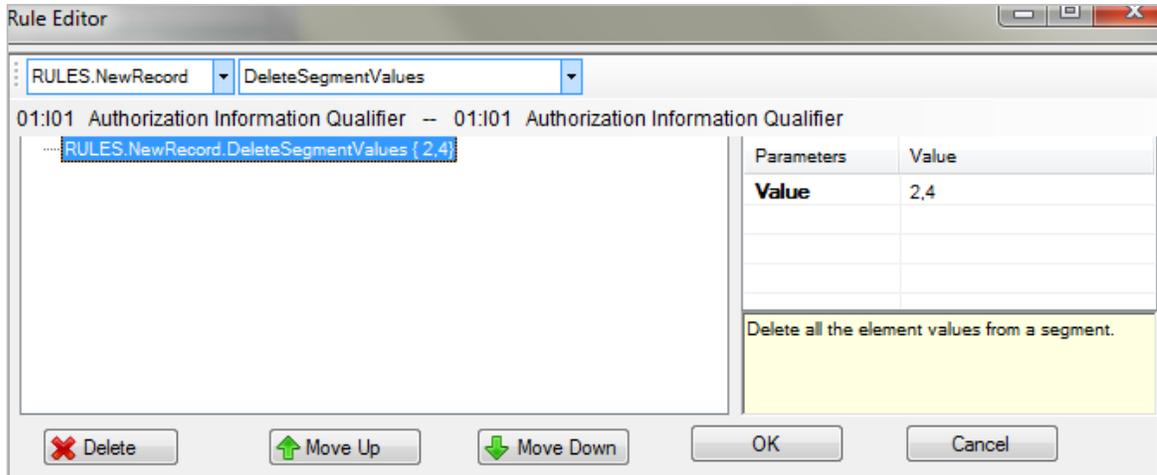
Value

Where:

Value Target segment ordinal number or multiple target segment ordinal numbers separated by a comma.

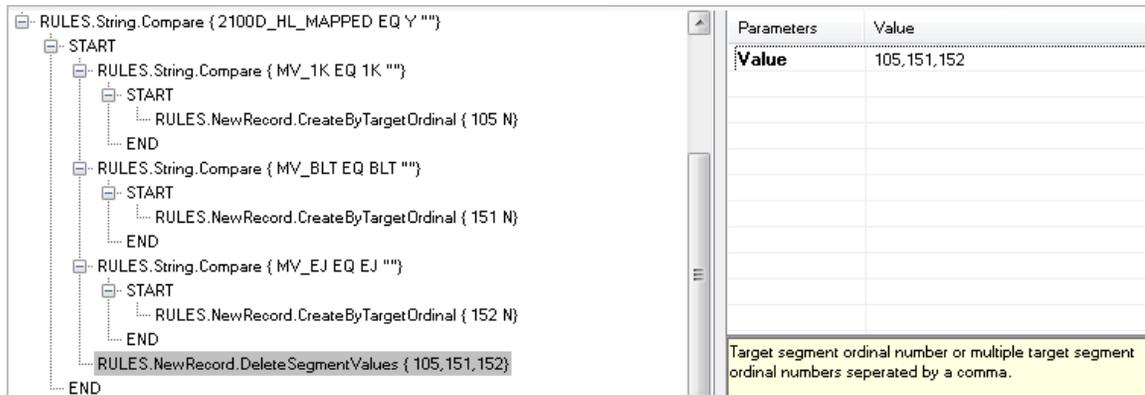
Example 1

This example deletes all values on target segments BPR(2) and TRN(4).



Example 2

This example shows multiple String.Compare rules being used to qualify mapping data per Target Ordinal Number. A NewRecord.DeleteSegmentValues business rule is added at the end to clear the ordinals.



InsertRecordBy2DArray

Future rule for Database targets only. Do not use.

OutputMemoryLayout

Future rule for positional flat file. Do not use.

PostCreate

This rule creates new SE, GE, or IEA records if there is no data in the source. Put a PostCreate rule on each element that should contain output data in these trailing enveloping segments.

You must select Create Trailer from the Target settings to make this work.

Format of Parameters

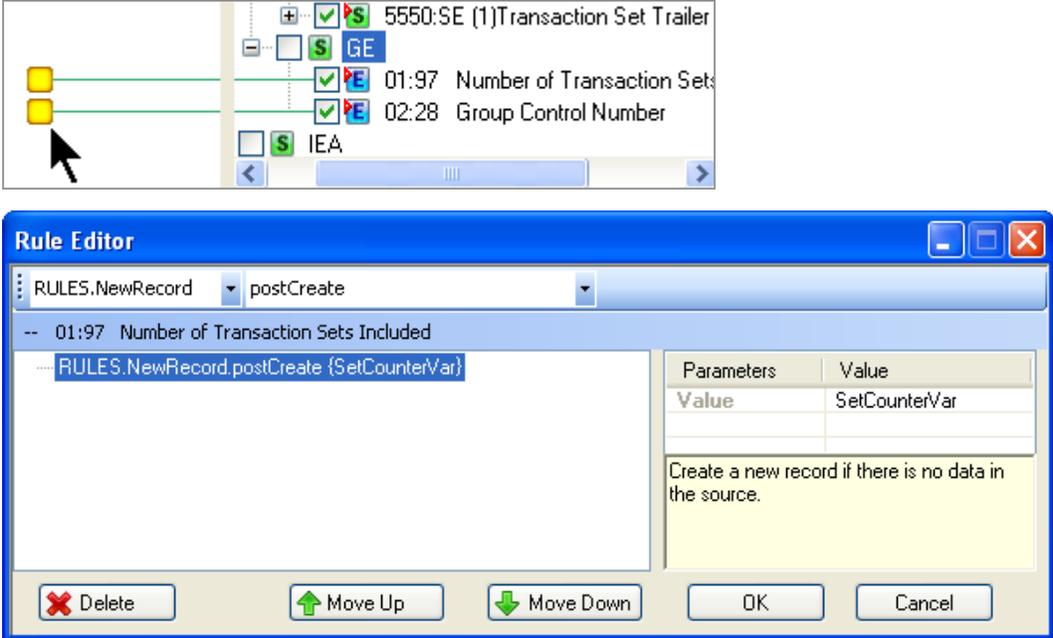
Value

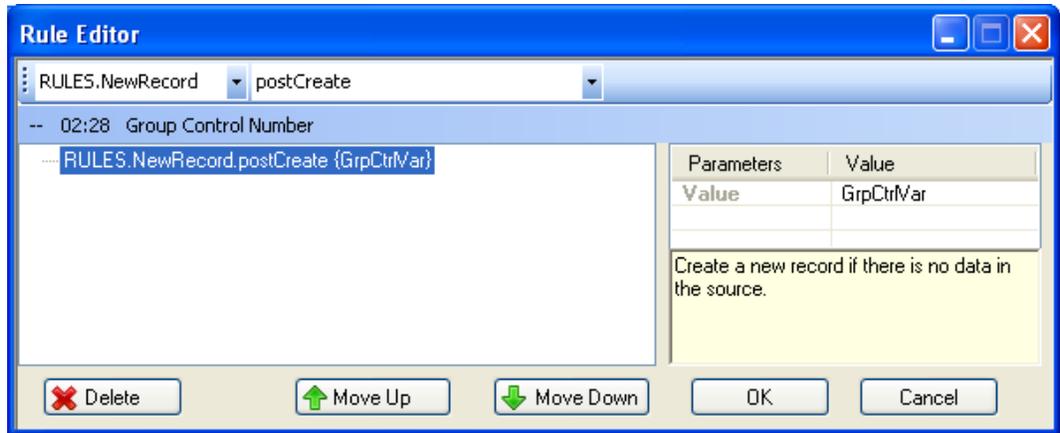
Where:

Value The entire record excluding terminator, surrounded with double quotes if the record contains spaces.

Examples

This example creates a new target GE segment when the mapped value does not exist. **Settings | Target Guideline Settings ... | Create Trailers** is selected. **PostCreate** rules are on the target GE01 and GE02.





Rules.Rules

RunAfterComposite

This rule specifies that a sub-rule or block of rules should run after the current composite has been written out. It is a source rule on a segment. It is similar to [RunAfterLoop](#) on page 205.

RunAfterLoop

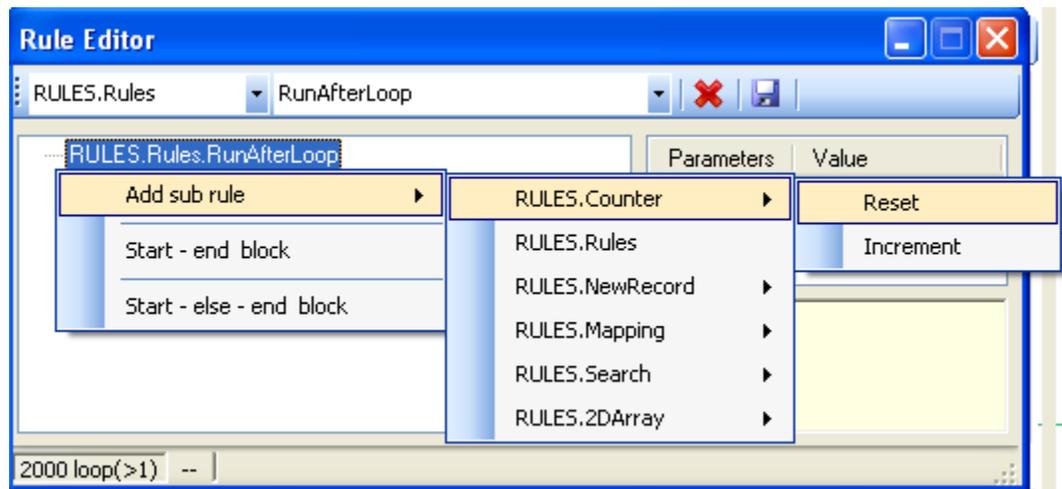
This rule specifies that a sub-rule or block of rules should run after the current loop has been written out. It is a source rule on the loop header.

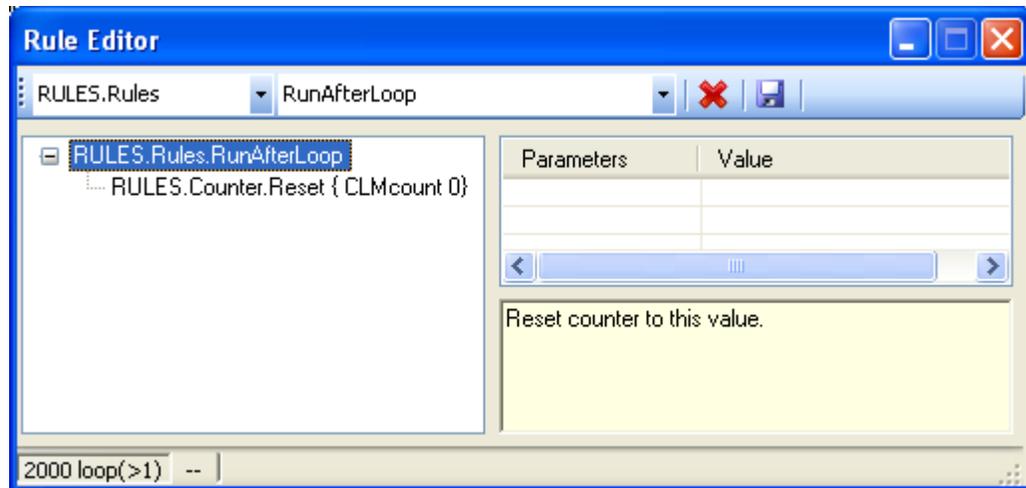
Format of Parameters

None.

Example

This example shows a rule applied to a loop header. It specifies that after the loop is written out, the counter should be reset.





RunAfterSegment

This rule specifies that a sub-rule or block of rules should run after the current segment has been written out. It is a source rule on a segment. It is similar to [RunAfterLoop](#) on page 205.

RunAfterSegmentExit

This rule specifies that a sub-rule or block of rules should run after the current segment, including all repeating segments, has been written out. It is a source rule on a segment. It is similar to [RunAfterLoop](#) on page 205.

RunAtEndOfFile

This rule specifies that a block of rules should run at the end of the file. It is a source rule on any segment or loop. It is similar to [RunAfterLoop](#) on page 205.

RunBeforeFile

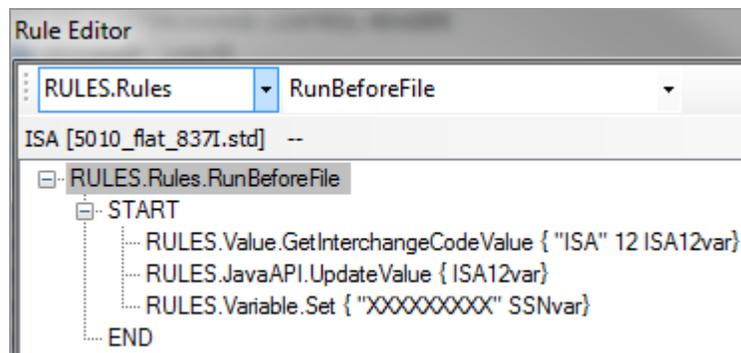
This rule specifies that a block of rules should run before mapping starts on the file. It is a source rule on the first line in the source.

Format of Parameters

None.

Example

This example shows a block of rules that start before mapping. The first one puts the value from the target ISA12 into a variable, the second one sends this value to the calling program, and the third sets up a variable for use later in the translation.



RunBeforeLoop

This rule specifies that a sub-rule or block of rules should run before the current loop has been written out. It is a source rule on a loop. It is similar to [RunBeforeFile](#) on page 207.

RunBeforeSegment

This rule specifies that a sub-rule or block of rules should run before the current segment has been written out. It is a source rule on a segment. It is similar to [RunBeforeFile](#) on page 207.

RunNoData

This rule is to run if the current segment has no data. This source rule requires a sub-rule.

Format of Parameters

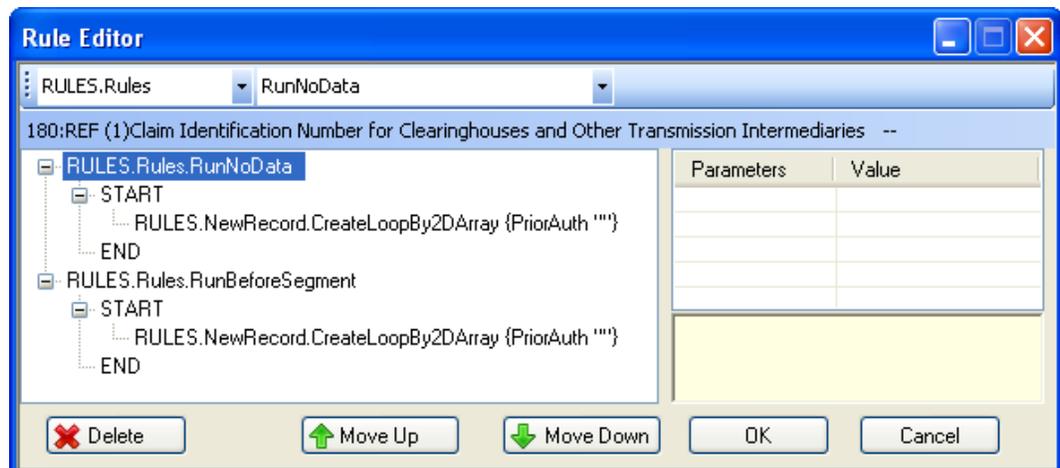
None.

Example

This set of rules on a source segment specifies rules to run regardless of whether the current segment contains data.

The first set of rules (RunNoData) executes if the current segment doesn't exist in the source. It creates target segments from the contents of the PriorAuth array.

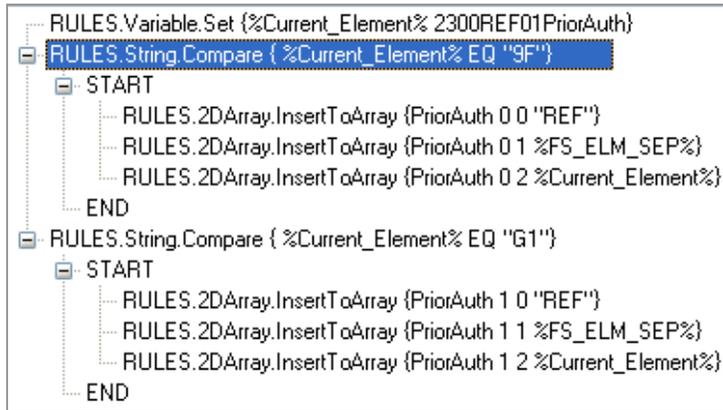
The second set of rules (RunBeforeSegment) executes if the current segment does exist. It specifies the same action.



An example scenario when you would use this rule is this REF segment in the 837D step-up map. The source REF allows two code values. One code value maps the segment to the Referral Number REF in the target. The other maps it to the Prior Authorization REF in the target. The source REF can repeat 2 times.



Each element in the Prior Authorization REF contained rules like these:



With the values captured, the RunNoData and RunBeforeSegment rules on the next segment (the Claim Identification Number for Clearinghouse REF) places the entire contents of the array in the target. This would be the two REF segments.

Notice that there is no mapping.

RunOnLoopExit

This rule on a source loop header runs after all iterations of the current loop have completed. It requires a sub-rule.

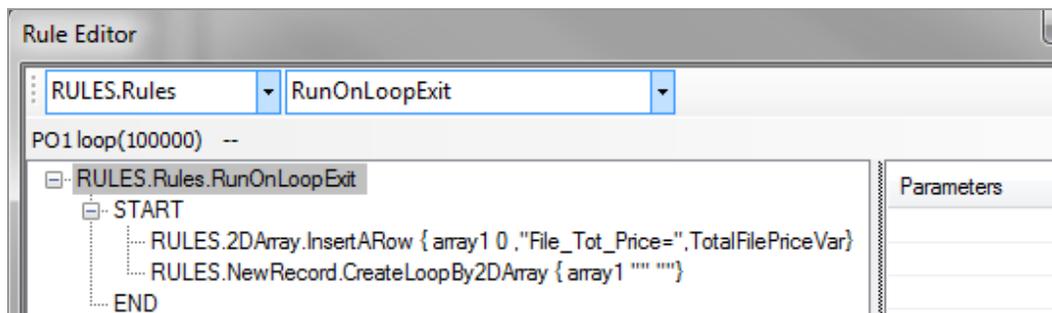
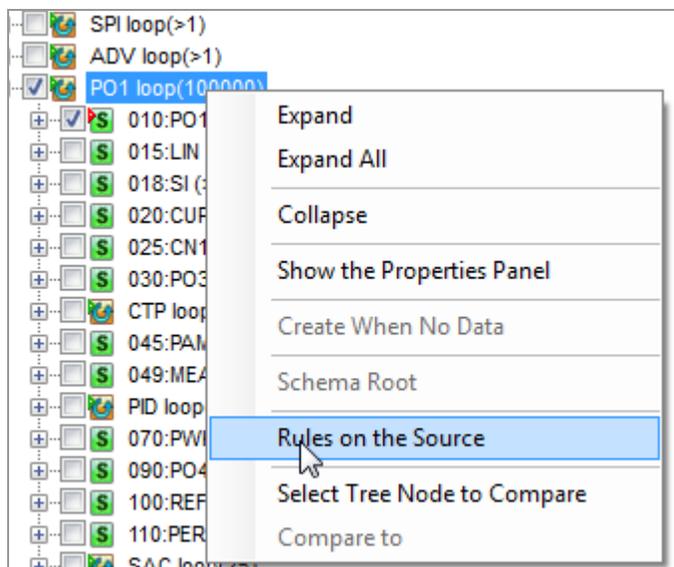
This rule is not available for XML source.

Format of Parameters

None.

Example

This example shows a rule applied to a source loop header. It specifies that after all iterations of the loop have been written out, a record is written out.



The record would look something like this:

```
File_Tot_Price=1250.00
```

Rules.Search

This rule performs a forward search for a segment, element, or subelement.

Note: Rules.Search can be used **only on the source** and the source cannot be an XML document.

Scope of Search

The scope of the search depends on where the search rule appears. It searches from the current location to the end of the current loop, including subordinate loops.

Examples

Location of search rule	Searches this area
ST segment or any segment before the first loop	All segments from the location of the rule to the end of the transaction
Segment within the 1000A loop	All segments within the 1000A loop
2000A HL segment	All segments and loops within the 2000A or its subordinate loops
2000B SBR segment	All segments and loops from the SBR down to the end of the 2000B loop

Element

This rule performs a forward search to find an element or subelement and captures its value if it exists.

Format of Parameters

SegmentID *Ordinal* *ElementPosition* *ResultVar*

Where:

SegmentID ID of segment to be searched, a literal in double quotes. The segment must be in the same loop as the rule, or in a subordinate of the loop. Please see [Rules.Search](#), Scope of Search.

Ordinal Ordinal number of the segment. You can find this on the segment's properties pane.

ElementPosition Position to be searched within the element using the format *Element-subelement*. Remember that indexes start with 0. For example, if you are searching for the REF02, this value would be at 1 or the sub-element (1-3) since the index starts at 0.

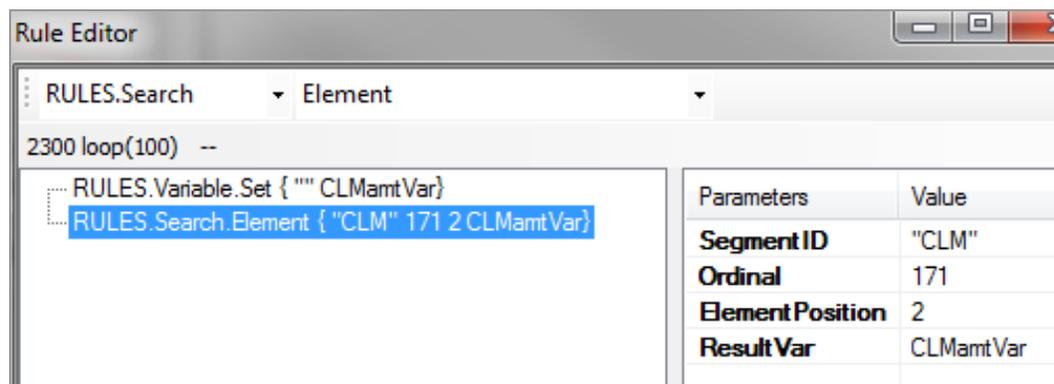
Examples:

Second element 2
Second element (a composite) - first subelement 2-1

ResultVar Variable to hold the value, if found.

Example

This example clears the variable CLMamtVar. It then searches the first CLM segment at ordinal 171 in the current loop or its subordinate loops. If it has an element 2, that value is placed in CLMamtVar.



Parameters	Value
Segment ID	"CLM"
Ordinal	171
Element Position	2
Result Var	CLMamtVar

ElementInCurrentSegment

This rule performs a forward search to find an element/sub-element in the current segment and capture its value if it exists. It is just like the rule [Element](#), but omits the need to specify a segment ID and ordinal; it will search the current segment only. Refer to [Element](#) on page 212 for additional information.

Format of Parameters

ElementPosition ResultVar

Segment

This rule performs a forward search on a segment. It returns a 1 if the segment is found. You can have it put the segment in a variable if it is found. 2 pass is not required.

Format of Parameters

SegmentID Ordinal ResultVar SegmentVar

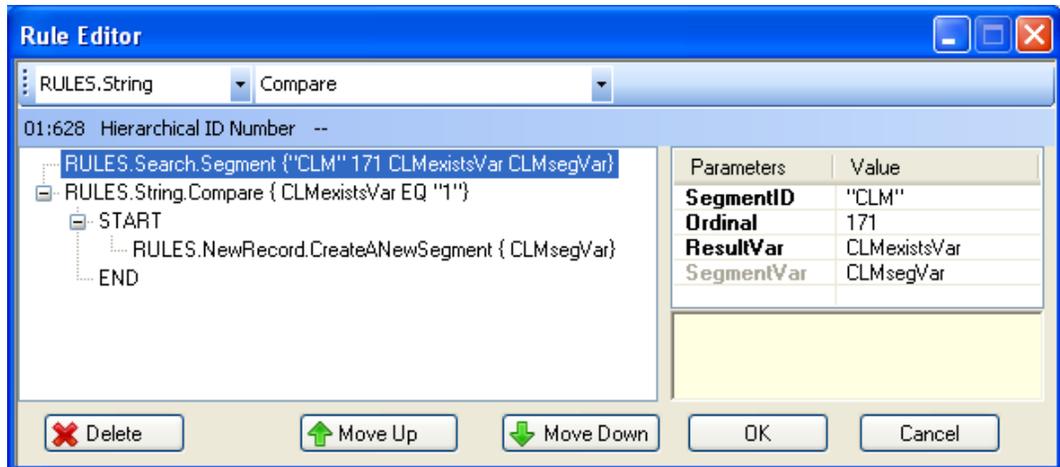
Where:

<i>SegmentID</i>	ID of segment to be searched. The segment must be in the current loop or its subloop. This can be a variable, %Current_Element%, or a literal in double quotes. See Rules.Search , Scope of Search on page 211.
<i>Ordinal</i>	Ordinal number of segment. You can find this on the segment's properties pane.
<i>ResultVar</i>	Variable that holds 1 if the segment exists.
<i>SegmentVar</i>	Variable to hold the segment, if the segment exists.

Example

This example does a forward search to the CLM segment at ordinal 171. If it exists, a 1 is placed in CLMexistsVar and the CLM segment itself is placed in CLMsegVar.

The next rule checks for a 1 in CLMexistsVar and, if found, writes out the CLM segment at the current location.



SegmentsInLoop

This rule searches a loop for all instances of a certain segment and puts them in an array. It should appear before the segment that it is capturing. 2-Pass is not needed for this rule.

Format of Parameters

SegmentID Ordinal ResultVar ArrayName

Where:

SegmentID ID of segment to be put in array. The segment must be in the current loop or its subloop. This can be a variable or a literal in double quotes. See [Rules.Search](#), Scope of Search on page 211.

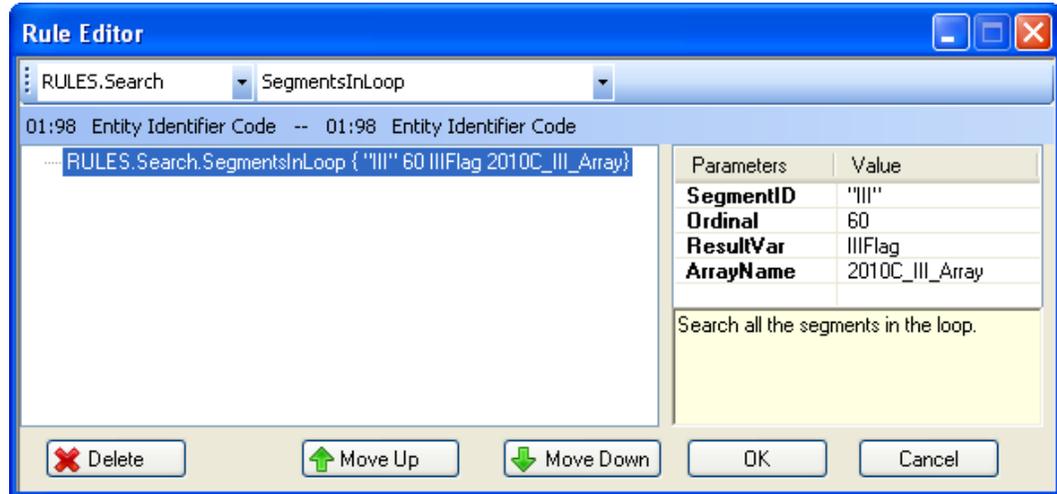
Ordinal Ordinal number of segment. You can find this on the segment's properties pane.

ResultVar Name of a variable that will hold 1 if the segment exists.

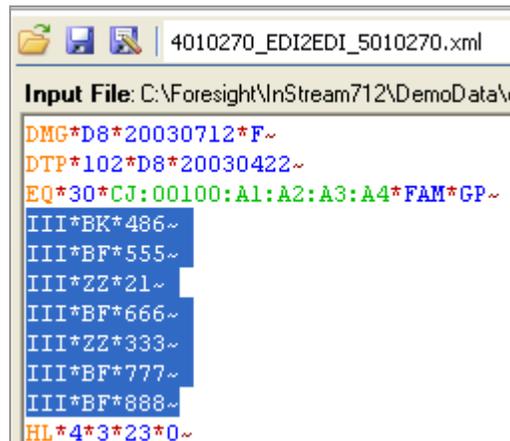
ArrayName Name of a new array to hold the segments. This array name must be a literal, optionally surrounded by double quotes. The rule will automatically create the array.

Example

This rule searches down through the current loop and finds any III segments, located at ordinal 60. You can see a segment's ordinal in its properties pane. It puts a 1 in IIIFlag if a III segment is found. The III segments go in array 2010C_III_Array. This will contain all instances of III in all iterations of the loop.



Assume that the data contained these III segments:



Array 2010C_III_Array would contain:

```

1, 1, III, BK, 486
1, 2, III, BF, 555
1, 3, III, ZZ, 21
1, 4, III, BF, 666
1, 5, III, ZZ, 333
1, 6, III, BF, 777
1, 7, III, BF, 888
↑ ↑ ↑ ↑ ↑
① ② ③ ④ ⑤
  
```

Where:

- ① Iteration of the loop.
- ② Iteration of the segment.
- ③ Segment tag.
- ④ First value in segment.
- ⑤ Second value in segment.

Rules.String

AddXMLEscapeCodes

Replaces special characters with XML escape codes.

Special characters (e.g., <, >, and !) have designated meaning in XML. To use them as characters in XML, you must “escape” (change) them into XML escape codes. If special characters are not changed into XML escape codes, the mapping may not produce XML output.

Examples of special characters and their XML escape codes:

```
< = &lt;;  
> = &gt;;  
' = &apos;;
```

This rule can be used on the source or target and is useful when translating to EDI-to-XML using 2Darrays.

Format of Parameters

Value NewValue

Where:

Value Value to be checked for XML special characters. This is a literal in double quotes or a variable containing the value.

NewValue The resulting variable.

Example

Use this rule when special characters could be sent as part of a Subscriber Last Name, such as O'BRIEN. This rule adds the XML escape code for the apostrophe in the name. For example, O'BRIEN, would be converted to O&apos ;BRIEN and stored in the variable specified in NewValue.

The screenshot shows the 'Rule Editor' window. At the top, there are two dropdown menus: 'RULES.String' and 'AddXMLEscapeCodes'. Below these, the rule name is '03:1035 Name Last or Organization Name --'. The rule definition is 'RULES.String AddXMLEscapeCodes (%Current_Element% 2100CNM103)'. To the right of the rule definition is a table with two columns: 'Parameters' and 'Value'. The table contains two rows: 'Value' with the value '%Current_Element%' and 'NewValue' with the value '2100CNM103'.

Parameters	Value
Value	%Current_Element%
NewValue	2100CNM103

Append

Concatenates two or more values into a variable.

Format of Parameters

“InputValues” ResultVar

Where:

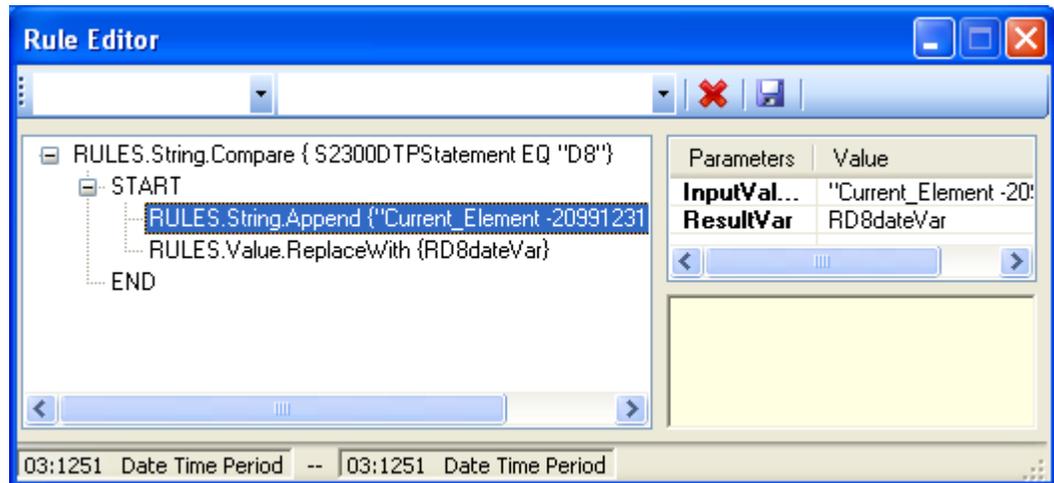
InputValues A list of two or more values, separated by spaces. This can consist of literals, variables, and reserved variables. The entire list of values is enclosed in double quotes. Spaces are separators so you cannot have spaces in your own strings.

ResultVar Variable to contain the result.

Example

This rule builds a date range by concatenating the current value, a hyphen, and a literal and storing it in variable RD8dateVar. It then uses that in the output.

This rule assumes that the date qualifier was previously captured in variable S2300DTPStatement.



Parameters area:

Parameters	Value
InputValues	"Current_Element -20991231"
ResultVar	RD8dateVar

Example segment before translation: DTP*434*D8*20030212~

Same segment after translation: DTP*434*RD8*20030212-20991231~

Compare

Compares two values as strings based on the operand. It requires a sub-rule to execute an action if the comparison is true.

Format of Parameters

Value1 *Operand* *Value2* *CaseSensitive*

Where:

Value1 Contains the first value to be compared.

Operand Choose how to compare the two values.

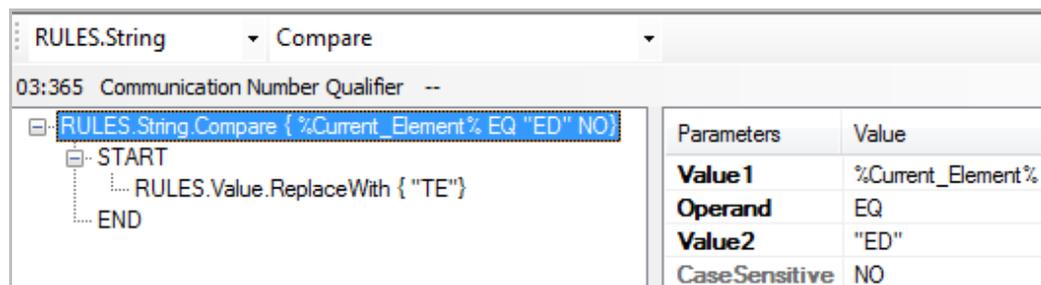
Value2 Contains the second value to be compared.

CaseSensitive Should the comparison consider case? Default is yes.

Do not use these uppercase key words in START-END rules, even if they are embedded in other words: START ELSE END. For example, do not use a variable called **SENDERID** since it contains END. You may call it SenderID, however.

Example 1

These rules check to see if the current element contains ED. If true, it uses the literal value TE in the output.



The screenshot shows a rule editor interface. The rule is named 'RULES.String.Compare' and is set to 'Compare'. The parameters are: Value1: %Current_Element%, Operand: EQ, Value2: "ED", and CaseSensitive: NO. The rule body contains a START block with a sub-rule 'RULES.Value.ReplaceWith {"TE"}' and an END block.

Parameters	Value
Value1	%Current_Element%
Operand	EQ
Value2	"ED"
CaseSensitive	NO

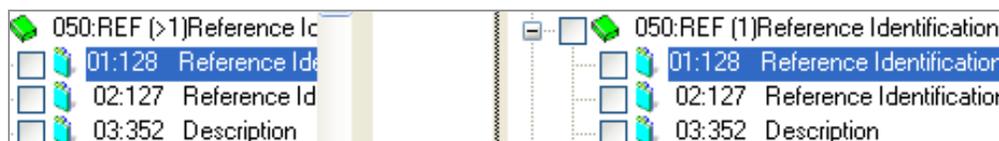
Example 2 – Extended Example

Our plan:

- If the source REF01 is LU, we want to use our own variation of the REF segment in the output:

REF*ZZ*UNSUPPORTED DATA~

- Otherwise, we want to map the REF01 and REF02 from the source.



The screenshot shows a data mapping tool interface. On the left, the source data is shown with elements: 050:REF (>1)Reference Id, 01:128 Reference Id, 02:127 Reference Id, and 03:352 Description. On the right, the target data is shown with elements: 050:REF (1)Reference Identification, 01:128 Reference Identification, 02:127 Reference Identification, and 03:352 Description. The mapping shows that the source REF01 is mapped to the target REF01, and the source REF02 is mapped to the target REF02.

To do this:

1. See if the value on the REF01 is LU. If so, replace it with ZZ. Otherwise, use the value in the output.

Parameters	Value
Value 1	%Current_Element%
Operand	EQ
Value 2	"LU"
Case Sensitive	NO

2. In the same way, replace the value in the REF02 if the REF01 was ZZ:

Parameters	Value
Value 1	REF01var
Operand	EQ
Value 2	"LU"
Case Sensitive	NO

The REF segment will only be generated if at least one of its elements is mapped:

CompareC

Performs two comparisons based on the operands. It requires a sub-rule to execute an action if the comparison is true.

Format of Parameters

Value1 Operand Value2 Condition Value3 Operand Value4 CaseSensitive

Where:

<i>Value1</i>	Contains the first value to be compared.
<i>Operand</i>	Choose how to compare the two values.
<i>Value2</i>	Contains a value to be compared to Value1.
<i>Condition</i>	AND or OR.
<i>Value3</i>	Contains another value to be compared.
<i>Operand</i>	Choose how to compare the two values.
<i>Value4</i>	Contains a value to be compared to Value4.
<i>CaseSensitive</i>	Should the comparison consider case? Default is yes.

Example

This rule checks to see if REF01var contains LU and REF02length contains a value less than 10. If *both* are true (AND), it executes the sub-rule that puts “BAD DATA” into the output. Otherwise, it uses the current value. Since CaseSensitive was not set, the default is YES – the comparison is case sensitive.

The screenshot shows a rule editor interface. At the top, there are two dropdown menus: the first is set to 'RULES.String' and the second is set to 'CompareC'. Below these, the rule text is displayed in a tree view. The selected rule is '02:127 Reference Identification -- 02:127 Reference Identification'. The rule body contains the following code:

```
RULES.Variable.Set { %Current_Element% REF02var}
RULES.String.CompareC { REF01var EQ "LU" AND REF02length LT "10"
  START
  RULES.Value.ReplaceWith { "BAD DATA"}
  ELSE
  RULES.Value.ReplaceWith { REF02var}
  END
```

To the right of the rule text is a 'Parameters' table with the following data:

Parameters	Value
Value1	REF01var
Operand	EQ
Value2	"LU"
Condition	AND
Value3	REF02length
Operand	LT
Value4	"10"
CaseSensitive	""

CompareX

Performs complex comparisons based on the operands. It requires a sub-rule to execute an action if the comparison is true.

Format of Parameters

Expression

Where:

Expression One or more comparisons connected with AND or OR. Each comparison is in this format:

Value1 Value to be compared. This is a literal in double quotes or a variable containing the value.

Operand Choose how to compare the two values.

Value2 Value to be compared to *Value1*. This is a literal in double quotes or a variable containing the value.

Use parentheses to clarify which conditions go together. In these examples, one of the underlined sections must be true:

VarA EQ VarB AND VarC LT VarB OR VarC GE VarB OR VarD NE VarF

VarA EQ VarB AND VarC LT VarB OR (VarC GE VarB OR VarD NE VarF)

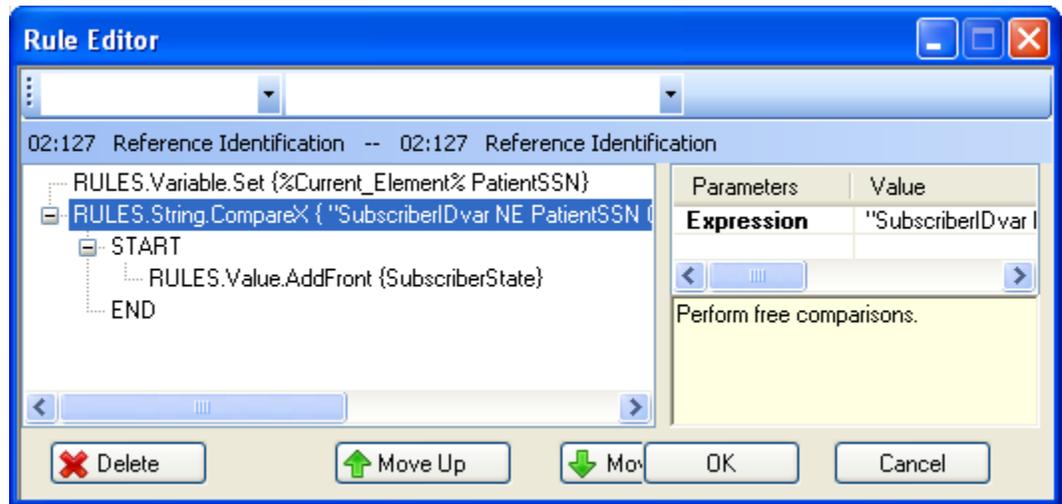
VarA EQ VarB OR (VarC LT VarB OR VarC GE VarB OR VarD NE VarF)

Example

This rule checks to see if either is true:

- SubscriberIDvar NE Patient SSN
- SubscriberState NE OH AND SubscriberState NE MI)

Either is true, it adds the contents of SubscriberState to the front of the current value.



Parameters	Value
Expression	"SubscriberDvar NE PatientSSN OR (SubscriberState NE "OH" AND SubscriberState NE "MI")"

ContainC

This rule checks a value for one or more characters and executes other rules based on whether the character was found. For the test to be true, any one or more of the characters must be found.

Format of Parameters

Value Characters

Where:

Value The item to check.

Characters One or more characters. This can be a literal in double quotes or a variable. If *any* of the characters are in the value, the test is true and the sub-rules execute accordingly.

Example 1

These rules check the current element for a dash, period, underscore, or space. If any are found, the rule is true and the output will contain INVALID.

The screenshot shows the Rule Editor interface. At the top, there are two dropdown menus: the first is set to 'RULES.String' and the second to 'ContainC'. Below this, the rule name '03:365 Communication Number Qualifier --' is displayed. The rule tree on the left shows a 'START' node leading to 'RULES.String.ContainC { %Current_Element% "-._" }', which then leads to 'RULES.Value.ReplaceWith { "INVALID" }' and finally to an 'END' node. On the right side, a table lists the parameters for the rule:

Parameters	Value
Value	%Current_Element%
Character(s)	"-._"

Example 2

These rules do the same thing, but the characters to check for are in a variable.

The screenshot shows the Rule Editor interface. At the top, there are two dropdown menus: the first is set to 'RULES.String' and the second to 'CompareC'. Below this, the rule name '04:364 Communication Number -- 04:364 Communication Number' is displayed. The rule tree on the left shows a 'START' node leading to 'RULES.Variable.Set { "-._" BadCharVar }', which then leads to 'RULES.String.ContainC { %Current_Element% BadCharVar }', which then leads to 'RULES.Value.ReplaceWith { "INVALID" }' and finally to an 'END' node. On the right side, a table lists the parameters for the rule:

Parameters	Value
Value	%Current_Element%
Character(s)	BadCharVar

ContainS

This rule checks a value for a string of characters and executes other rules based on whether the string was found. For the test to be true, the whole string must be found.

Format of Parameters

Value String

Where:

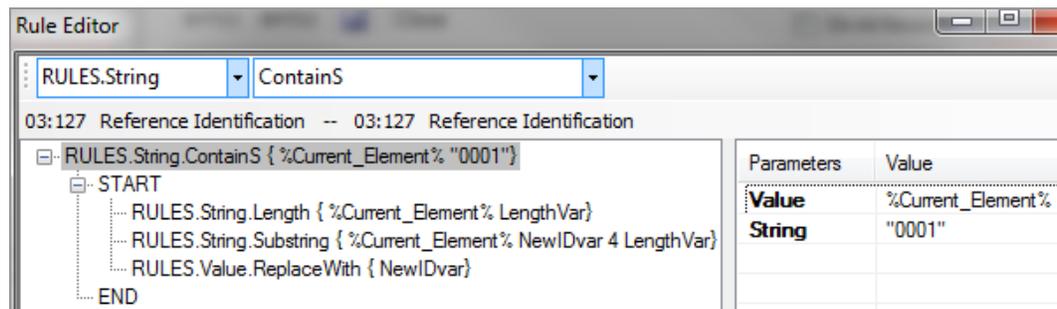
Value The item to check.

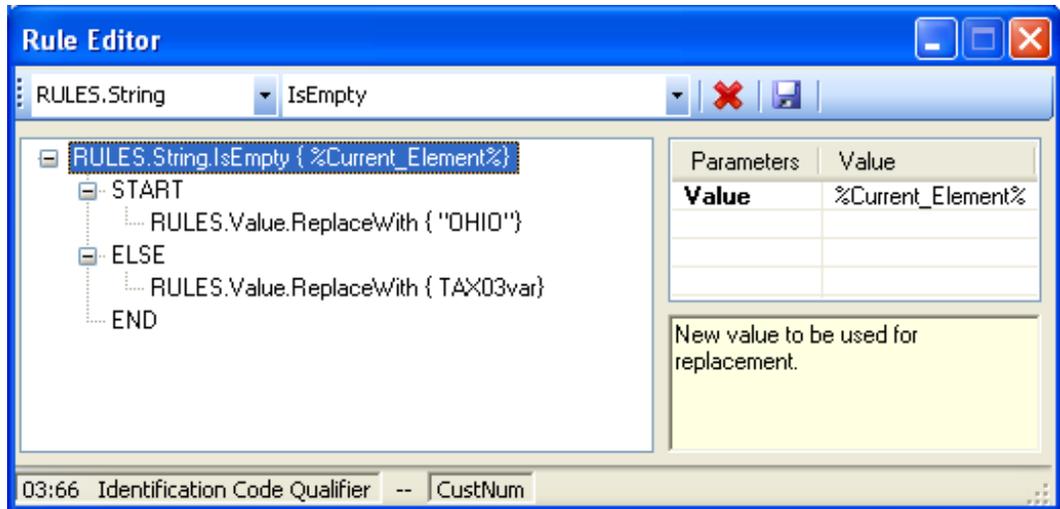
Characters One or more characters. This can be a literal in double quotes or a variable. If all of the characters are in the value, in order, the test is true and the sub-rules execute accordingly.

Example

These rules check the current element for 0001. If the entire string 0001 is found, these rules execute:

1. A rule to put the length of the value into a variable called LengthVar.
2. A rule to put the characters after the first four into a variable called NewIDvar.
3. A rule to use the contents of NewIDvar in the output.





HasValue

This rule checks whether an item has a value. It requires a sub-rule to act on the results of the test.

Format of Parameters

Value

Where:

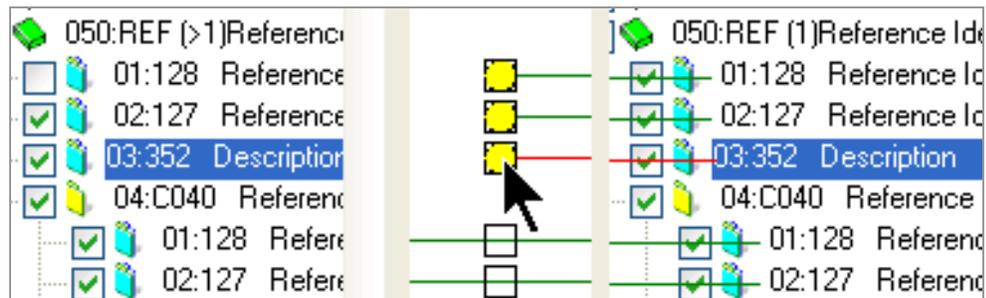
Value The variable that has a value or not.

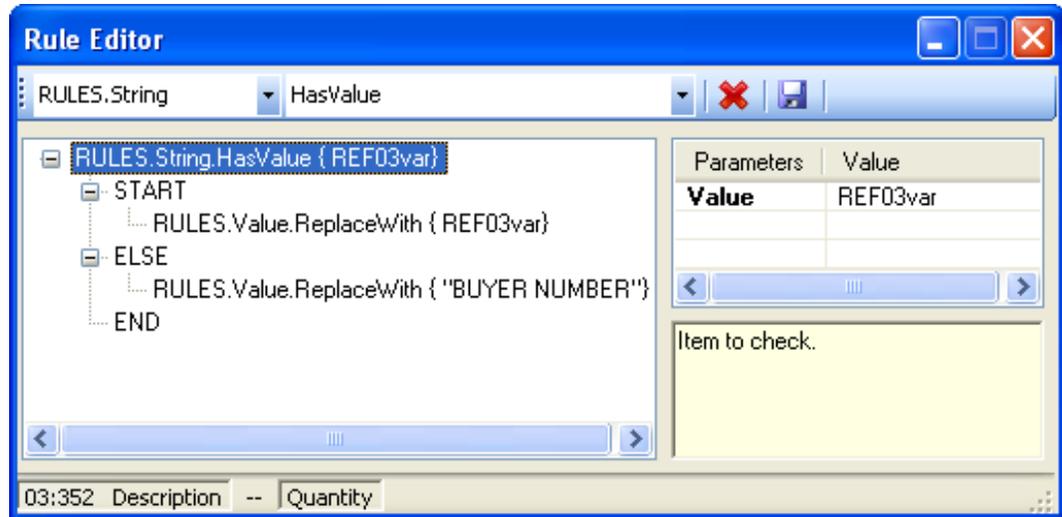
Example

This rule checks to see if a variable has a value.

- If true (it does have a value), we use the contents of variable REF03var in the output.
- If false (it does not have a value), we use the literal value BUYER NUMBER.

At least one element in the segment is mapped so that the segment will be in the output.





Please see page 272 for directions on how to set up conditional rules like this one.

IsEmpty

This rule checks whether an item is empty. It requires a sub-rule to act on this information.

Format of Parameters

Value

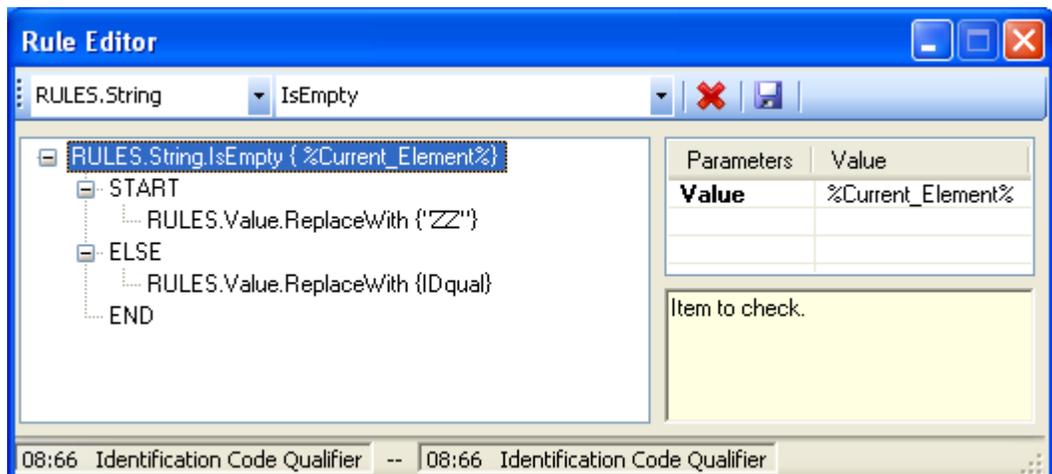
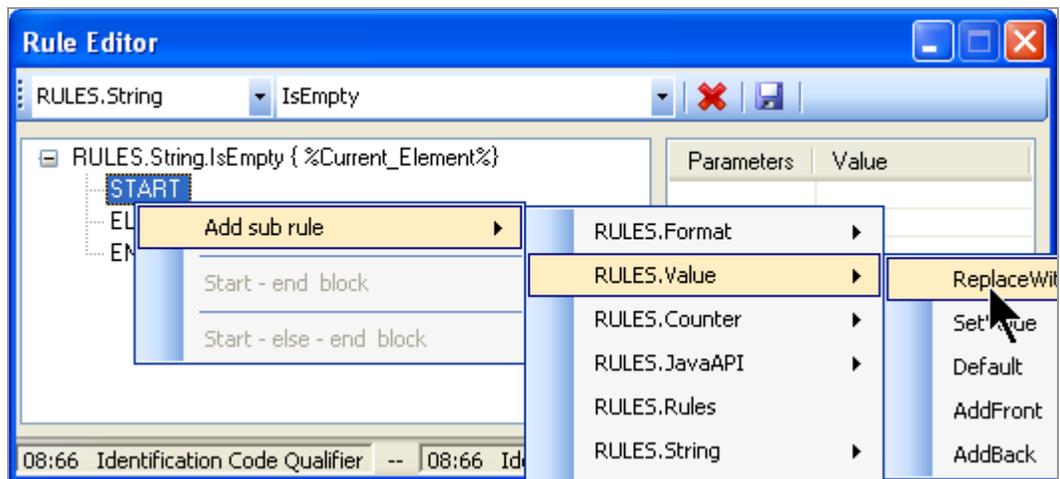
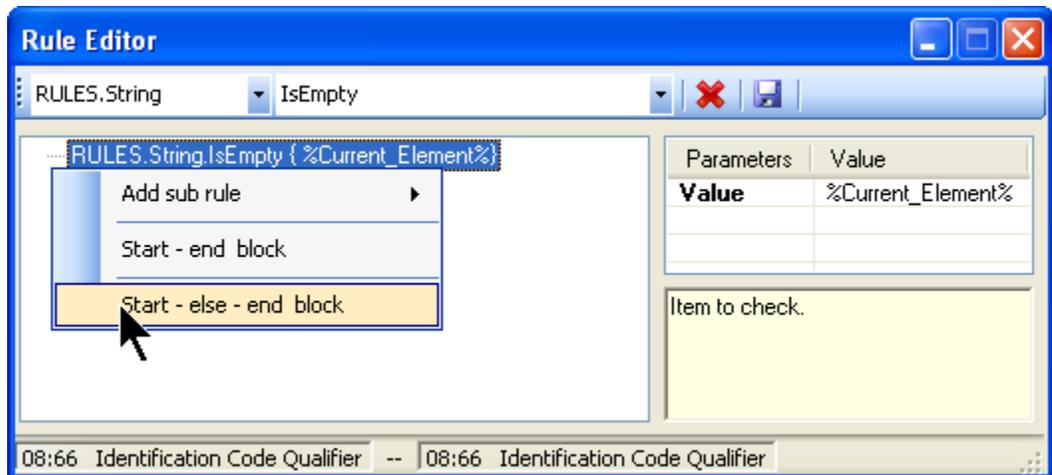
Where:

Value The item that is empty or not.

Example 1

These rules check to see if the current element is empty.

- If true (and it is empty), we use the literal value ZZ in the output.
- If false (and it is not empty), we use the contents of the variable IDqual, which presumably has been set with another rule.

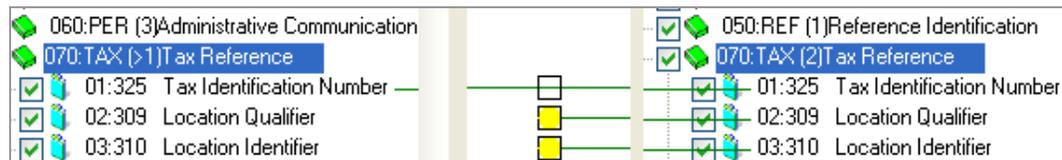


Please see page 272 for directions on how to set up conditional rules like this one.

Example 2 – Extended Example

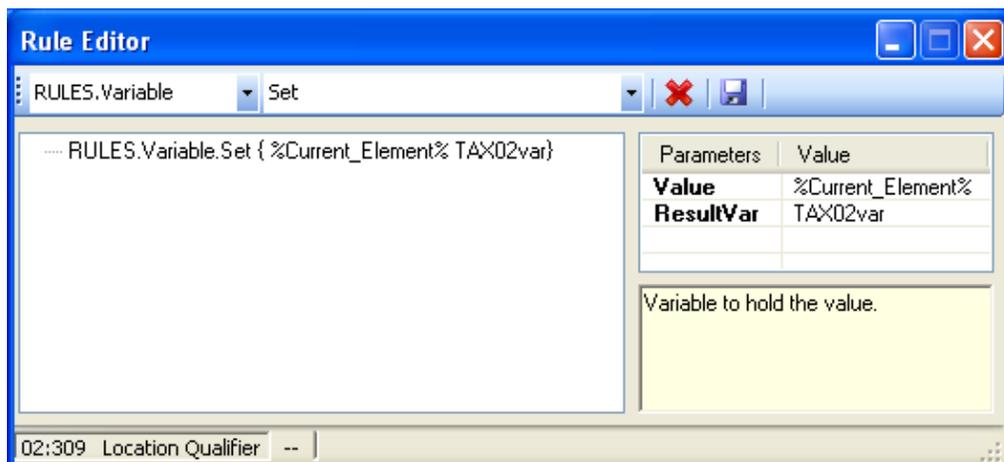
Our plan:

- If the source TAX02 is empty, we want to put ZZ in the output:
- Otherwise, we want to map the TAX02 from the source.
- We want to treat TAX03 the same way.

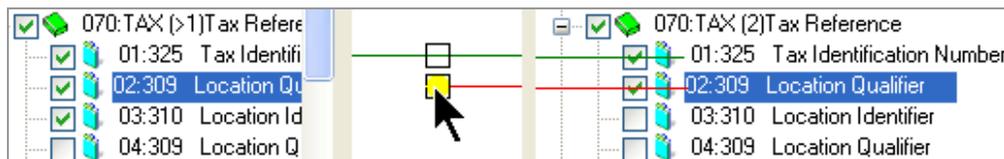


Steps:

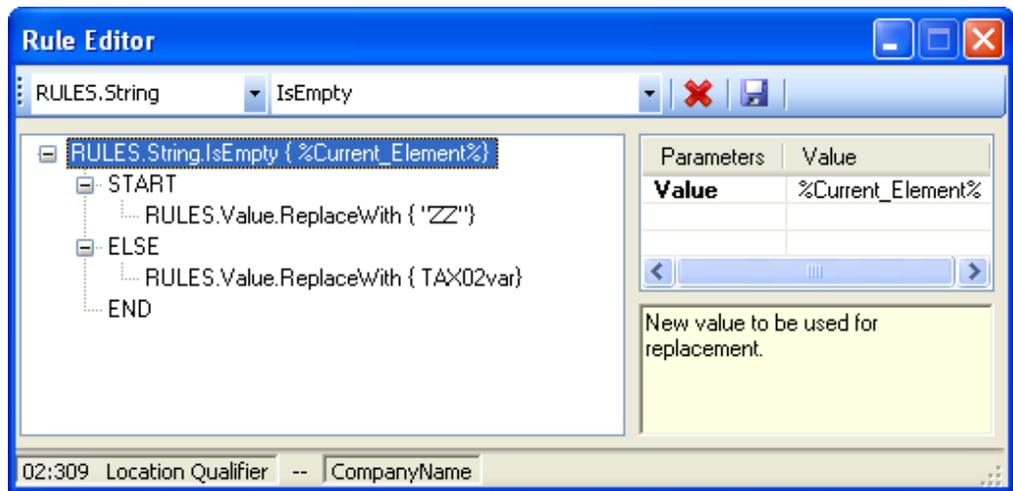
1. To collect the value from the **TAX02**, create a source rule on it:



2. The same way, collect the source value from the **TAX03**, using variable **TAX03var**.
3. Now populate the target TAX02.



Create the String.Compare rule shown in Example 1. This populates the output with ZZ or the actual value from the source - which is stored in variable tAX02var.



4. Finally, populate the TAX03 by setting up a similar rule.



This is the same as the rule in Step 3, except for the ReplaceWith values.

Length

This rule puts the length of a value into a variable.

Format of Parameters

Value ResultVar

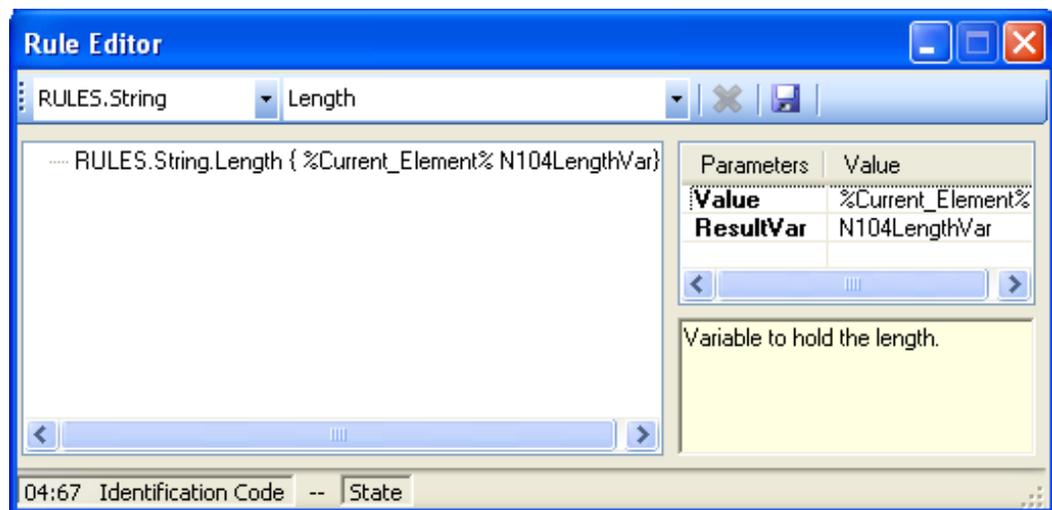
Where:

Value The value to measure.

ResultVar Variable to hold the length.

Example

This rule puts the length of the current value into variable N104LengthVar:



Loop

Repeatedly executes a list of rules until a specific condition fails. This rule requires one or more sub-rules.

Format of Parameters

“Expression”

Where:

“Expression” A test in this format:

Value1 Operand Value2 Condition Value3 Operand Value4...

Where:

Value1 The first value to compare.

Operand Relational operand to use for comparison:
EQ, NE, GT, GE, LT, or LE

Value2 The second value to compare.

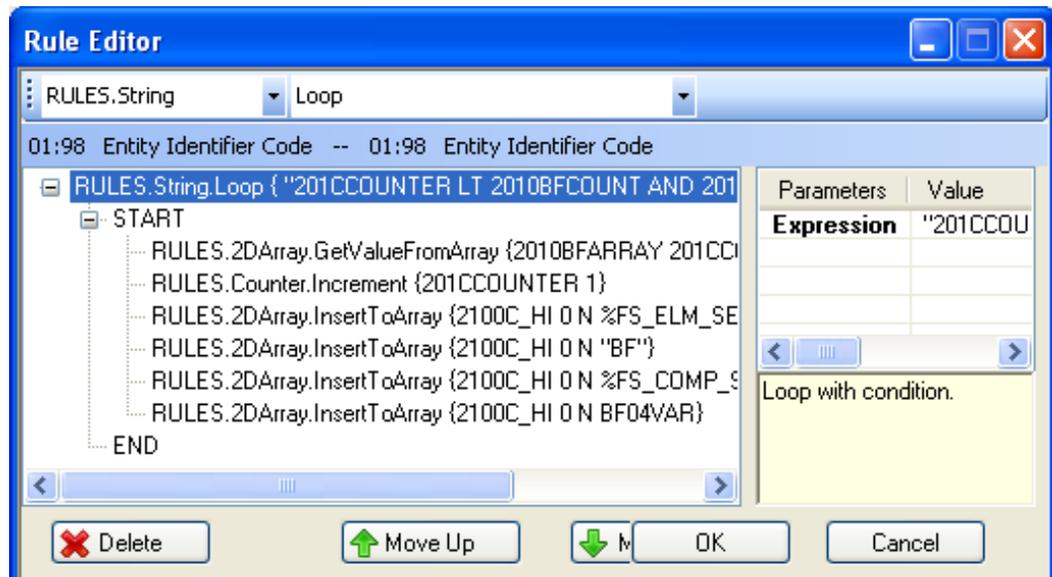
Condition Logical operand to use for comparison:
AND/OR

Value3 A third value to compare.

Operand Relational operand to use for comparison:
EQ, NE, GT, GE, LT, or LE

Value4 A fourth value to compare.

Example



Parameters:

Parameters	Value
Expression	"201CCOUNTER LT 2010BFCOUNT AND 201CCOUNTER LT '9'"

This checks a counter 201CCOUNTER to see if it is less than the value in 2010BFCOUNT and less than 9. If true, it executes a series of steps that puts values into an array that will be used later to create an HI segment. The counter is incremented as one of the steps.

RemoveC

This rule removes specified characters from a string.

Format of Parameters

InputValue ResultVar OldChar NewChar Option

Where:

Value The original value.

Characters Characters to be removed.

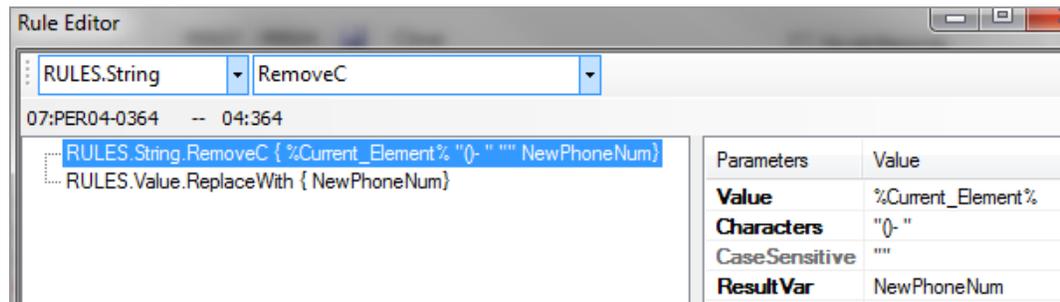
CaseSensitive If alphabetic characters are included, should the case be considered? Select YES or NO.

ResultVar A variable to hold the new value.

Example

This removes the following any of these special characters from a phone number: () - .

The characters within the quotes include a space, so spaces will be removed as well as parentheses and hyphens.



RemoveXMLEscapeCodes

Replaces XML escape codes with their actual values.

Special characters (e.g., <, >, and !) have designated meaning in XML. XML escape codes are used to signify those characters in XML data.

Examples of XML escape codes and the special character they represent:

```
&lt;    = <
&gt;    = >
&apos; = '

```

This rule can be used on the source or target and is useful when translating from XML-to-EDI using 2Darrays.

Format of Parameters

Value NewValue

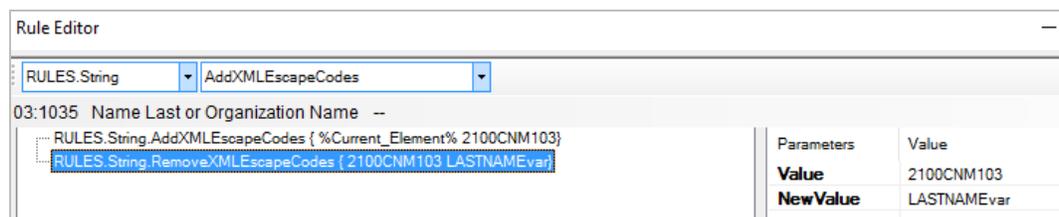
Where:

Value Value to be checked for XML escape codes. This is a literal in double quotes or a variable containing the value.

NewValue The resulting variable.

Example

Use this rule when XML escape characters could appear as part of a Subscriber Last Name, such as O'BRIEN. This rule removes the XML escape code for the apostrophe and replaces it with the actual character ('). For example, O'BRIEN would be converted to O ' BRIEN and stored in the variable specified in NewValue.



The screenshot shows the Rule Editor interface. At the top, there are two dropdown menus: the first is set to 'RULES.String' and the second to 'AddXMLEscapeCodes'. Below these, the rule name is '03:1035 Name Last or Organization Name --'. The rule definition is shown as a sequence of actions: 'RULES.String.AddXMLEscapeCodes { %Current_Element% 2100CNM103}' followed by 'RULES.String.RemoveXMLEscapeCodes { 2100CNM103 LASTNAMEvar}'. To the right of the rule definition is a table with two columns: 'Parameters' and 'Value'. The table contains two rows: 'Value' with the value '2100CNM103' and 'NewValue' with the value 'LASTNAMEvar'.

Parameters	Value
Value	2100CNM103
NewValue	LASTNAMEvar

Replace

This rule replaces all occurrences of a character with another character.

Format of Parameters

InputValue ResultVar OldChar NewChar Option

Where:

InputValue The original value.

ResultVar Variable to hold the result.

OldChar The old character, surrounded with double spaces if it is a literal.

NewChar The new character, surrounded with double spaces if it is a literal.

Option One of these:

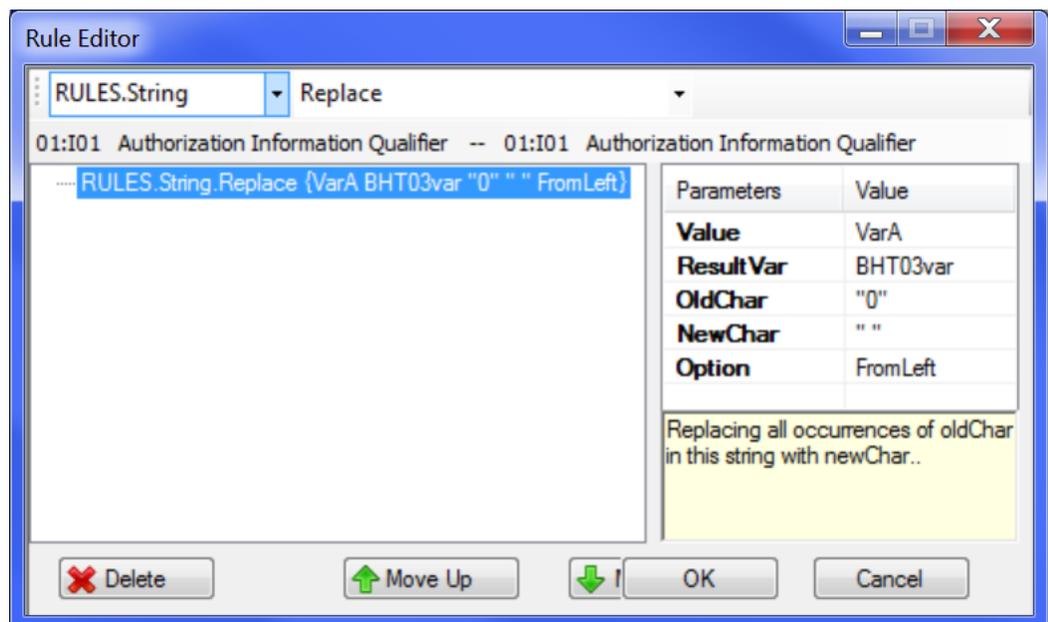
FromLeft Replace leading characters that match *OldChar*.

FromRight Replace trailing characters that match *OldChar*.

ALL Replace all matching characters.

Example

This replaces leading zeros with spaces.



Substring

This rule selects part of a value and puts it into a variable.

Format of Parameters

InputValue ResultVar Start Length

Where:

InputValue The original value.

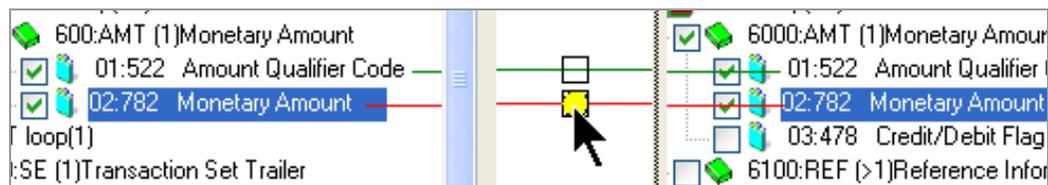
ResultVar A variable to hold the substring.

Start The position of the first character to be extracted from *InputValue*. This can be a number or a variable containing a number. It starts with 0, so that position 1 is 0 in the rule.

Length The position of the last character to be extracted from *InputValue*. This can be a number or a variable containing a number. If the *Length* is greater than the length of *InputValue*, *Length* will be set to the number of characters in *InputValue*. If empty, length is to the end of the value.

Example

This example has a rule on the source and a rule on the target. The element is mapped.



Source Rule

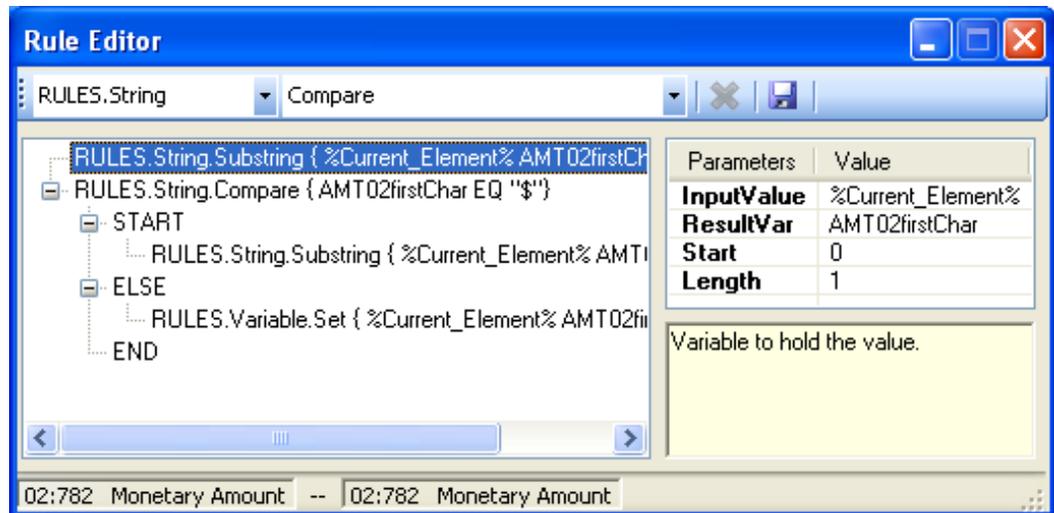
This rule removes the first character if it is a dollar sign:

Source	Target	
\$123.45	123.45	(removes leading dollar sign)
123.45	123.45	(if no dollar sign, leaves the first character)

The first source rule below puts the first character in the current element into variable **AMT02firstChar**. Note that a *Start* value of 0 is actually the 1st character in the value.

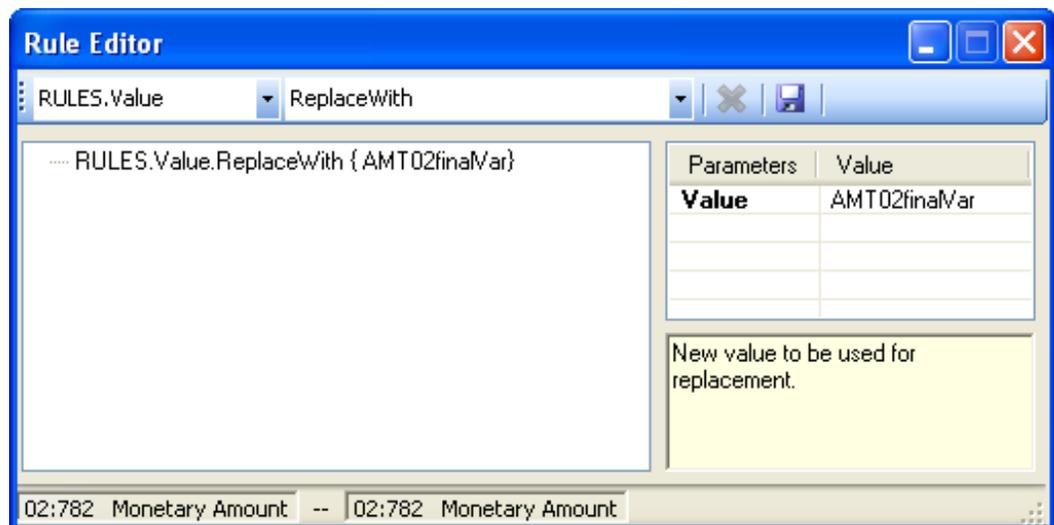
The Compare rule checks the contents of **AMT02firstChar** to see if it is a dollar sign.

- If true (it is \$), the Substring rule takes characters in position 2 through the end from the original value and put them in **AMT02finalVar**. Again, the “1” actually points to position 2 since the position numbering starts with 0.
- If false (it is not \$), the Variable.Set rule puts the entire contents of the current element into **AMT02finalVar**.



Target Rule

We then use the contents of **AMT02FinalVar** in the output data.



ToLowercase

This rule converts the input value to lowercase.

Value *NewValue*

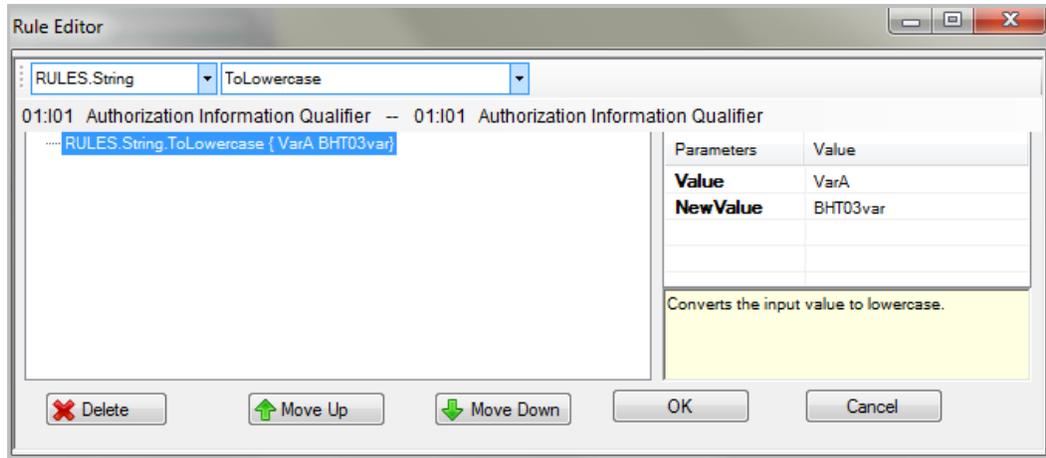
Where:

Value Value to be converted.

NewValue Variable to hold the converted value.

Example

This example takes the value in VarA, converts it to lowercase, and stores it in the variable BHT03var.



ToUppercase

This rule converts the input value to uppercase.

Format of Parameters

Value *NewValue*

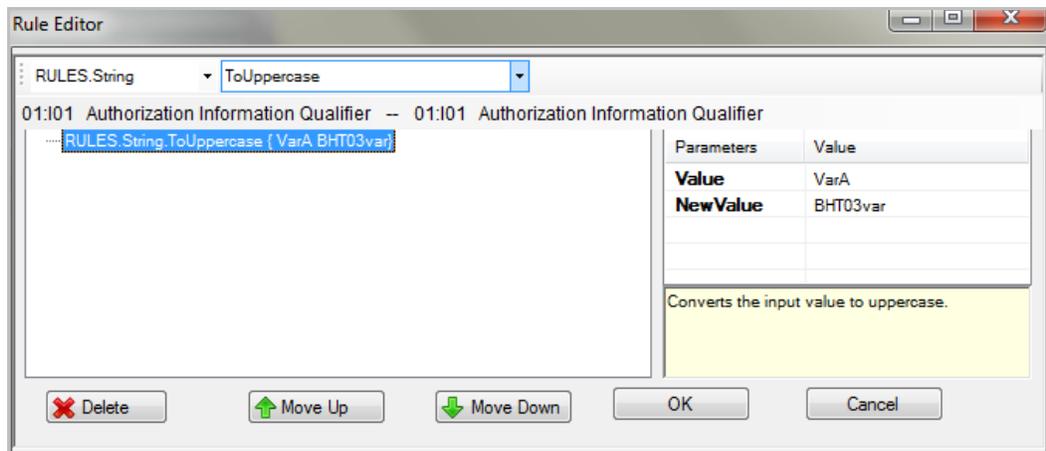
Where:

Value Value to be converted.

NewValue Variable to hold the converted value.

Example

This example takes the value in VarA, converts it to uppercase, and stores it in the variable BHT03var.



Rules.StringList

Insert

This rule allows values to be inserted into a list.

Format of Parameters

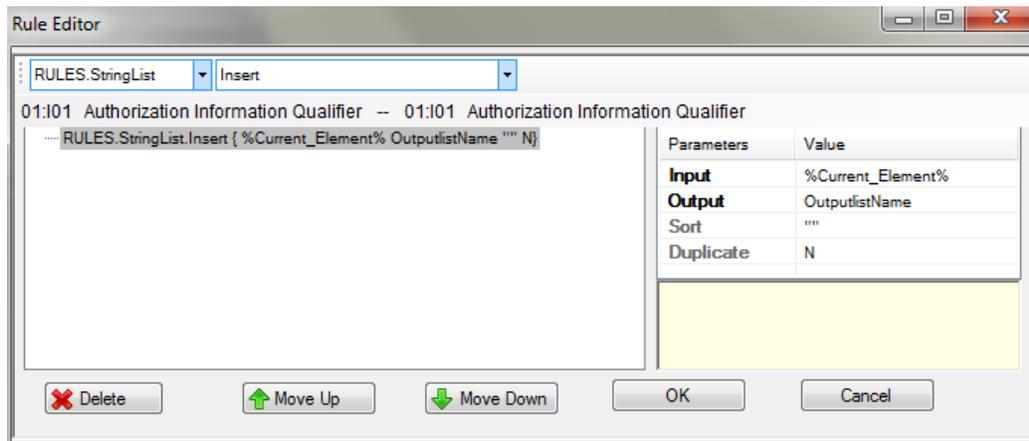
Input Output Search Duplicates

Where:

<i>Input</i>	List of literal values separated by commas and surrounded by double quotes, or a variable containing the list.
<i>Output</i>	The name of the output StringList.
<i>Sort</i>	Sort (Y) or do not sort (N) in the output StringList. Default = Yes.
<i>Duplicates</i>	Insert duplicates (Y) or do not insert duplicates (N) in the output string list. Default = Yes.

Example

This example inserts Current_Element AA, BB, BB, CC, DD into a StringList named OutputlistName, without duplicate values.



From the log, the OutputlistName looks like this:

```
[OutputlistName] = [3920394930203, AA, BB, CC, DD]
```

notice the duplicate (BB, BB) is omitted.

Search

Determines whether a value is in a list of literal values.

Format of Parameters

SearchValue *Operand* *ListOfValues*

Where:

SearchValue Is this value in the list?

Operand Choose one:

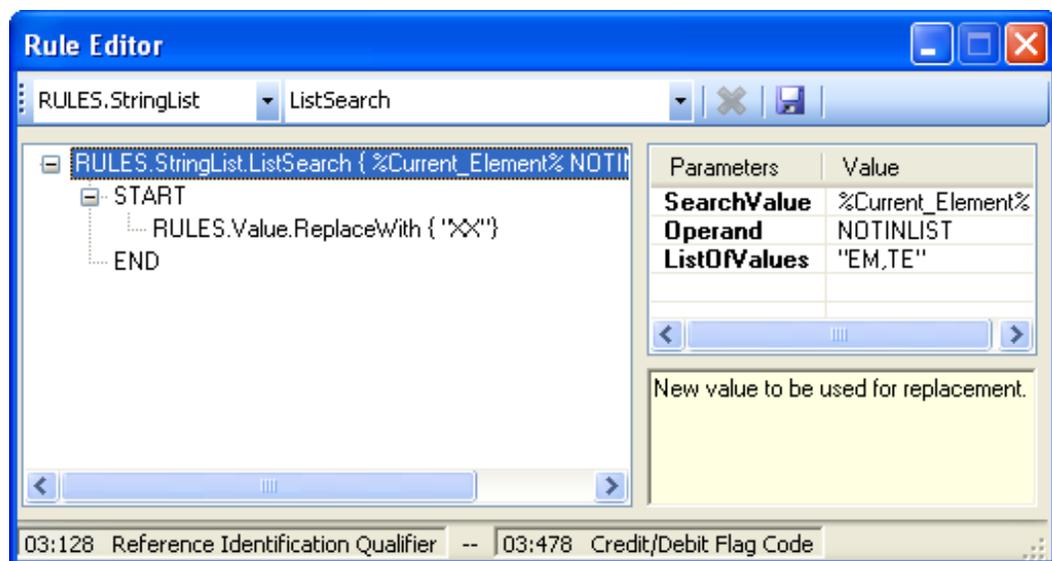
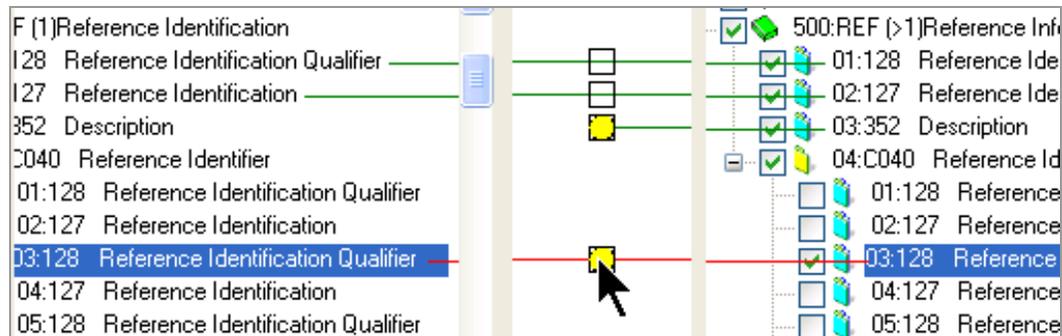
NOTINLIST

INLIST

ListOfValues List of literal values, separated by commas, and surrounded by double quotes.

Example

This rule checks the current value against the literal values TE and EM. If the value is not in the list, it uses XX in the output. It is mapped so that the source value will be used if it is in the list.



ListSearchX

Compares two lists and takes action based on whether they contain a common value. An action rule is required after ListSearchX.

Format of Parameters

ListOfValues *Operand* *ListOfValues*

Where:

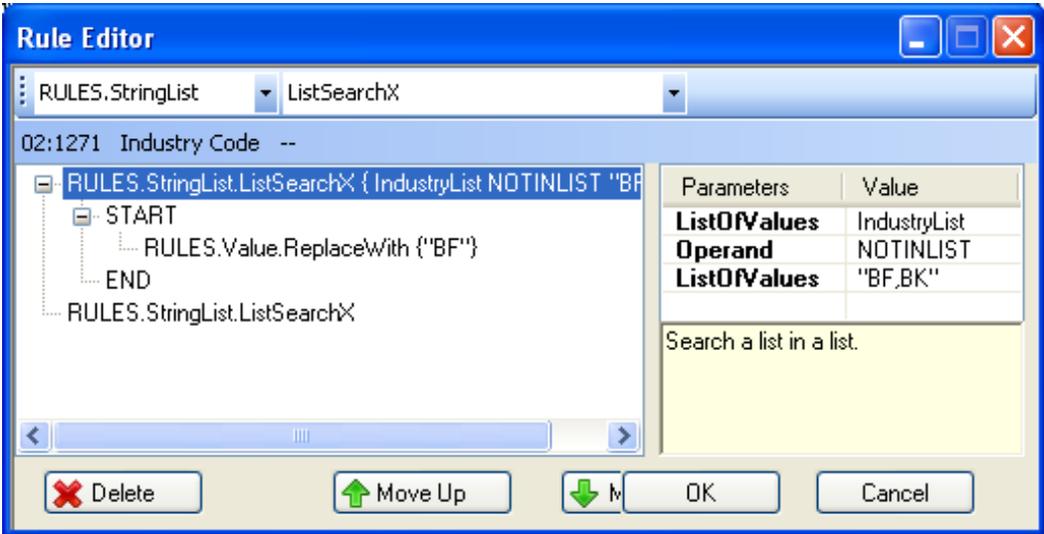
ListOfValues This is a list of literal values, separated by commas, or a variable containing the list.

Operand Choose one:
INLIST
NOTINLIST

ListOfValues Another list of literal values, separated by commas, or a variable containing the list.

Example

If IndustryList does not contain either BF or BK, the output will contain BF.



Rules.TableLookup

GetUniqueSequentialNumber

This rule lets you get a unique sequential number, starting with a number stored in an external file. It looks in the file for the number, returns it in a variable, and then increments the number in the file.

Format of Parameters

File ResultVar

Where:

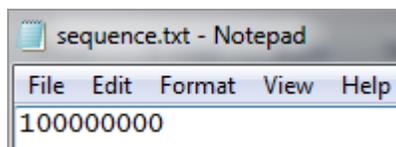
File This is the path and filename of a file containing a number to return and then increment by 1. It can also be environment variable FS_UNIQUESEQPATH (see Example 2 below). It should contain a digit or digits without leading zeros or spaces before or after.

ResultVar A variable to hold the new number.

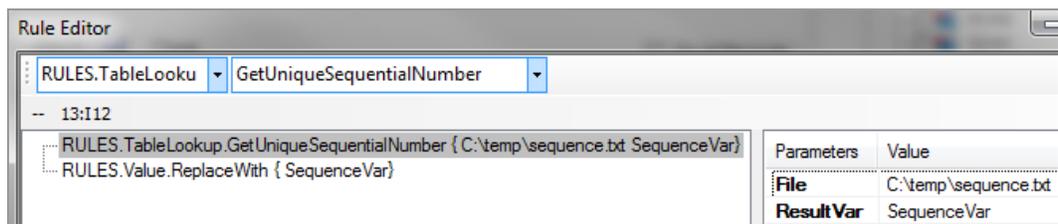
Example 1

This gets the next ST control number from a file, uses it in the output, and increments the number in the file by 1.

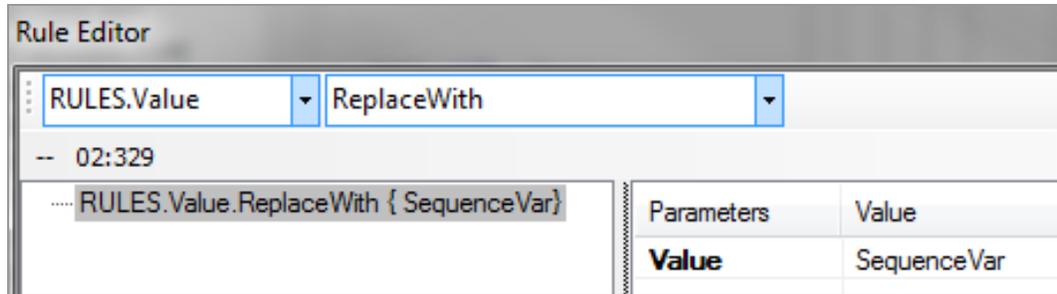
The file containing the sequence number:



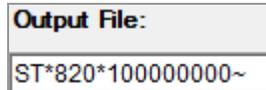
Rule on the ST-02:



Rule on the SE-02:

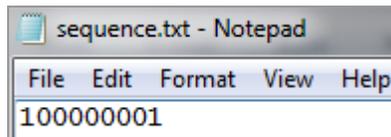


Result on the ST:



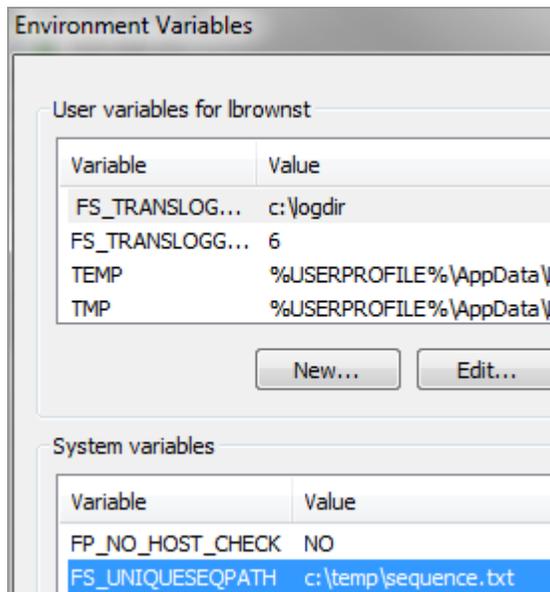
The SE-02 would contain the same result: 100000000.

The sequence number in the file has been automatically incremented by 1:

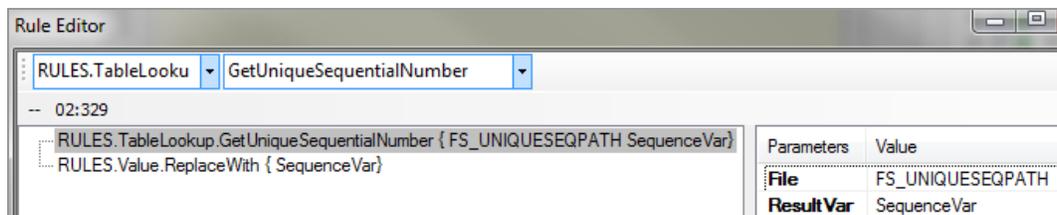


Example 2

This is similar to Example 1 but it uses environment variable FS_UNIQUESEQPATH to identify the file:



Rule:



SearchInTable

This rule looks for one or more values in an external table and returns another value from the same row.

The table is in CSV format and located in Foresight Translator's \Bin directory. The first row is column headings and all other rows contain data.

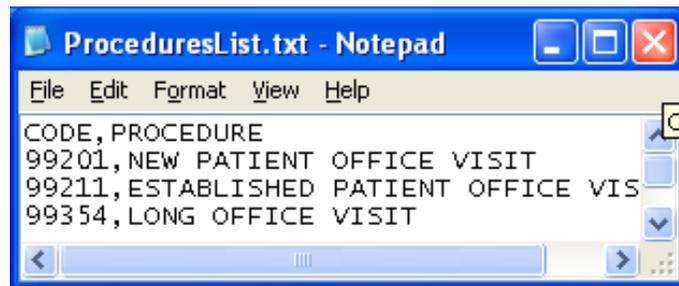
Format of Parameters

TableName InputColumn OutputColumn

Where:

TableName Filename of table to search, a literal surrounded by double quotes. The table must be in Foresight Translator's \Bin directory.

Example table:



The first row is for column headings, separated by commas.

Each additional row contains data, separated by commas.

InputColumn What to search for, and where, in this format:

column=value column=value column=value

This is the heading of the column to search, followed by an equal sign and the value to search for in the column.

If you have multiple columns, separate each with a space. With multiple columns, all values must be found in the same table row in order for the return variable to be populated.

Example: CODE=SV20202var CITY=SubCITYvar STATE="OH"

OutputColumn What to return, in this format:
variable=column

Example:

```
ReturnProcVar=PROCEDURE
```

This is the variable that is returned, followed by an equal sign, followed by the heading of the column that contains the value to be returned.

If the search value is not found, the return variable contains its own name. In our example, ReturnProcVar would contain “ReturnProcVar”.

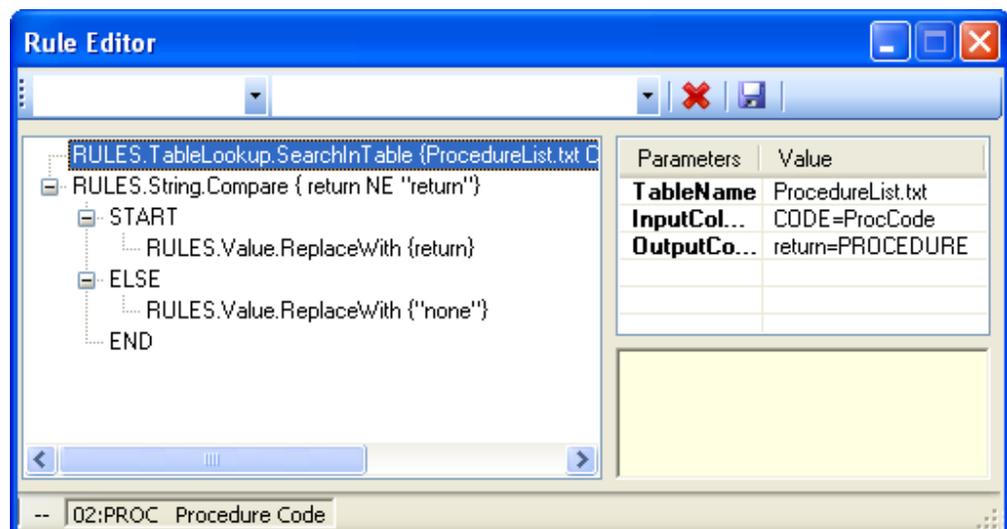
Example 1

The following target rule says:

1. Look in a file called **ProcedureList.txt**.
2. Look in the CODE column for the value in variable **ProcCode**.

If the value is found, return the corresponding contents of the PROCEDURE column in a variable called **return**.

If the value is not found, return will contain the literal value “return,” – the name of the return variable itself.



The String.Compare rule checks the value in the return variable. If it does not contain “return,” then it uses the value that it does contain in the output. Otherwise, it uses the word “none” in the output.

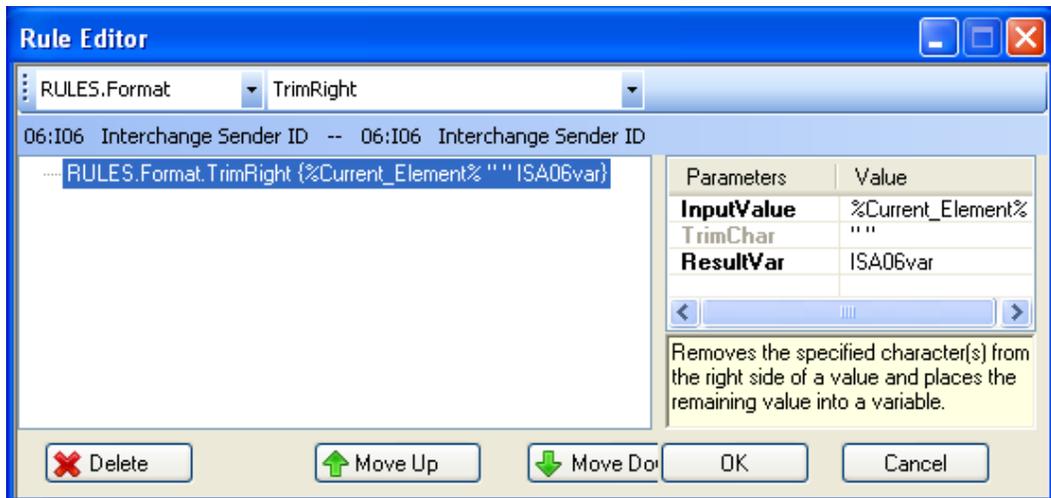
As always, remember to clear the return variable before each execution of the rule.

Example 2

This rule searches external file CoreSysTable.txt, in Foresight Translator's \Bin directory:

```
CoreSysTable.txt
SENDERcol, RECEIVERcol, CORESYScol
SAAAAA, RAAAAA, EAST
SAAAAA, RZZZZZ, WEST
SBBBBB, RBBBBB, WEST
SZZZZZ, RCCCCC,
SDDDDD, RDDDDD, EAST
```

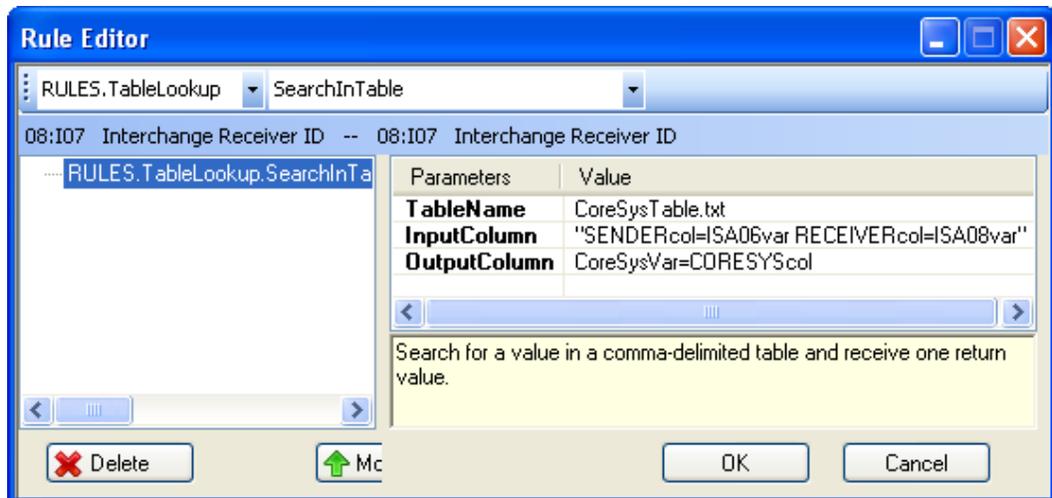
It uses the ISA06 and the ISA08 without any trailing spaces:



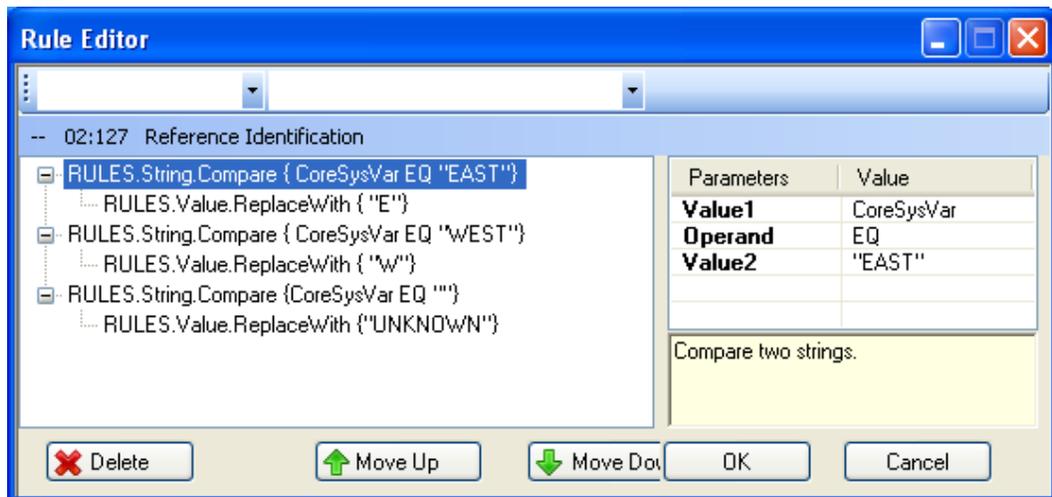
The TableLookup searches the external file for a row in which:

- The SENDERcol contains a value that matches the contents of ISA06var
- AND
- The RECEIVERcol contains a value that matches the contents of ISA08var. If a row is found that contains the ISA06 and ISA08 values, the value in the CORESYScol column is placed in a variable called CoreSysVar.

Notice the space between the two columns in the InputColumn parameter.



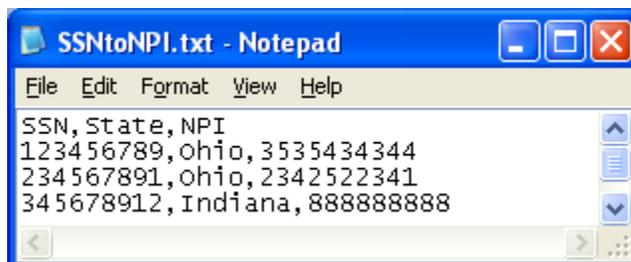
You can then use CoreSysVar various ways:



Example 3 – Extended Example

This example replaces a social security number with a corresponding NPI number.

Foresight Translator's \Bin directory has this table. If the source value is in the SSN column, we want to use the corresponding value in the NPI column in the target element.



Example source segment

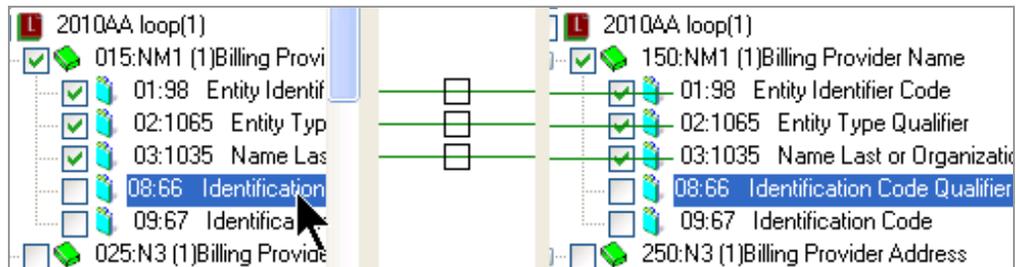
NM1*85*2*JONES*****34*123456789~

Same segment after translation

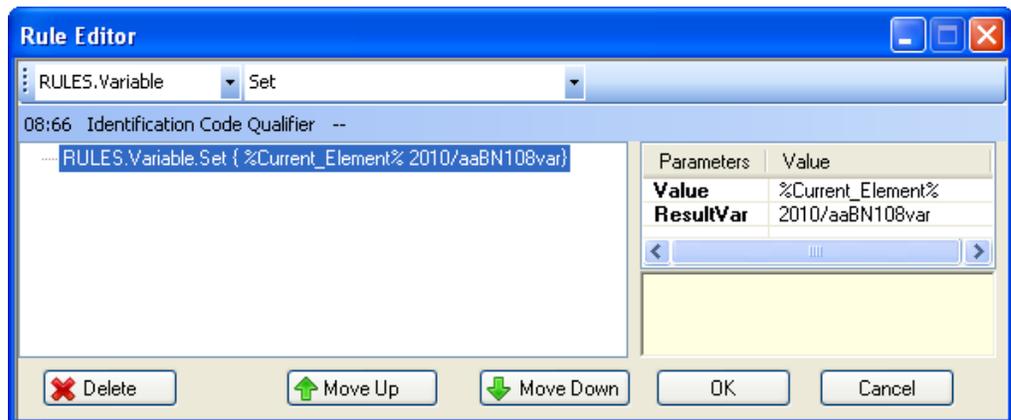
NM1*85*2*JONES*****XX*353543434
4^

Steps:

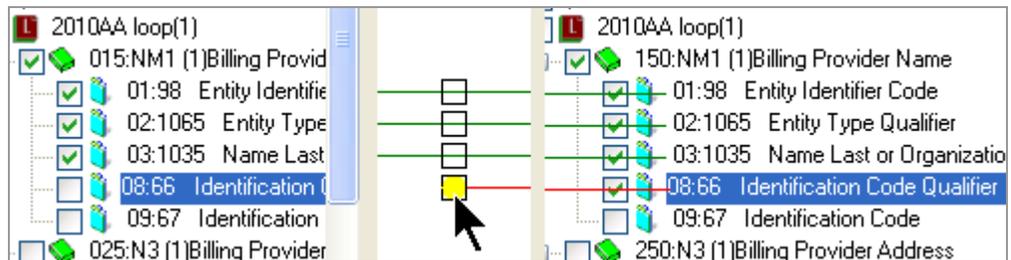
1. First, we need to capture the source value from the NM108 (the qualifier). Later, we will see if it is 34, which designates a social security number.



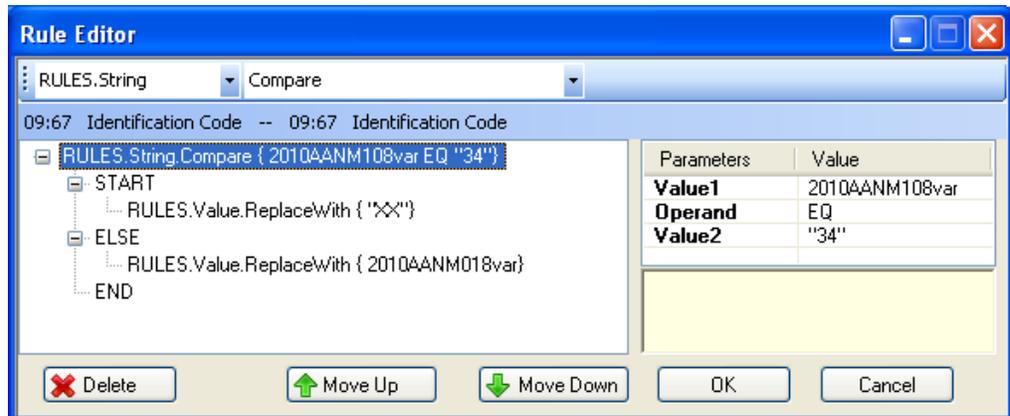
This rule on the source element captures the qualifier in variable **2010AANM108var**.



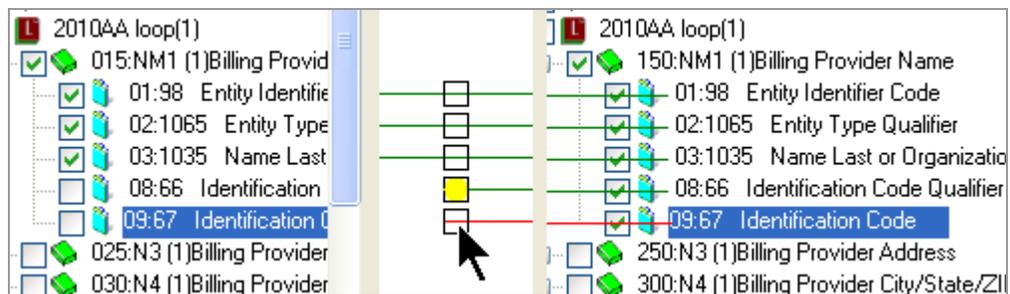
2. Likewise, capture the value in the NM109 in variable **2010AANM109var**.
3. Set up a rule on the target NM108.



This checks variable **2010AANM108var** to see if it contains 34 – the qualifier for a Social Security Number. If it is 34, replace it with XX. Otherwise, use whatever was in the source.



4. Put rules on the target NM109:



The first **String.Compare** set of rules checks 2010AANM108var.

- a. If it contains 34:

Check the SSN column in table SSNtoNPI.txt for the value in variable 2010AANM109var.

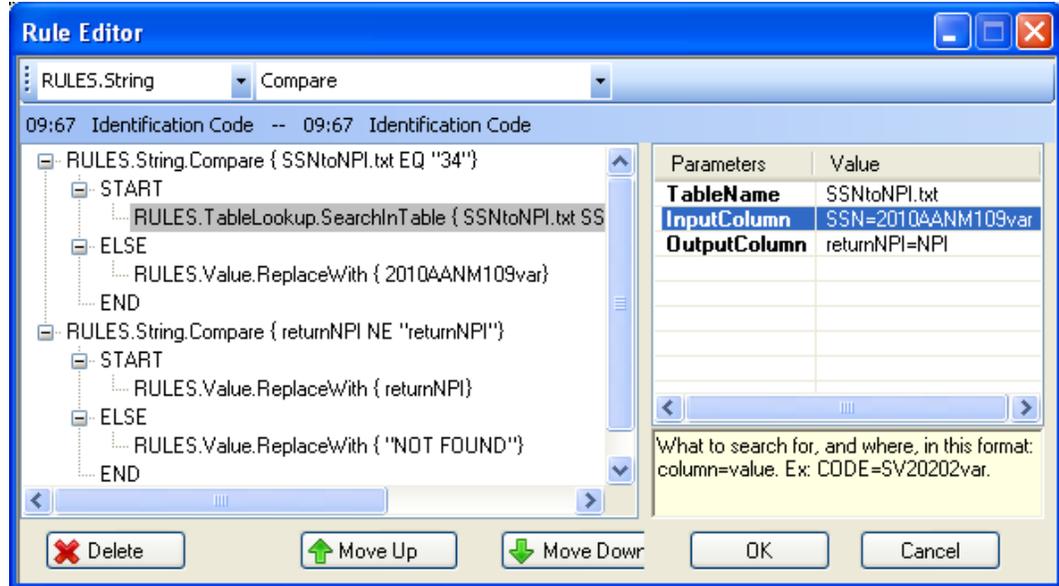
Place the corresponding value from the NPI column into a variable called returnNPI.

(If it is not found, the literal value returnNPI will be returned in variable returnNPI. A failed match always returns the name of the return value itself.)

- b. If it doesn't contain 34, use the value in returnNPI in the output.

The second String.Compare set of rules checks to see if it does not contain the literal value returnNPI.

- a. If it is true that it doesn't, it places the returned value into the output.
- b. Otherwise, it uses the literal value NOT FOUND in the output.



Rules.Value

AddBack

This rule appends a literal or the contents of a variable to the current element.

Format of Parameters

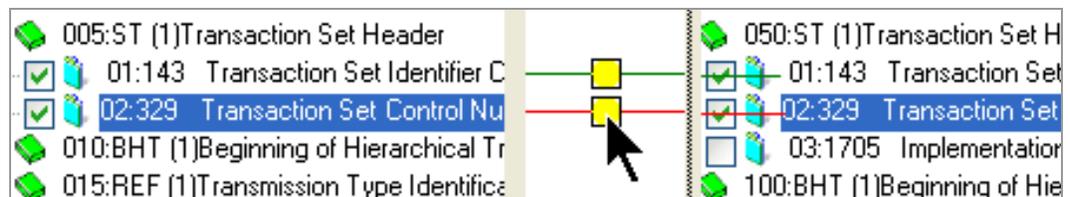
Value

Where:

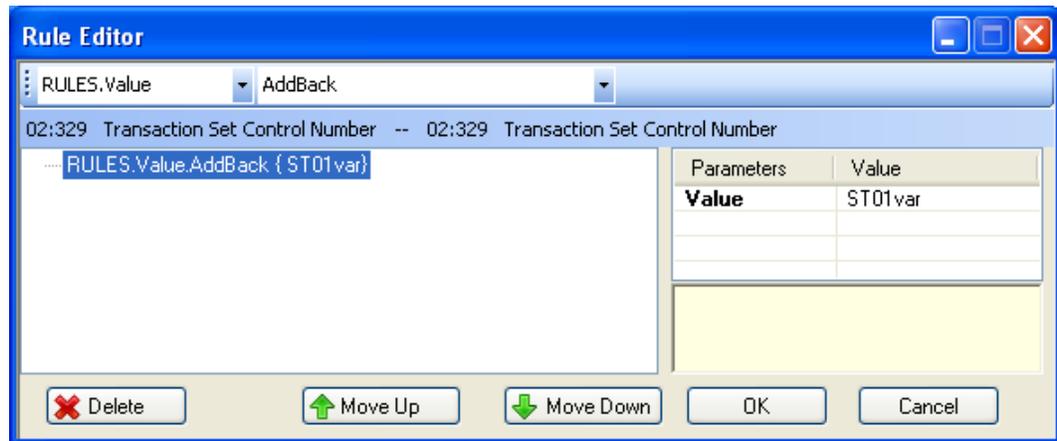
Value A literal in double quotes or a variable containing the value to append to the current element.

Example

This rule is on the ST02, which is mapped.



It appends the value in the variable ST01var to the contents of the ST02.



AddFront

This rule puts a literal or the contents of a variable in front of the value in the current element.

Format of Parameters

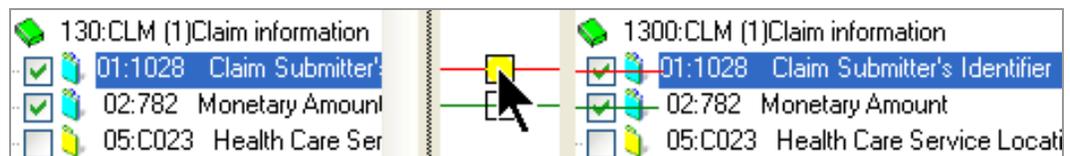
Value

Where:

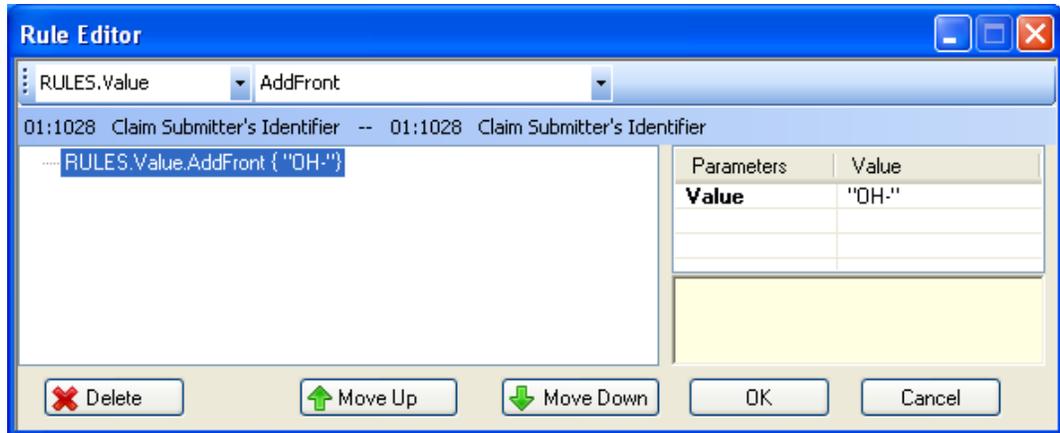
Value A literal in double quotes or a variable containing the value to put in front of the current element.

Example

This rule is on the CLM01, which is mapped.



It places a literal value "OH-" in front of the value in the CLM01.



Append

The rule concatenates two or more values into a variable.

Format of Parameters

"InputValues" ResultVar

Where:

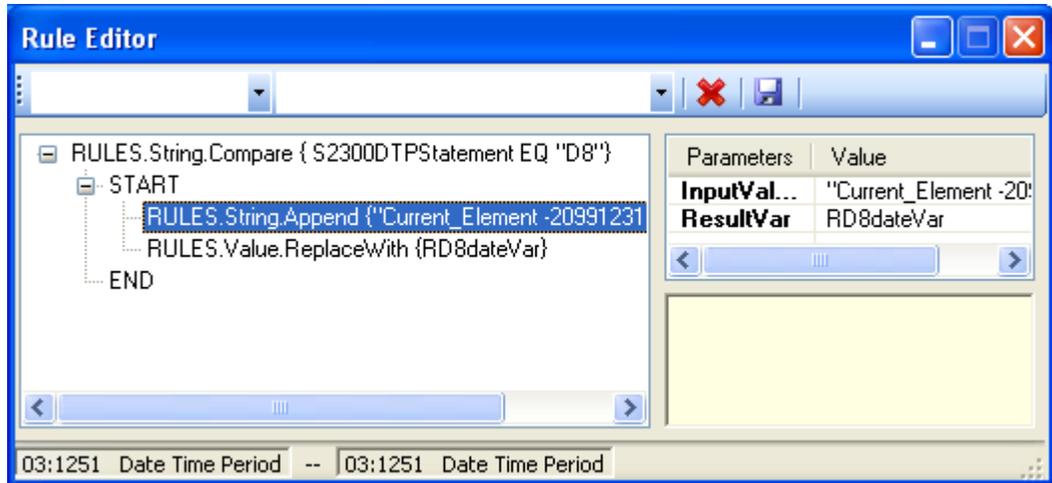
InputValues A list of two or more values, separated by spaces. This can consist of literals, variables, and reserved variables. The entire list of values is enclosed in double quotes. Spaces are separators so you cannot have spaces in your own strings.

ResultVar Variable to contain the result.

Example

This rule builds a date range by concatenating the current value, a hyphen, and a literal and storing it in variable RD8dateVar. It then uses that in the output.

This rule assumes that the date qualifier was previously captured in variable S2300DTPStatement.



Parameters area:

Parameters	Value
InputValues	"Current_Element -20991231"
ResultVar	RD8dateVar

Example segment before translation: DTP*434*D8*20030212~

Same segment after translation: DTP*434*RD8*20030212-20991231~

CheckType

Checks a value to see if it has a certain data type. It requires a sub-rule that executes if the test is true.

Format of Parameters

Value *Rule*

Where:

Value A literal in double quotes or a variable containing the value to add as default value.

Rule One of these:

HasAlpha Does it contain an alphabetic character? (an uppercase or lowercase letter)

HasNumeric Does it contain a numeric character? (a digit)

HasAlphanumeric Does it contain an alphabetic or numeric character? (an uppercase or lowercase letter or digit)

IsAlpha Is it entirely alphabetic? (punctuation and spaces are not alpha)

IsNumeric

Is it entirely numeric? (punctuation and spaces are not numeric)

IsAlphanumeric

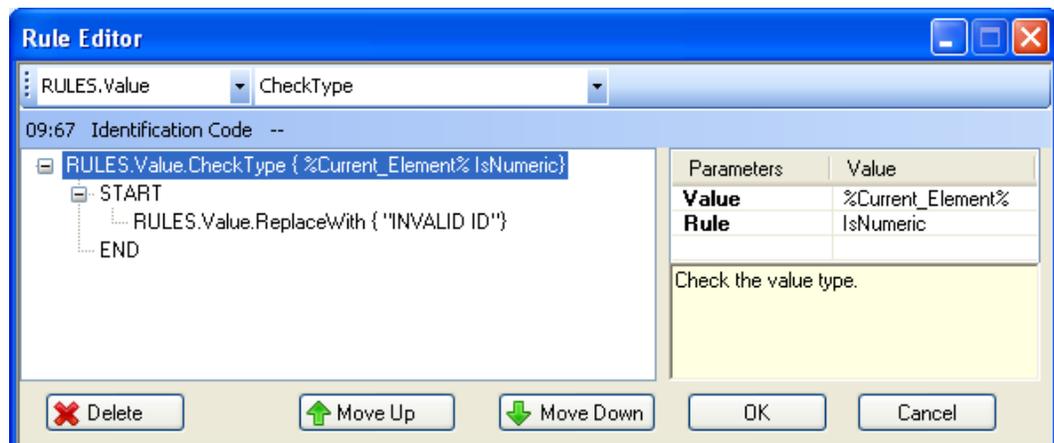
Is it entirely alphanumeric?

Examples:

Value	IsAlpha	HasAlpha	IsNumeric	HasNumeric	IsAlphanumeric	HasAlphanumeric
111			✓	✓	✓	✓
1.0				✓		✓
1. 0						
AaA	✓	✓			✓	✓
Aa A		✓				✓
#						
<space>						
A2		✓		✓	✓	✓

Example

If the current element is entirely numeric, the value is replaced with “INVALID ID.”



Default

This target rule adds a default value to flat file targets under certain conditions.

Format of Parameters

Value

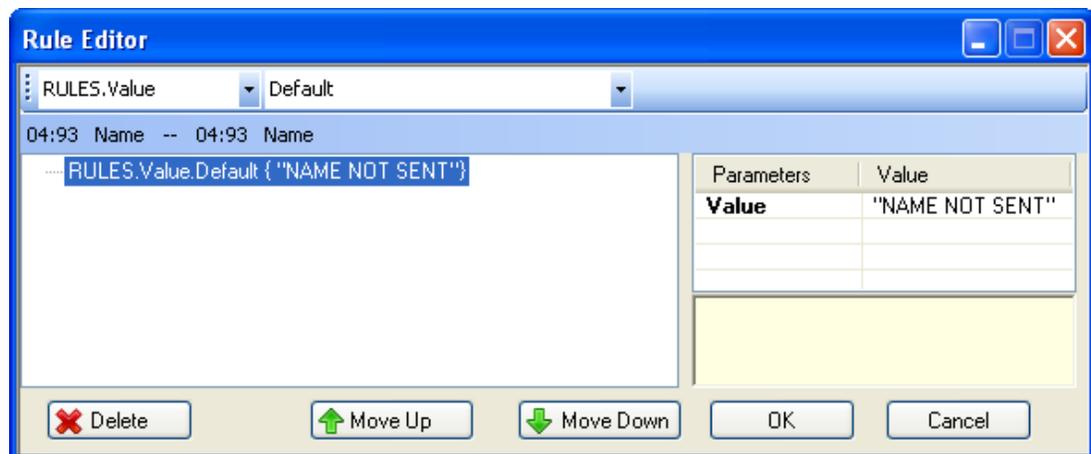
Where:

Value A literal in double quotes or a variable containing the value to add as default value.

Example

Assume that this Name element is optional in the source but mandatory in the target. We map it so that the source value will be used if it is sent.

This rule provides the value “NAME NOT SENT” if there is no data coming from the source.



GetInterchangeCodeValue

This puts a value from the target guideline's interchange or group enveloping into a variable. The target can be X12, EDIFACT, or TRADACOMS.

Format of Parameters

SegmentName *ElementPosition* *Result*

Where:

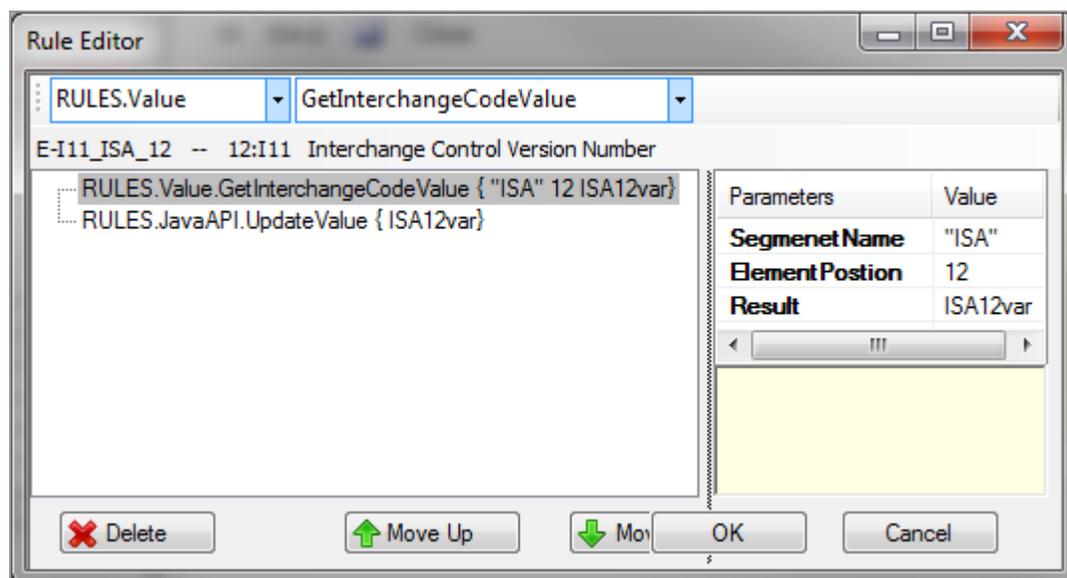
SegmentName The tag of the target segment that contains the value to be captured. A literal in double quotes or a variable containing the value.

ElementPosition The position number of the element within the segment. If the value is within a composite, this will be in the format *n-n*. Example: 4-2 is second subelement in the composite at position 4.

Result A variable to hold the captured value.

Example

This example captures the target's ISA12 (Interchange Control Version Number) into a variable ISA12var. The next rule sends the value to the Java program that called Foresight Translator.



IsUnPacked

This checks a target value to see if it has been automatically unpacked. It requires a block of sub-rules to take action based on the results.

Format of Parameters

Value

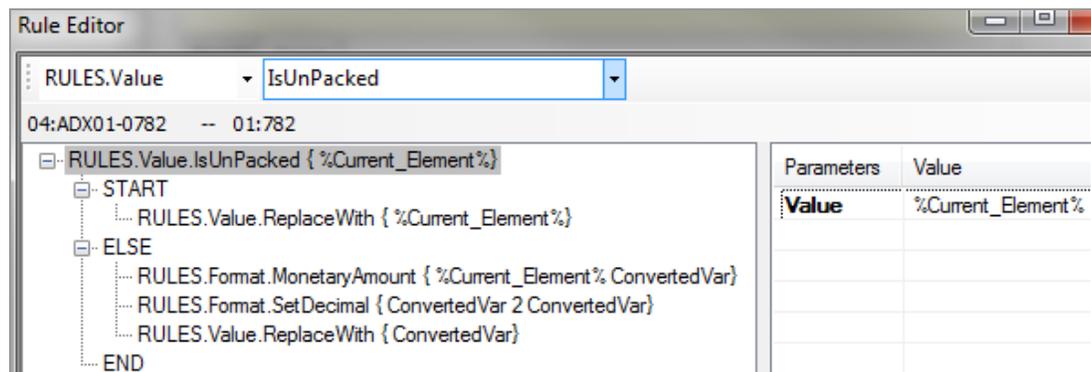
Where:

Value Value that may or may not have been unpacked.

Example

This example checks a value that is type N2 in the source and R in the target. If it is already unpacked, it simply passes the value through to the target. If it isn't unpacked, it:

1. Removes leading zeros with MonetaryAmount
2. Inserts an explicit decimal two digits from the right with SetDecimal
3. Uses the resulting value in the output



The screenshot shows the Rule Editor interface. At the top, the rule name is "IsUnPacked" under the "RULES.Value" category. Below the rule name, the rule definition is shown in a tree view:

```
04:ADX01-0782 -- 01:782
├── RULES.Value.IsUnPacked { %Current_Element% }
│   ├── START
│   │   └── RULES.Value.ReplaceWith { %Current_Element% }
│   └── ELSE
│       ├── RULES.Format.MonetaryAmount { %Current_Element% ConvertedVar }
│       ├── RULES.Format.SetDecimal { ConvertedVar 2 ConvertedVar }
│       └── RULES.Value.ReplaceWith { ConvertedVar }
└── END
```

On the right side of the editor, there is a table with two columns: "Parameters" and "Value". The table contains one row with the parameter "Value" and its corresponding value "%Current_Element%".

Parameters	Value
Value	%Current_Element%

ReplaceWith

This rule replaces the value in the current field with another value.

Format of Parameters

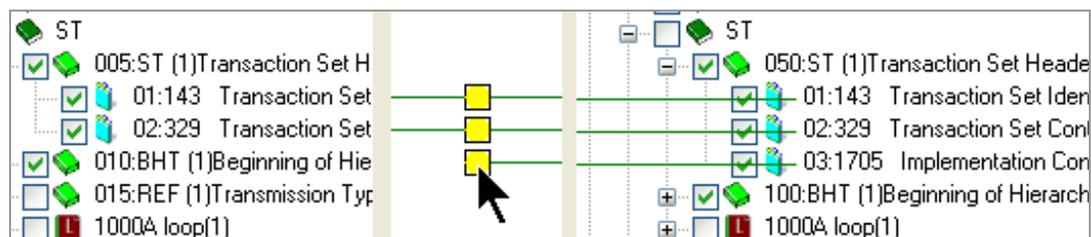
Value Delimiter

Where:

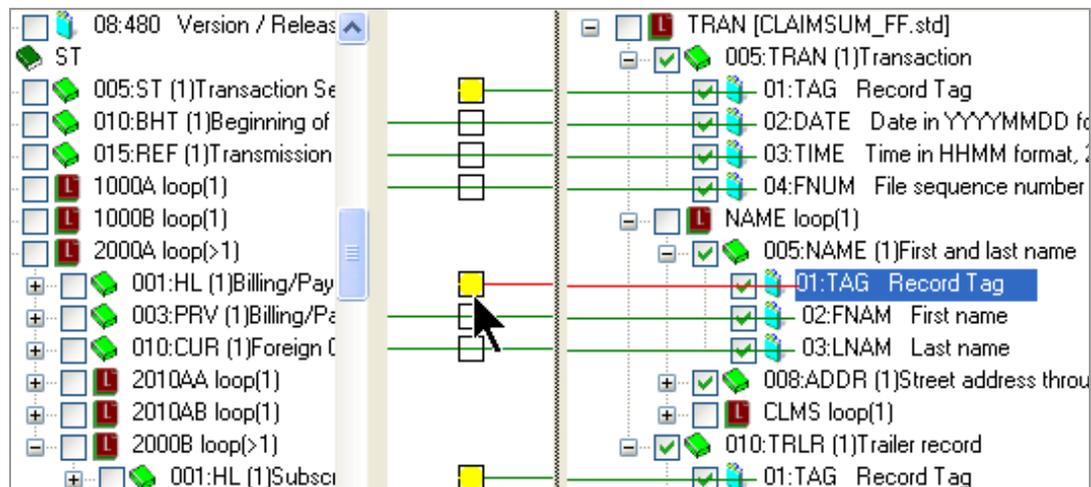
Value The new value – a reserved word, variable, or literal in double quotes.

Delimiter Yes – include segment delimiter
No – do not include segment delimiter

ReplaceWith rules can fill in unmapped fields:

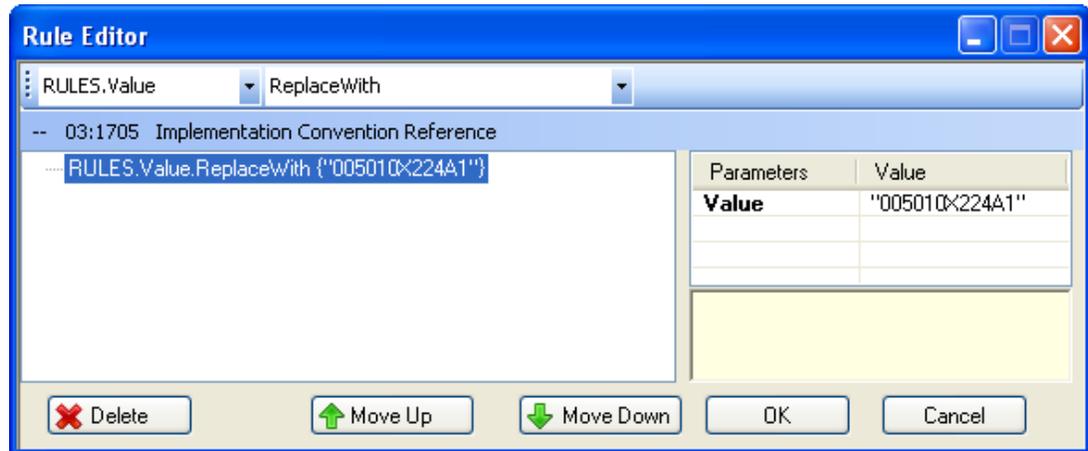


For flat file targets, they can fill the empty segment tags:

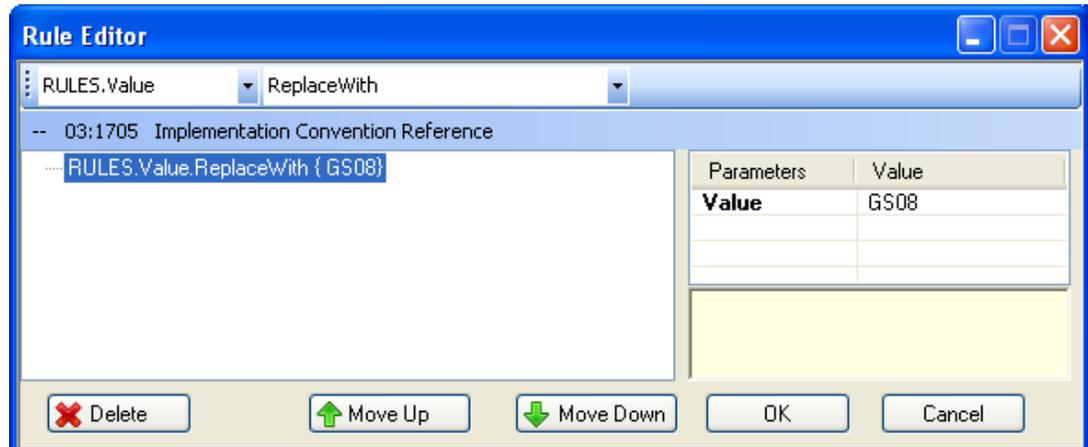


Example

This replaces the value in the current field with the literal 005010X224A1.



This replaces the value in the current field with the contents of variable **GS08var**:



SetValue

This target rule specifies a value for the output only if the segment has a qualifier that has been sent in the data, not when it is generated by a rule. This prevents infinite loops when using 2 pass.

Use it when:

- You are using 2 pass
- The source segment has a qualifier

Only use it on segments that have qualifiers - such as NM1 segments.

Do not map the element.

Format of Parameters

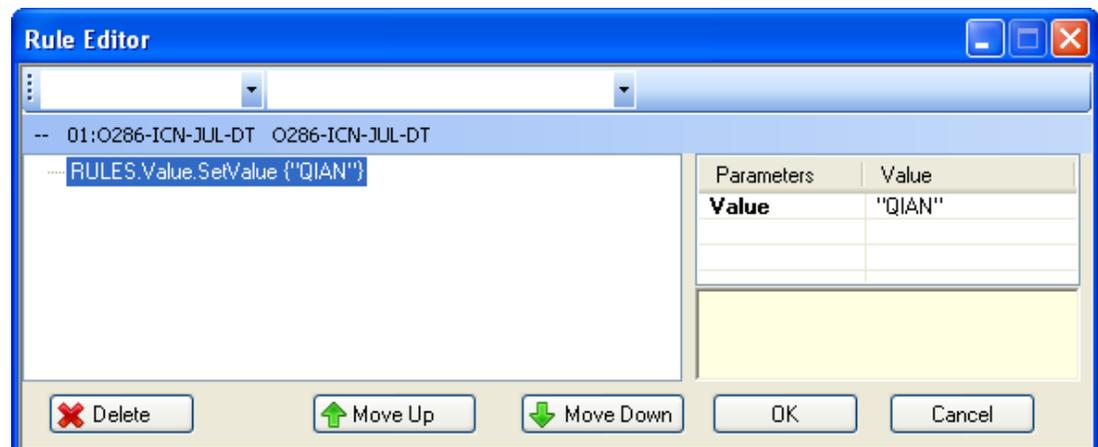
Value

Where:

Value The new value – a reserved word, variable, or literal in double quotes.

Example

This uses QIAN for the element if the segment has a qualifier.



SumFromArray

This rule adds (sums) numeric values in an array and saves the result to a variable.

Format of Parameters

ArrayName *Column* *Format* *ResultVar*

Where:

ArrayName Name of an existing array that contains the values. This must be a literal.

Column The column containing the values that are to be returned. This can be a variable or a literal in double quotes and the value can be a number, C for current column, or N for next column. Column indices start with 0.

Format Selected format
%[flags][TotalLength][.precision][DecimalLength]DataFormat. [totalwidth] [length] [specifier]

Where:

totalwidth = the total string length

length = the number of digits after the decimal

specifier = f (floating point), d (integer), or s (string)

Example: %10.2f,%10.3f

ResultVar Variable to hold the result.

Rules.Variable

Add

This rule adds one numeric value to another.

Format of Parameters

Value *ResultVar*

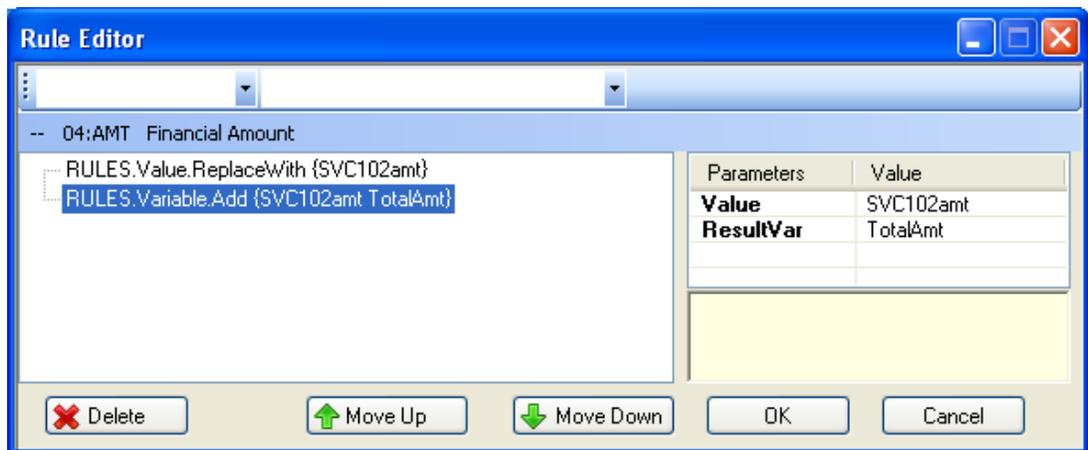
Where:

Value The amount to add to ResultVar: a variable, Current_Element, or literal in double quotes.

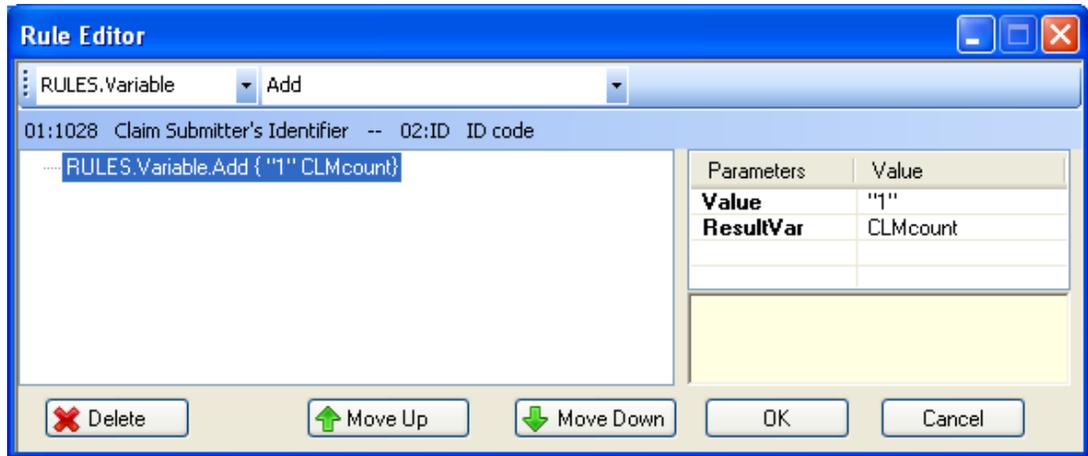
ResultVar Variable to hold the result.

Examples

The first rule uses the contents of SVC102amt in the output. The second rule adds the value in SVC102amt to a variable TotalAmt.



This rule adds 1 to the contents of the variable CLMcount.



Delete

This rule deletes a variable.

Format of Parameters

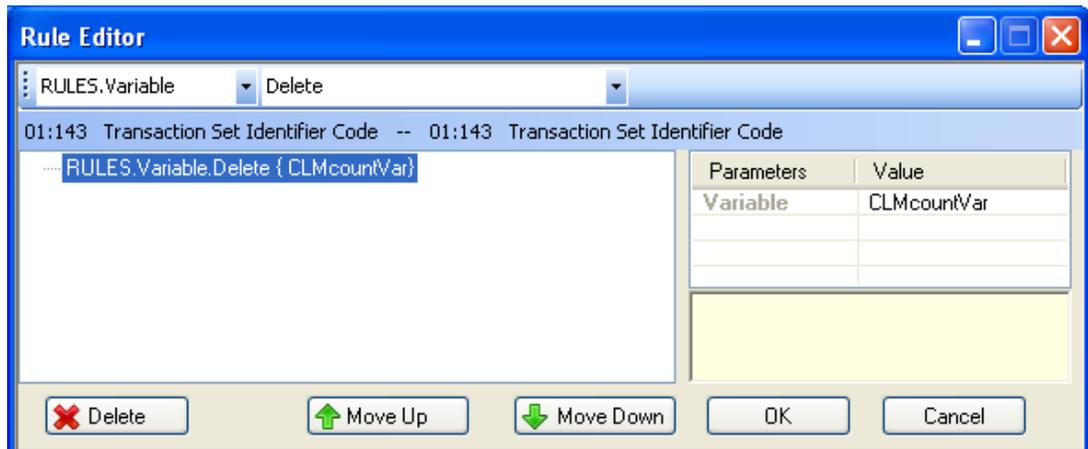
Variable

Where:

Variable A variable to delete. If omitted, all variables are deleted. If the variable does not exist, no action is taken.

Example

This rule on the ST01 element deletes CLMcountVar each time a new transaction set starts.



Divide

This rule divides two numbers, specifies how many decimal places to include, and puts the results in a variable.

Format of Parameters

VarA VarB Decimal ResultVar

Where:

VarA A variable containing the value to be divided (the dividend).

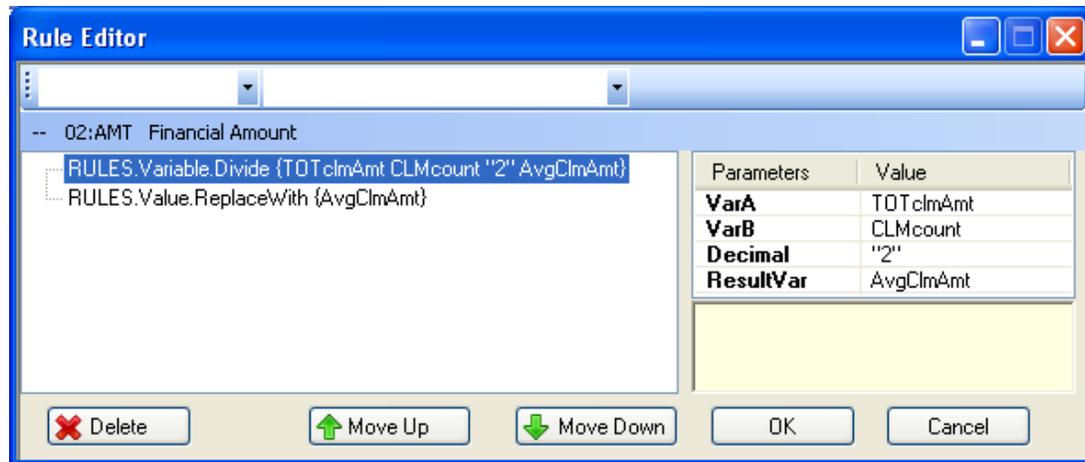
VarB A variable containing the number that *VarA* is to be divided by (the divisor). If this is zero, no action is taken.

Decimal Number of decimal places to use in the result. If this is zero, no action is taken.

ResultVar Variable to contain the result.

Example

This rule divides the value in *TotclmAmt* by the value in *CLMcount* and puts the result, with two decimal places, in variable *AvgClmAmt*.



DumpAllVariables

This rule lists the contents of all variables to the log.

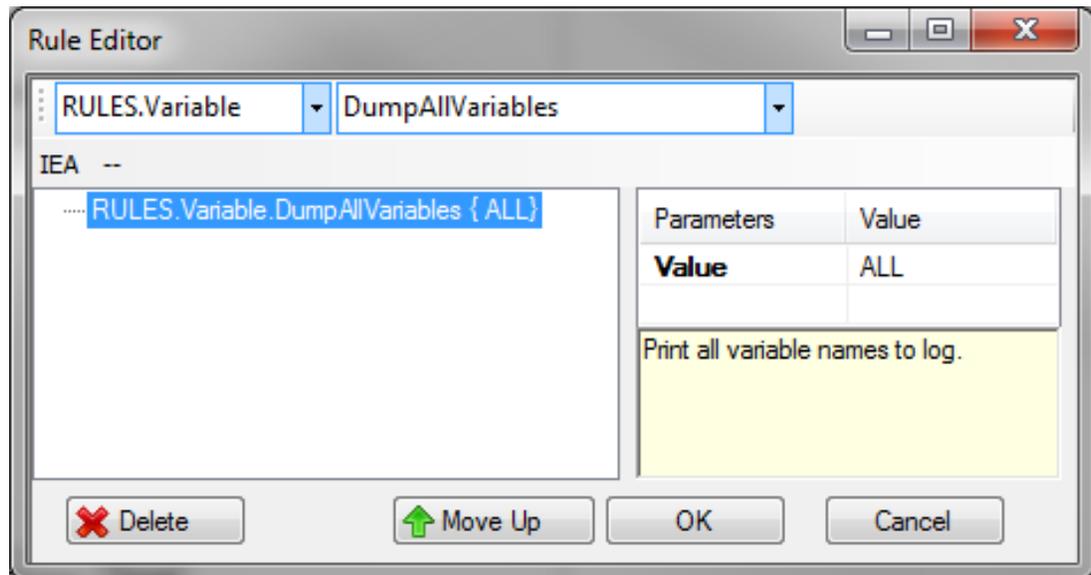
Format of Parameters

Value

Where:

Value Must be set to ALL.

Example



Example output

```
Size=4  
[GSDateVar],<20070818>  
[TotalPriceVar],<1064.75>  
[UnitPriceVar],<42.59>  
[UnitsVar],<25>
```

DumpVariables

This rule lists the contents of the specified variable(s) to the log.

Format of Parameters

Value

Where:

Value One or more variables, separated by spaces.

Example output

UnitsVar=[25]

GetEnvironmentVariable

This rule allows you to assign an environment variable to a local variable. For more information contact TIBCO Foresight Support.

Format of Parameters

Value ResultVar

Where:

Value Assigns an environment variable to a local variable.

ResultVar Variable to contain the result.

GetVariableSize

Debugging rule for Translator.exe. It prints out the number of variables that have been created.

Init

This rule removes the contents of one or more variables.

Format of Parameters

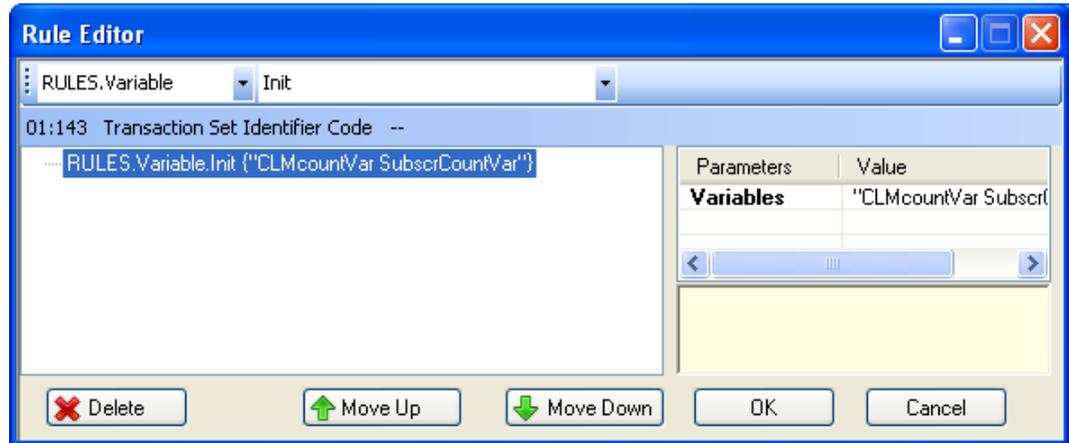
Variables

Where:

Variables One or more variables, each separated by a single space.
The list of variables is surrounded by double quotes. The maximum number of variables is 100.

Example

This rule clears out content from two variables.



IsAllSpace

This rule determines if a variable contains all spaces. It requires a block of sub-rules to take action based on the results.

Format of Parameters

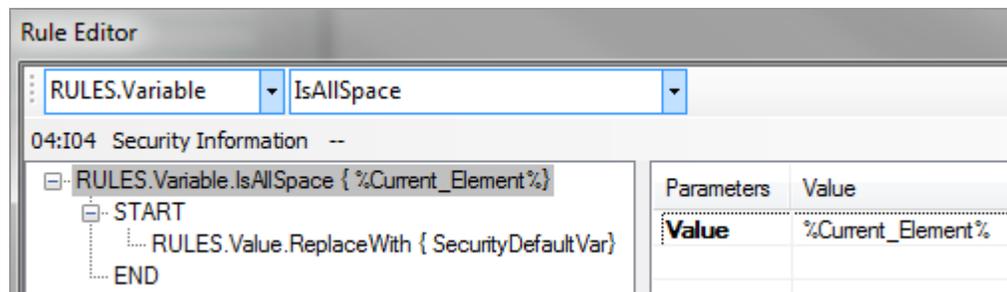
Value

Where:

Value The variable that might or might not contain all spaces.

Example 1

If the current element contains all spaces, then the output is the contents of variable SecurityDefaultVar.



Example 2

If the current element is all spaces, then put 0.000 in the output. Otherwise, insert a decimal in the source value and use it in the output:

```
RULES.Variable.IsAllSpace { %Current_Element% }
├─ START
│   └─ RULES.Variable.Set { "0.000" CUR03 }
│   └─ RULES.Value.ReplaceWith { CUR03 }
├─ ELSE
│   └─ RULES.Format.SetDecimal { %Current_Element% 3 CUR03 }
│   └─ RULES.Value.ReplaceWith { CUR03 }
```

IsEmpty

This rule determines if a variable is empty. It requires a sub-rule to take action based on the results.

Format of Parameters

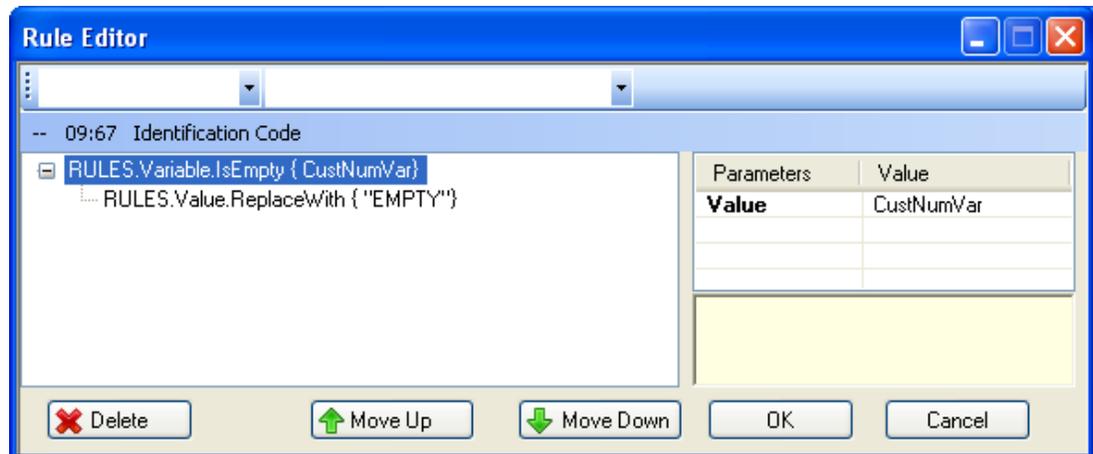
Value

Where:

Value The variable that might or might not be empty.

Example

This rule checks CustNumVar to see if it is empty. If so, it uses the literal “EMPTY” in the output.



Multiply

This rule multiplies two variables together and puts the result in a third variable.

Format of Parameters

VarA VarB ResultVar

Where:

VarA Variable containing the first value to multiply.

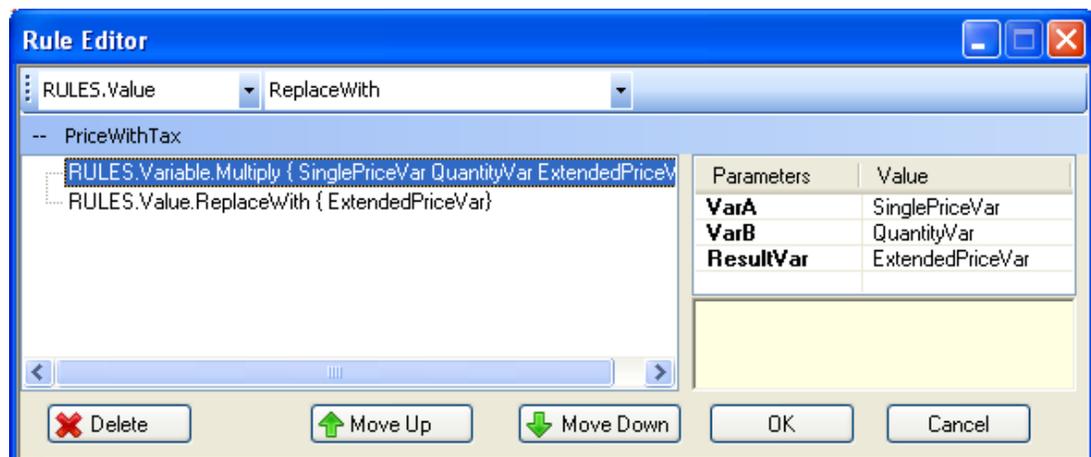
VarB Variable containing the second value to multiply.

ResultVar Variable to hold the product of the multiplication.

Example

This rule multiplies the contents of SinglePriceVar and QuantityVar and puts the product in ExtendedPriceVar.

The second rule uses the calculated value in the output.



Set

Places a value in a variable.

Format of Parameters

Value ResultVar

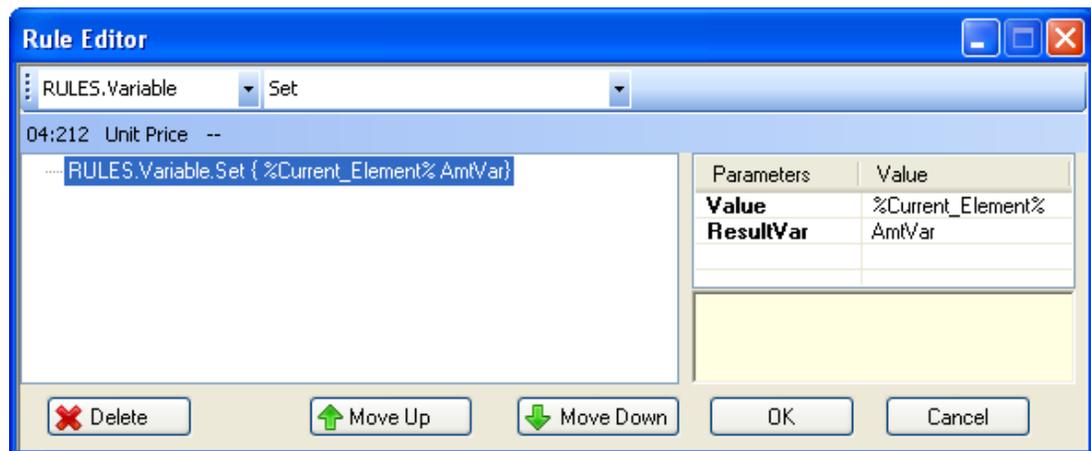
Where:

Value Value to put into VarA. This can be another variable, %Current_Element%, or a literal in double quotes.

ResultVar Variable to hold the value.

Example

This source rule assigns the value in the current element to variable **AmtVar**.



Subtract

Subtracts one variable from another and places the result in a variable.

Format of Parameters

VarA VarB ResultVar

Where:

VarA Starting value. If this is zero, no action is taken.

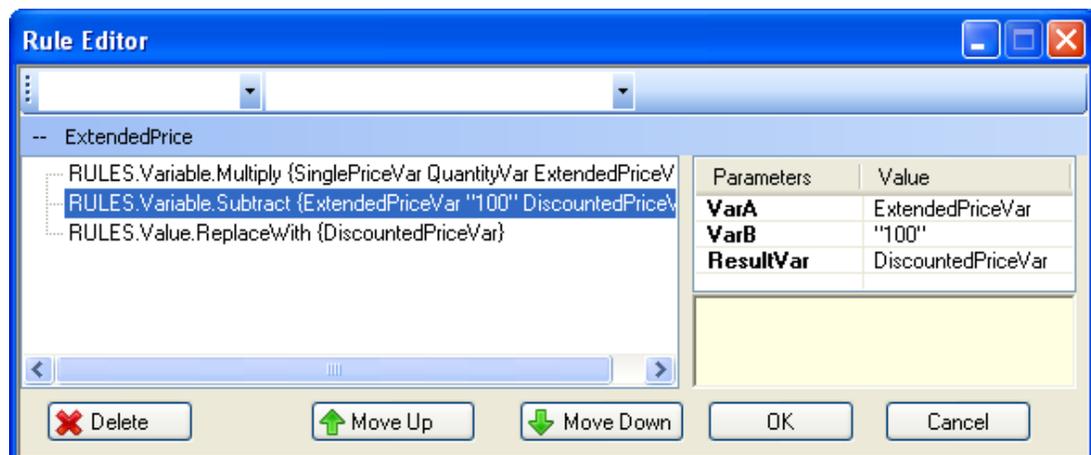
VarB Value to subtract from *VarA*. If this is zero, no action is taken.

ResultVar Variable to hold the result.

Example

The second rule subtracts 100 from the value in *ExtendedPriceVar* and places it in *DiscountedPriceVar*.

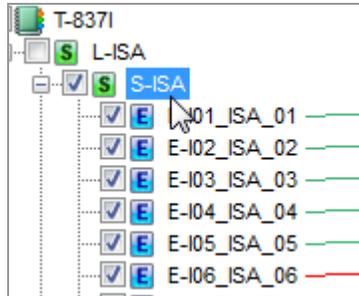
The third rule uses that value in the output.



Rules.XML

GetAttribute

Returns the value for an attribute from source XML. This rule works on “records” but not on “elements:”



Format of Parameters

VarA VarB ResultVar

Where:

VarA Starting value. If this is zero, no action is taken.

VarB Value to subtract from *VarA*. If this is zero, no action is taken.

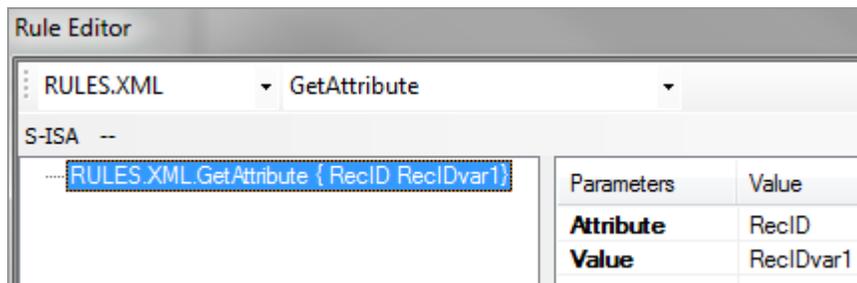
ResultVar Variable to hold the result.

Example

In Foresight Translator’s \DemoData directory, 837Iclean_xml.xml contains this:

```
<S-ISA RecID="HDRrec1" >  
attribute name ↗ ↖ attribute value
```

In DEMO4_XML_to_ED1_837I.map, this rule is on the S_ISA record:

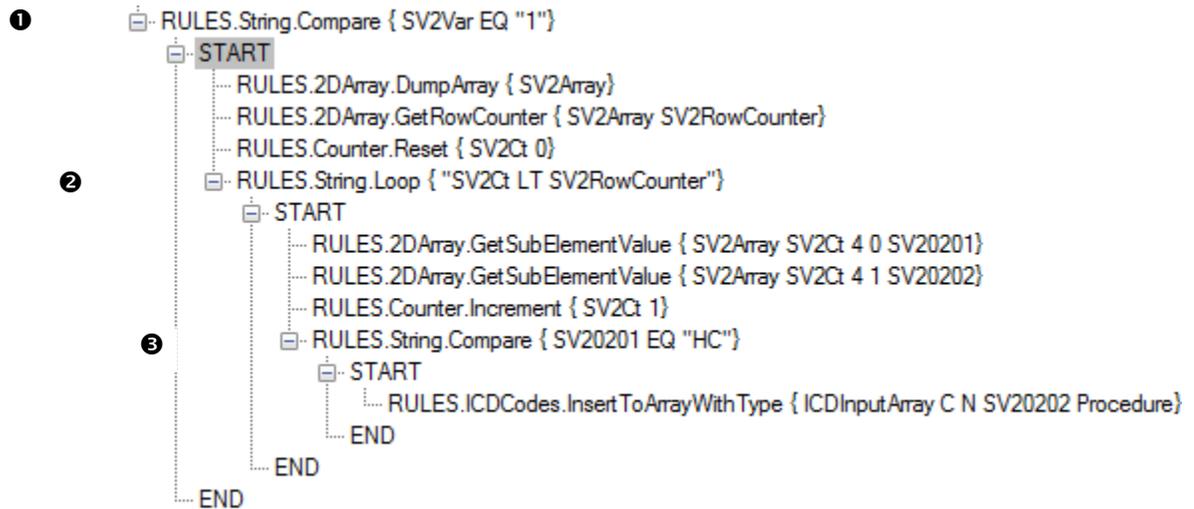


This rule will return HDRrec1 in variable RecIDvar1.

Conditional Rules

You can have rules that execute only if certain conditions are met. These conditional rules can be nested several layers deep.

This example has three layers of nesting:



- The first level starts with a String.Compare. If SV2Var contains 1, then the four rules that follow will execute.
- The second level executes repeatedly as long as its condition is true (SV2Ct is less than SV2RowCounter).
- The third level executes if the String.Compare condition is true (SV20201 equals HC).

Nesting can only take place based on a rule that contains a condition – like a String.Compare or a String.Loop. You couldn't start nesting on a rule that simply replaced a value, for example.

Text to avoid in nested rules

Do not use these uppercase key words in START-END rules, even if they are embedded in other words:

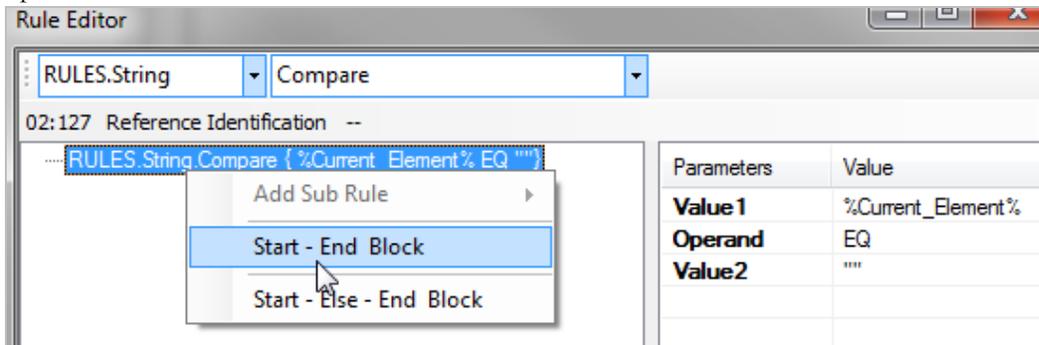
START
ELSE
END

Curly brackets: { }

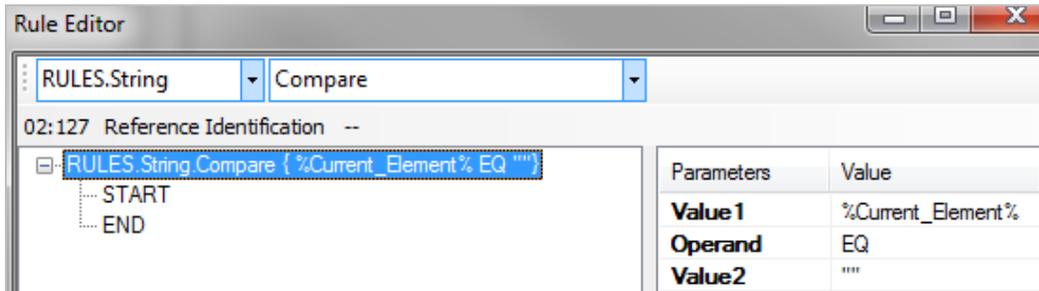
For example, do not use a variable called SENDERID since it contains END. You may call it SenderID, however.

How to set up nested rules

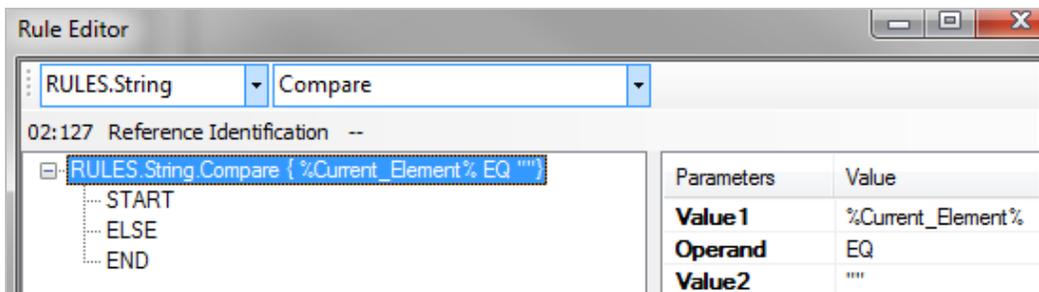
1. Right-click on a conditional rule, like this String.Compare and choose one of the options:



If you chose **Start – End Block**, you get a structure like this, where you can specify what rules to execute if the condition is true:

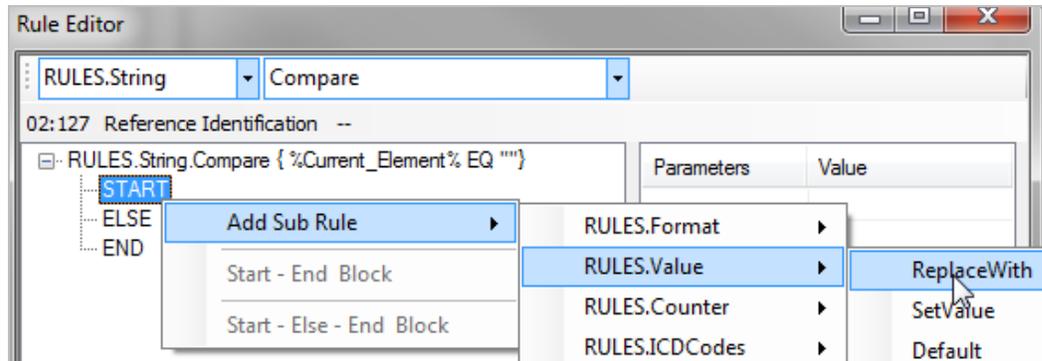


If you chose **Start – Else – End Block**, you get a structure like this, where you can specify what rules to execute if the condition is true, and also what rules to execute if the condition is false:



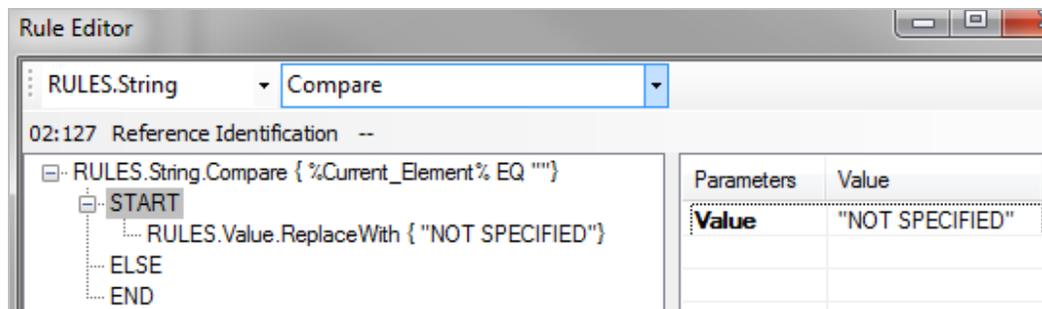
- Now, add the sub-rules:

Right-click on START and choose the rule that is to execute first if the condition is true:

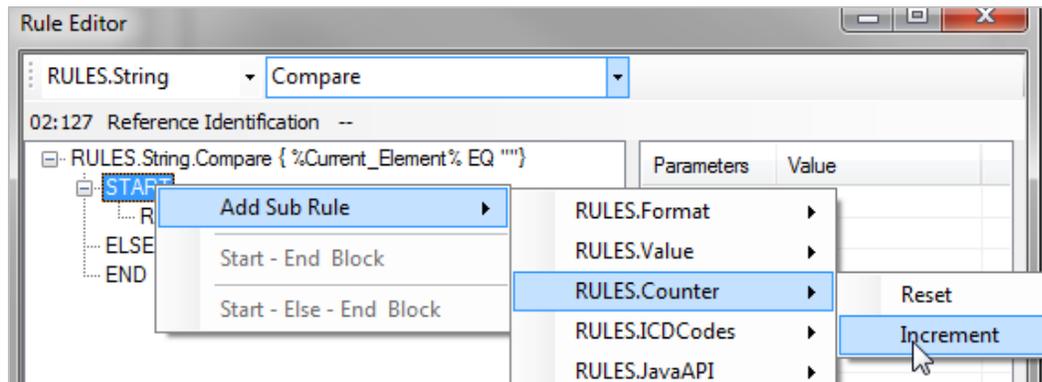


Add Sub Rule is only available on a START or ELSE.

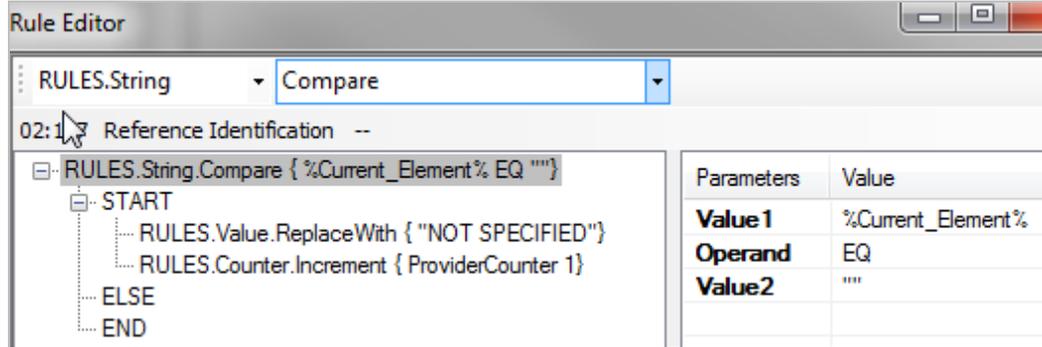
Example:



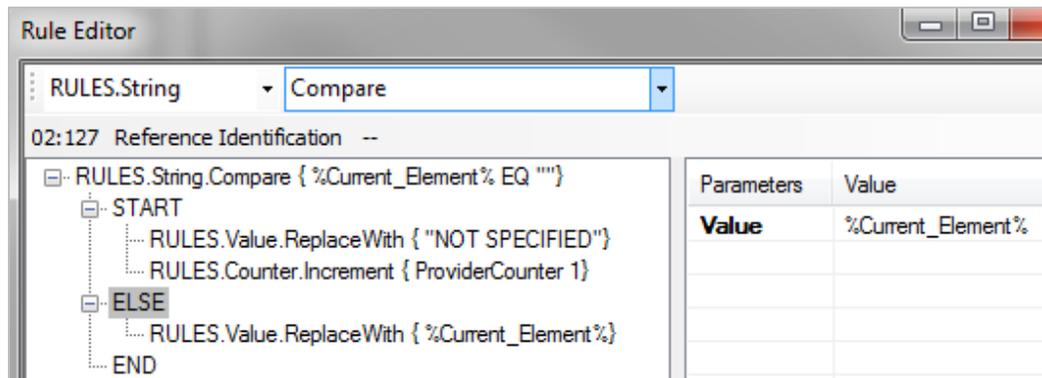
Continue right-clicking on START and adding rules that are to execute if the condition is true:



Example:



If you chose ELSE, right-click on ELSE and add rules that are to execute if the condition is false.



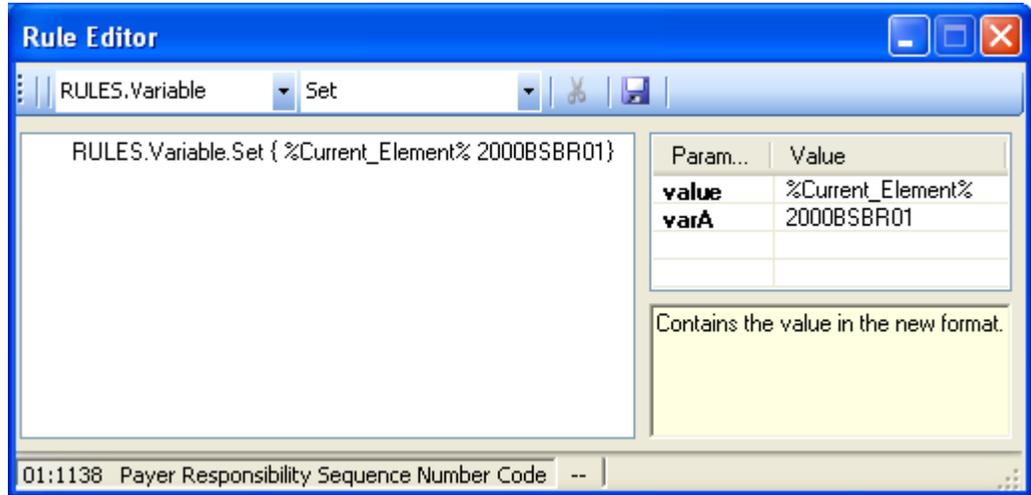
Example

Desired result:

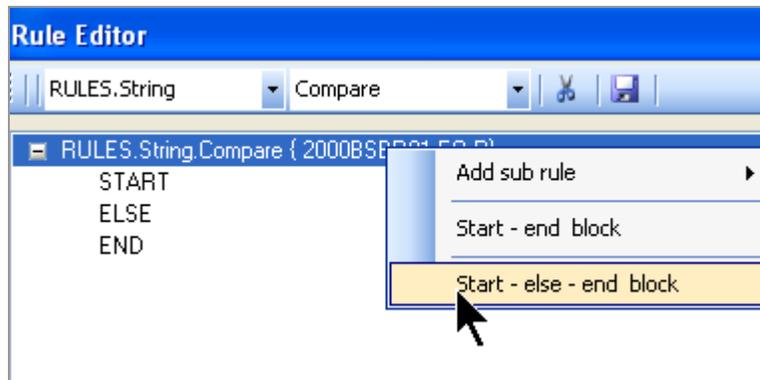
- If the contents of variable 2000BSBR01 is P, put nothing in the target SBR05.
- Otherwise, put the source value in the target.

Strategy:

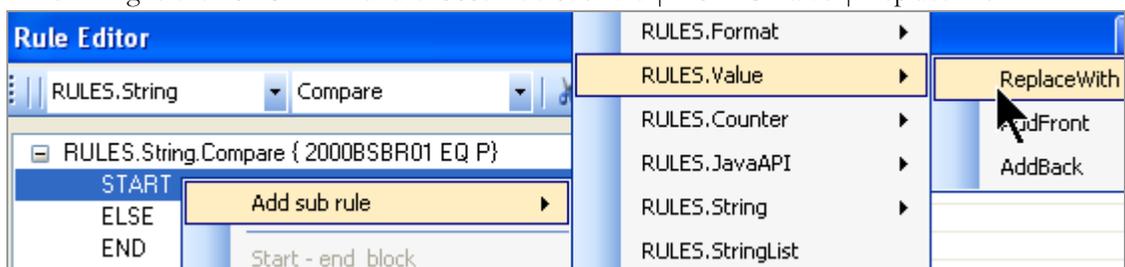
1. Map the SBR01 and SBR05.
2. On the SBR01, create a source rule to capture the current value in variable 2000BSBR01:



3. On the SBR05, create a String Compare rule to see if **2000BSBR01** equals **P**.
4. Right-click on the rule and choose **Start – else – end block**.



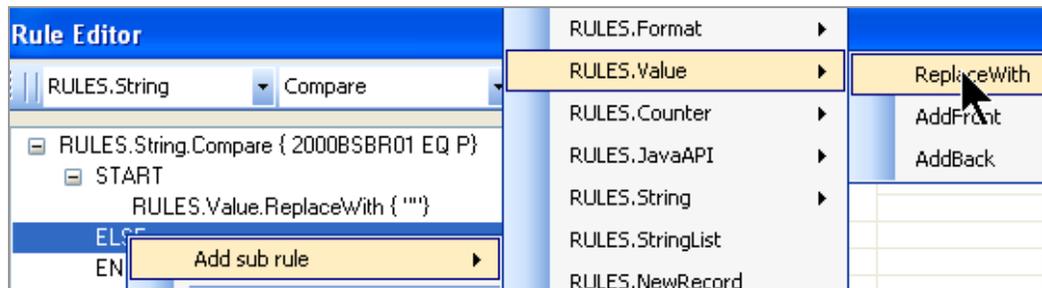
5. Right-click on **START** and choose **Add sub rule | RULES.Value | ReplaceWith**:



Type "" (two consecutive double quotes) for the value.

Param...	Value
value	""

- Right-click on **ELSE** and add a sub-rule to use the value in the **Current_Element**:



Param...	Value
value	Current_Element

Short Examples

- If the variable 2000BSBR01 contains P, put nothing in the target element. Otherwise, put the source value in the target element.

```

RULES.String.Compare { 2000BSBR01 EQ P}
  START
    RULES.Value.ReplaceWith { ""}
  ELSE
    RULES.Value.ReplaceWith { %Current_Element%}

```
- If the source's current element contains SY, use it in the target. Otherwise but nothing in the target element.

```

RULES.String.Compare { %Current_Element% EQ SY}
  START
    RULES.Value.ReplaceWith { SY}
  ELSE
    RULES.Value.ReplaceWith { ""}
  END

```
- This set of rules on the target GS08 checks for the desired value for an 837P or 837I.

If the source element contains 004010X096A1, use the value I in the target and set variable GSGOODvar to TRUE.

If the source element contains 004010X098A1, use the value P in the target and set variable GSGOODvar to TRUE.

If variable GSGOODvar doesn't contain TRUE, put the value Error... in the target.

```

❏ RULES.String.Compare { %Current_Element% EQ 004010X096A1}
  ❏ START
    RULES.Value.ReplaceWith {}
    RULES.Variable.Set {GSGOODvar TRUE}
  END
❏ RULES.String.Compare { %Current_Element% EQ 004010X098A1}
  ❏ START
    RULES.Value.ReplaceWith {P}
    RULES.Variable.Set {GSGOODvar TRUE}
  END
❏ RULES.String.Compare { GSGOODvar NE TRUE}
  ❏ START
    RULES.Value.ReplaceWith {Error...}
  END
```

11 Appendix A: Tutorials

List of Tutorials

Tutorial	Page
Mapping Tutorial 1 – EDI to XML	281
Mapping Tutorial 2 – EDI to Fixed Length Flat File	295
Mapping Tutorial 3 – EDI to EDI	312

Important

The tutorials in this section involving XML schemas (XSD files) illustrate very simple scenarios in which the EDI structure and the XML schema structure match exactly and direct mapping can be used.

Please note that XML schemas generated by means other than the EDISIM Standards Editor can be extremely complex and their use may require advanced business rules and assistance from TIBCO Foresight Support.

Refer to Appendix E: XML Schemas for more information.

Mapping Tutorial 1 - EDI to XML

Files used in Tutorial		
Purpose	File	Notes
Source	DEMO_4010_850.std (company guideline)	In Foresight Translator's \Database directory Created by Foresight Translator installation
Target	DEMO_Order.xsd (XML schema)	In Foresight Translator's \Database directory Created by Foresight Translator installation
Map	DEMO_850_to_XML.map	Create during tutorial For reference, a completed map called DEMO_850_to_XML_DONE.map is already in the Database directory
File to translate	TranslatorDemo_850_4010_X_1.txt	In Foresight Translator's \DemoData directory Created by Foresight Translator installation

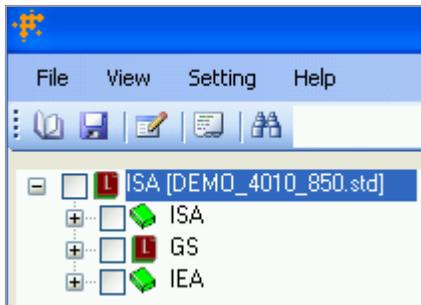
Opening Source and Target

1. Open **Translation Tool.exe** in Foresight Translator's \Bin directory.
2. Choose File | Open Source Guideline.



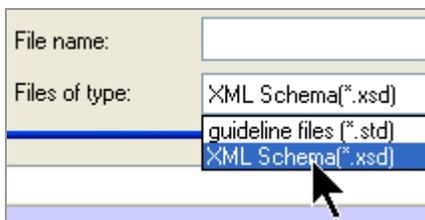
Choose **DEMO_4010_850.std** in Foresight Translator's \Database directory.

It will look like this in Translation Tool:



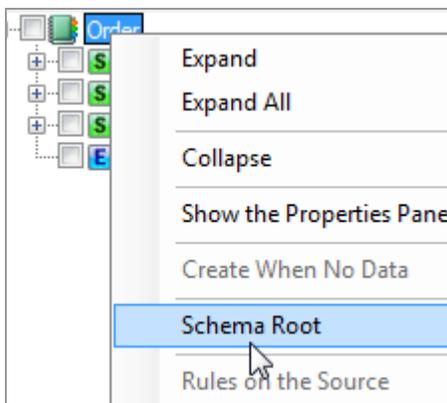
3. Choose **File | Open Target Guideline** and go to Foresight Translator's \Database directory.

For file type, chose **XML Schema**:

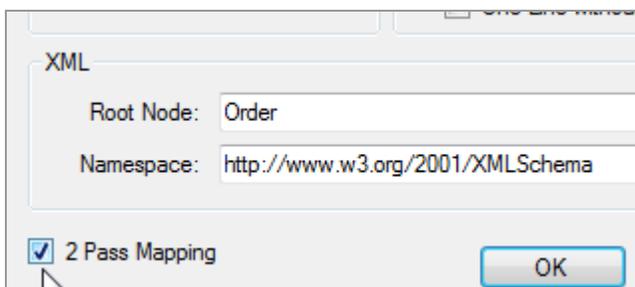


Open DEMO_Order.xsd.

4. Right-click on **Order** at the top right and set it as the **Schema Root**:



5. Since the source and target structures are so different, go to **Settings | Target Guideline Settings...** and turn on 2 Pass Mapping:



6. Save it as a map file in Foresight Translator's \Database directory:

Choose **File | Save As |** <go to Foresight Translator's Database directory> | **DEMO_850_to_XML.map.**

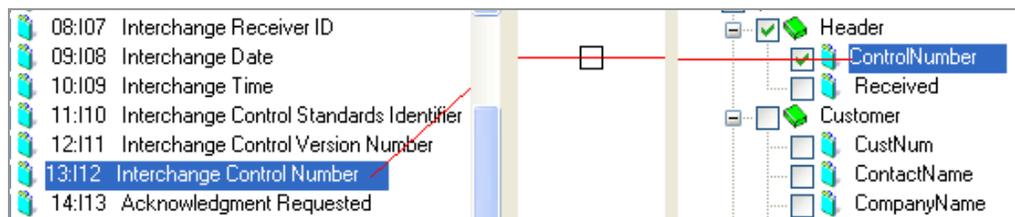
Maps must be stored in this directory.

Mapping Data

1. Open the **ISA** segment on the left (source) and **Header** on the right (target).



2. Drag from ISA13 Interchange Control number to ControlNumber, like this:



This tells Foresight Translator to put data from the ISA13 into the XML's ControlNumber element.

3. If the line connecting the two is red, it means that it is the selected connection. Translation Tool will not let you scroll away from a selected item. In this case, click on an unmaped item on the right so that all lines are unselected (green).

Now you can scroll freely.

Looking at Properties

We are looking for an 8-digit date to put into the target's Received element.

1. Click **ISA09** Interchange Date.
2. If the Properties panel is not displaying at the bottom, click this toolbar button:



3. In the Properties pane at the bottom, be sure that the **Properties** tab is selected.

Element ID [I08]	
Description	Interchange Date
Purpose	Date of the interchange
MinMax	06/06
Type	DT
Ordinal	9
Doc Desc	

Unmapped Properties Rules Test

4. Scroll down in the properties pane to see the MinMax lengths:

MinMax	06/06
Type	DT

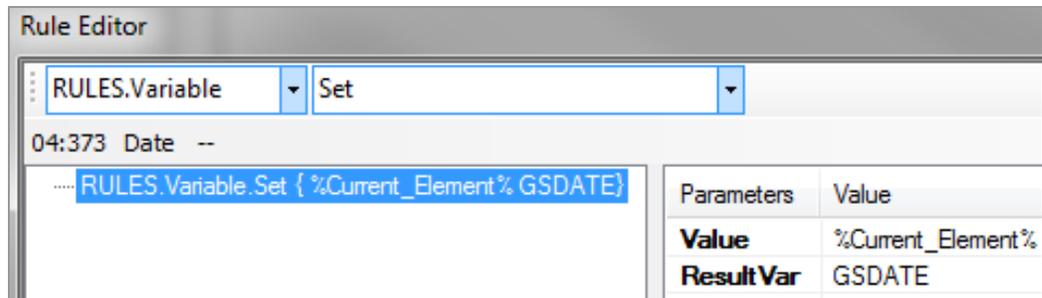
We decide not to use it. We want 8 digits.

5. In the top pane, open the **GS loop** and the **GS segment**, click the **GS04**, and notice that it's MinMax is 08/08.

Appending Values

We want the GS04 and the GS05 combined into the **Received** target field.

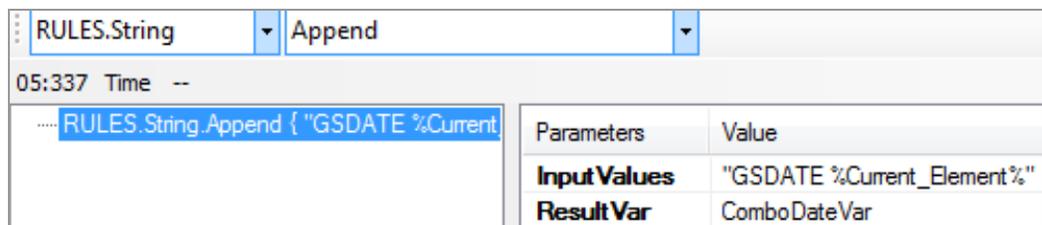
1. Right-click on the GS04 and choose **Rules on the source**.
2. Click **Rule Editor** at the bottom of the box, and set up and save this rule:



This captures the incoming date value in variable GSDATE.

Save the rule.

3. Add and save this source rule to the GS05:

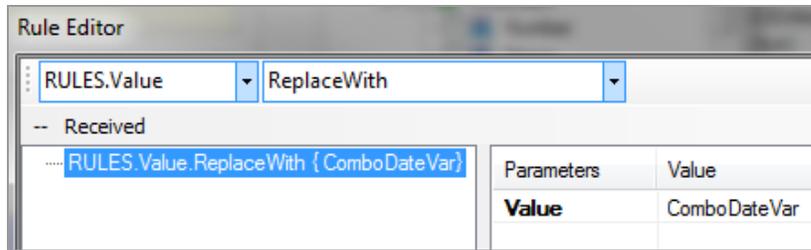


If the rule box toolbar disappears off the top of the screen, right-click on the rule and choose **Save**.

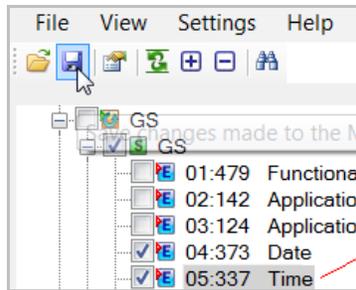
4. Drag from the target's Received field in the target to the middle to get a line like this:



5. Click on the box in the center and create this target rule:



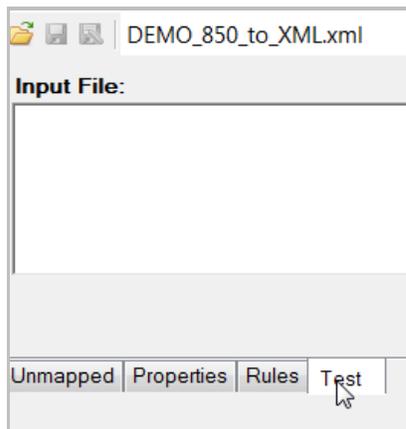
6. Save your rule and the map.



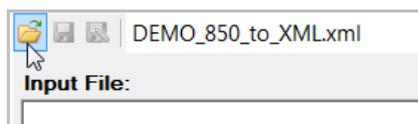
Testing your Map

To test the fields we have mapped so far:

1. In the Properties pane at the bottom, choose the **Test** tab.



2. Click the **Open input file** button:



In Foresight Translator's \DemoData directory, select **TranslatorDemo_850_4010_X_1.txt**.

This is an 850 EDI file to be translated.

3. Ensure that the map file name is showing to the right of the Open input file button.
4. Click the **Run** button:



The mapped elements should appear in the output area at the bottom right. Notice the combined date and time in <Received>.

```

    <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  - <Order xmlns="http://www.foresightcorp.com/EXSD/v10"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <Header>
    <ControlNumber>000000001</ControlNumber>
    <Received>200708180900</Received>
    </Header>
    <Total />
  - </Order>

```

The Customer Complex Element

1. On the target side, open **Customer** and its **Address** element.
2. Drag these connections:

Source	Target
GS02	ContactName
310 N104	CustNum
310 N102	CompanyName
330 N301	Street
340 N401	City
340 N403	ZIP
340 N402	State

3. Save and test.

You should see the newly mapped elements in the output:

```

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  - <Order fs:noNamespaceSchemaLocation="DEMO_Order.xsd"
      xmlns:fs="http://www.w3.org/2001/XMLSchema-instance">

```

```

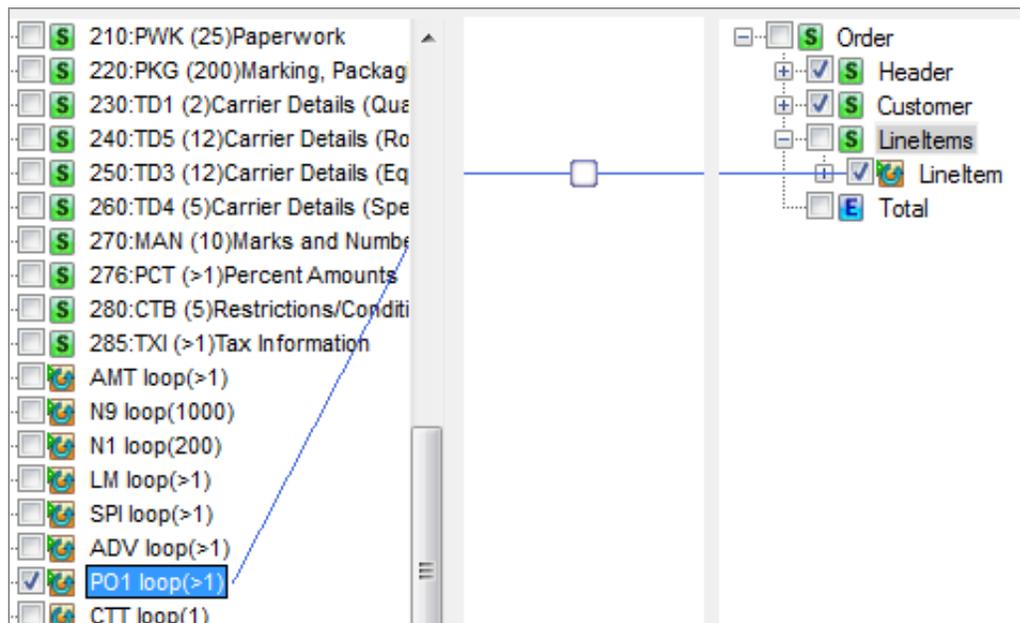
= <Header>
  <ControlNumber>000000001</ControlNumber>
  <Received>2007-08-18T09:00:00</Received>
</Header>
= <Customer>
  <CustNum>112221234</CustNum>
  <ContactName>KAVERCORP</ContactName>
  <CompanyName>KAVER CORPORATION, INC.</CompanyName>
= <Address>
  <Street>682 FALL RIVER PLACE</Street>
  <City>DAMASCUS</City>
  <ZIP>20872</ZIP>
  <State>MD</State>

```

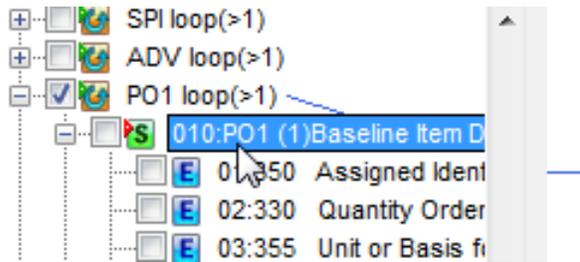
Repeating Loops

We need to specify which source repeating loops correspond to which target repeating loops.

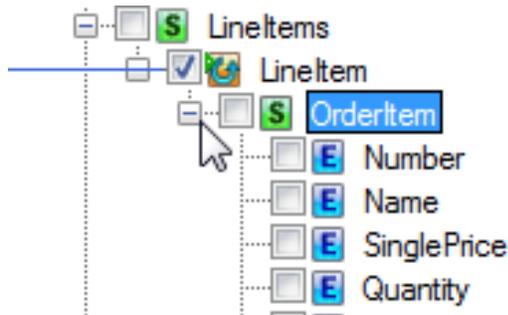
1. In the target, open **LineItems**.
2. Drag the **PO1 loop** to the **LineItem** loop. These correspond to one another.



- In the source, open the **PO1** segment:



- In the target, open **OrderItem**:



- Drag these connections:

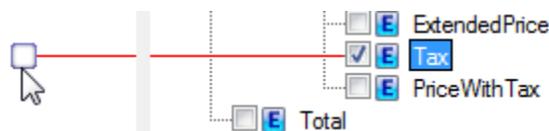
Source	Target
PO107	Number
PO102	Quantity
PO104	SinglePrice

- Save.
- Test to be sure you get Number, Quantity, and SinglePrice three times in your output.

Inserting a Literal Value

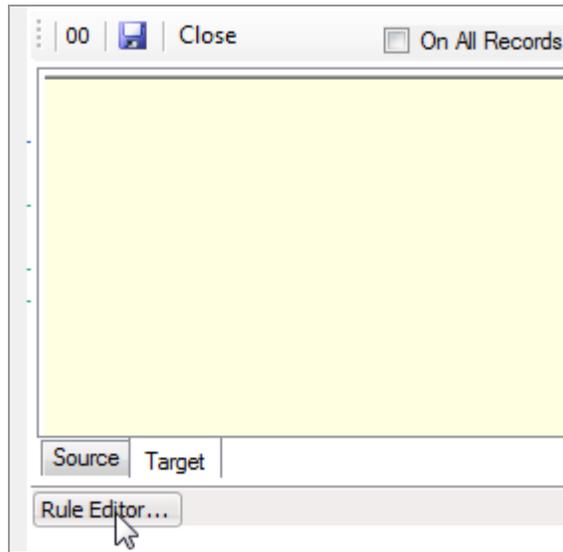
Set up the **Tax** element to contain the literal value .0675 rather than a value from the input data:

- Drag from **Tax** to the middle to get a line like this:

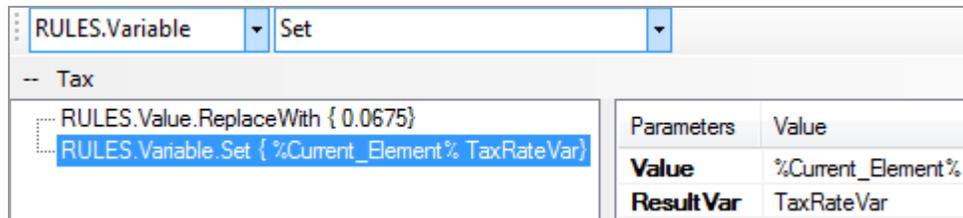


- Click on the little box to enter the rule editor.

3. Click **Rule Editor...** at the bottom:



4. Set up these rules:



The first gives a literal value that always goes in this field. The tax rate is 6.75%.

The second saves this rate in variable `TaxRateVar` so that it can be used in a calculation later.

5. Save the rules and the map.
6. Test the rule to be sure you get the value 0.0675 in this element each time.

Calculations

Extended Price

We want the target's **ExtendedPrice** element to contain this calculation:

$$\text{SinglePrice} * \text{Quantity}$$

1. Add this target rule to the **SinglePrice** field:

RULES.Variable	Set						
04:212 Unit Price -- SinglePrice							
.....RULES.Variable.Set { %Current_Element% SinglePriceVar}	<table border="1"> <tr> <th>Parameters</th> <th>Value</th> </tr> <tr> <td>Value</td> <td>%Current_Element%</td> </tr> <tr> <td>Result Var</td> <td>SinglePriceVar</td> </tr> </table>	Parameters	Value	Value	%Current_Element%	Result Var	SinglePriceVar
Parameters	Value						
Value	%Current_Element%						
Result Var	SinglePriceVar						

2. Add this target rule to the **Quantity** field:

RULES.Variable	Set						
02:330 Quantity Ordered -- Quantity							
.....RULES.Variable.Set { %Current_Element% QuantityVar}	<table border="1"> <tr> <th>Parameters</th> <th>Value</th> </tr> <tr> <td>Value</td> <td>%Current_Element%</td> </tr> <tr> <td>Result Var</td> <td>QuantityVar</td> </tr> </table>	Parameters	Value	Value	%Current_Element%	Result Var	QuantityVar
Parameters	Value						
Value	%Current_Element%						
Result Var	QuantityVar						

3. Add these target rules to the **ExtendedPrice** field:

RULES.Variable	Multiply								
04:212 Unit Price -- ExtendedPrice									
.....RULES.Variable.Multiply { SinglePriceVar QuantityVar}	<table border="1"> <tr> <th>Parameters</th> <th>Value</th> </tr> <tr> <td>VarA</td> <td>SinglePriceVar</td> </tr> <tr> <td>VarB</td> <td>QuantityVar</td> </tr> <tr> <td>Result Var</td> <td>ExtendedPriceVar</td> </tr> </table>	Parameters	Value	VarA	SinglePriceVar	VarB	QuantityVar	Result Var	ExtendedPriceVar
Parameters		Value							
VarA	SinglePriceVar								
VarB	QuantityVar								
Result Var	ExtendedPriceVar								
.....RULES.Value.ReplaceWith { ExtendedPriceVar}									

- In the first rule, we create variable **ExtendedPriceVar** that contains the quantity times the single price.

The second rule actually places the contents of ExtendedPriceVar into the ExtendedPrice element.

- Save the rules.
- Test.

You should see the ExtendedPrice field that is calculated from SinglePrice times Quantity.

Tax

We now need to calculate the tax and add it to the extended price to get the PriceWithTax.

We already specified the tax rate at 0.0675 and put this value in TaxRateVar.

- Target rules on PriceWithTax field:

RULES.Variable	Multiply	Parameters	Value
-- PriceWithTax			
RULES.Variable.Multiply { ExtendedPriceVar TaxRateVar		VarA	ExtendedPriceVar
RULES.Variable.Add { TaxAmtVar ExtendedPriceVar}		VarB	TaxRateVar
RULES.Value.ReplaceWith { ExtendedPriceVar}		Result Var	TaxAmtVar

- Test.

You should see data like this:

```

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <Order xmlns="http://www.foresightcorp.com/EXSD/v10"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <Header>
<ControlNumber>000000001</ControlNumber>
<Received>200708180900</Received>
  </Header>
- <Customer>
<CustNum>112221234</CustNum>
<ContactName>KAVERCORP</ContactName>
<CompanyName>KAVER CORPORATION, INC.</CompanyName>
- <Address>
<Street>682 FALL RIVER PLACE</Street>
<City>DAMASCUS</City>
<ZIP>20872</ZIP>
<State>MD</State>
  </Address>
  </Customer>
- <LineItems>
- <LineItem>
- <OrderItem>
<Number>11111</Number>
<Name />

```

```

<SinglePrice>20.00</SinglePrice>
<Quantity>15</Quantity>
<ExtendedPrice>300.00</ExtendedPrice>
<Tax>0.0675</Tax>
<PriceWithTax>320.25</PriceWithTax>
  </OrderItem>
  </LineItem>
- <LineItem>
- <OrderItem>
<Number>22222</Number>
<Name />
<SinglePrice>20.00</SinglePrice>
<Quantity>25</Quantity>
<ExtendedPrice>500.00</ExtendedPrice>
<Tax>0.0675</Tax>
<PriceWithTax>533.75</PriceWithTax>
  </OrderItem>
  </LineItem>
- <LineItem>
- <OrderItem>
<Number>33333</Number>
<Name />
<SinglePrice>18.00</SinglePrice>
<Quantity>25</Quantity>
<ExtendedPrice>450.00</ExtendedPrice>
<Tax>0.0675</Tax>
<PriceWithTax>480.37</PriceWithTax>
  </OrderItem>
  </LineItem>
</LineItems>
<Total />
</Order>

```

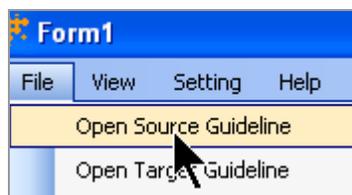
Mapping Tutorial 2 - EDI to Fixed Length Flat File

This tutorial uses a target that has fixed length, non-delimited fields.

Files used in Tutorial		
Purpose	File	Notes
Source	837AQ120.std (EDI guideline)	In Foresight Translator's \Database directory Created by Foresight Translator installation
Target	DEMO_CLAIMSUM_FF.std (flat file guideline)	In Foresight Translator's \Database directory Created by Foresight Translator installation
Map	DEMO_837AQ120_to_CLAIMSUM_FF.map	Create during tutorial For reference, a completed map called DEMO_837P_to_CLAIMSUM_FF_DONE.map is already in the Database directory
File to translate	TranslatorDemo8_4010_837P.txt	In Foresight Translator's \DemoData directory Created by Foresight Translator installation

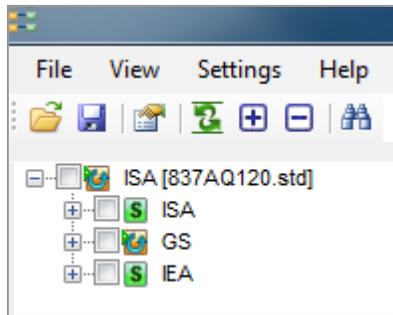
Open Source and Target Guideline

1. Open **Translation Tool.exe** in Foresight Translator's \Bin directory.
2. Choose File | Open Source Guideline.



Choose **837AQ120.std** in Foresight Translator's \Database directory.

It will look like this in Translation Tool:



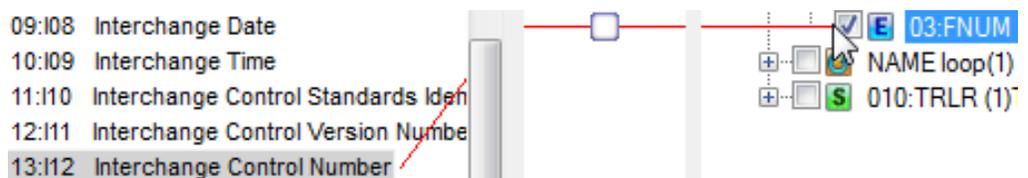
3. Choose **File | Open Target Guideline** and select **DEMO_CLAIMSUM_FF.std** from Foresight Translator's \Database directory.

Mapping Data

1. Open the ISA segment on the left (source) and the TRAN record on the right (target).



2. Drag from **ISA13 Interchange Control number** to TRAN03 FNUM, like this:



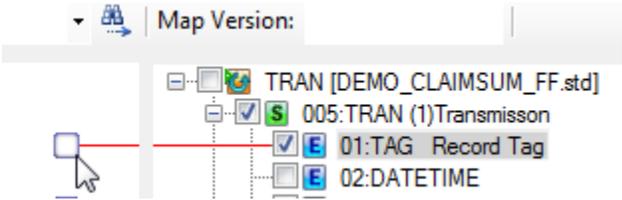
This shows that Foresight Translator should put data from the ISA13 into the TRAN03.

3. The line connecting the two is red, meaning that it is the selected connection and you cannot scroll it off the screen. Click on an unmapped item in the target so that this line is unselected (green). Now you can scroll freely.

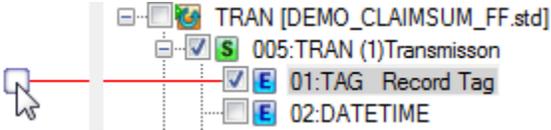
Inserting a Literal Value

To set up the segment tags, which are not in the source data:

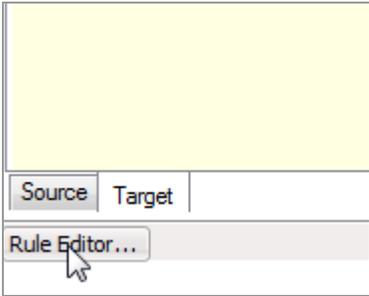
1. Drag from the TAG field at TRAN01 to the middle to get a line like this:



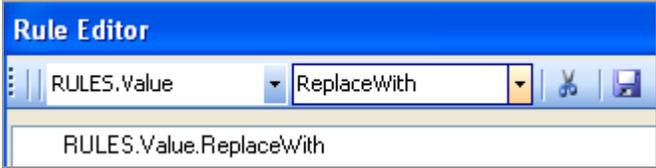
2. Click on the box to enter the rule editor:



3. Click **Rule Editor...** at the bottom:



4. In the first two fields, select these:

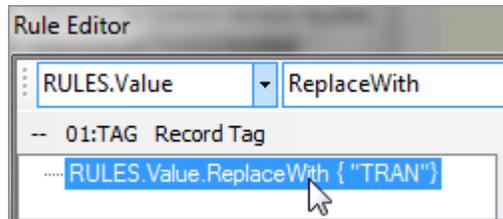


5. In the Parameter area to the right, type **"TRAN"**:

Parameters	Value
Value	"TRAN"

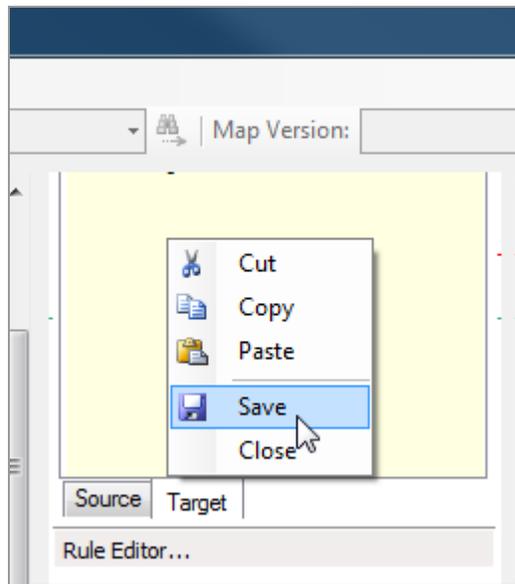
The quotes mean it is a literal value that always goes into this field.

6. Click on the rule in the left pane to be sure your parameter is added to it:



7. Click **OK** and then save the rule.

If the top of the rule box is off the screen, right-click and Save:



8. The box is now yellow, indicating a rule on the target side:



Saving your Map

1. Choose **File | Save As** and navigate to Foresight Translator's \Database directory.

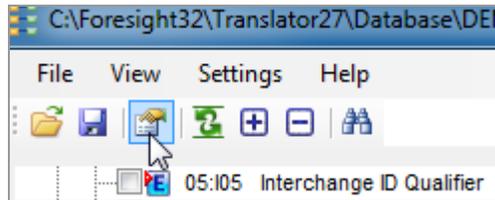
Maps must be stored there.

2. Name it DEMO_837AQ120_to_CLAIMSUM_FF.map

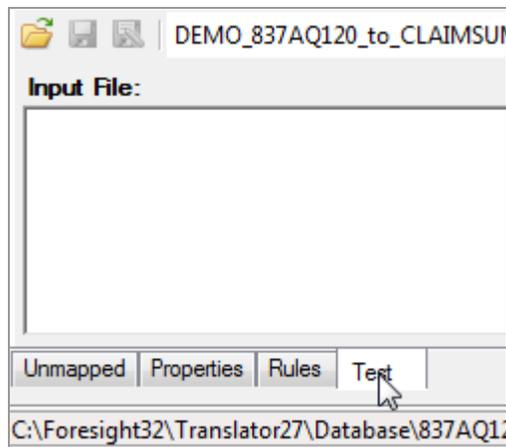
Testing your Map

To test the two fields we have mapped so far:

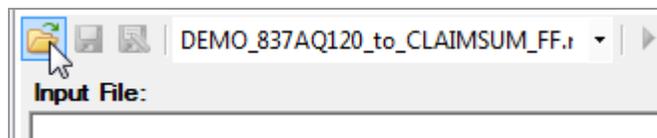
1. If the Properties panel is not displaying at the bottom, click this toolbar button:



2. In the Properties panel, choose the **Test** tab.

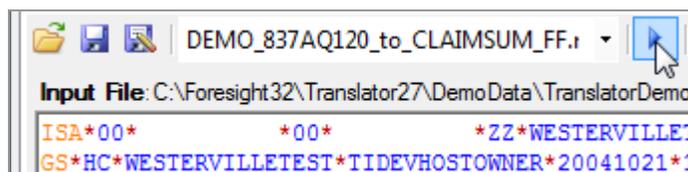


3. Click the **Open input file** button:



Select **TranslatorDemo_4010_837P.txt** in Foresight Translator's \DemoData directory.

4. Ensure that the map file name is showing to the right of the Open input file button.
5. Click the **Run** button:



Two fields should appear in the output area at the bottom right.

TRAN 000000001

Where:

- TRAN is the literal value placed in the field by our rule.
- The unmapped DATETIME field in the middle is blank.
- 000000001 is mapped from the ISA13.

Mapping two Fields into One

To map the ISA10 Interchange Date AND the ISA09 Interchange Time into the DATETIME field:

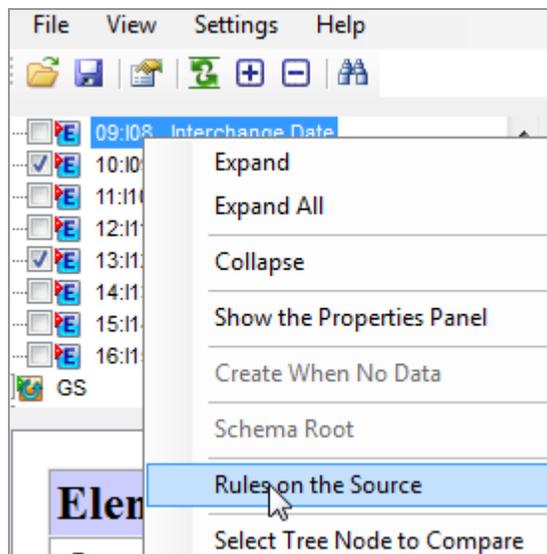
1. Drag the **ISA10** Interchange Time to **DATETIME**.

You now have the time in DATETIME.

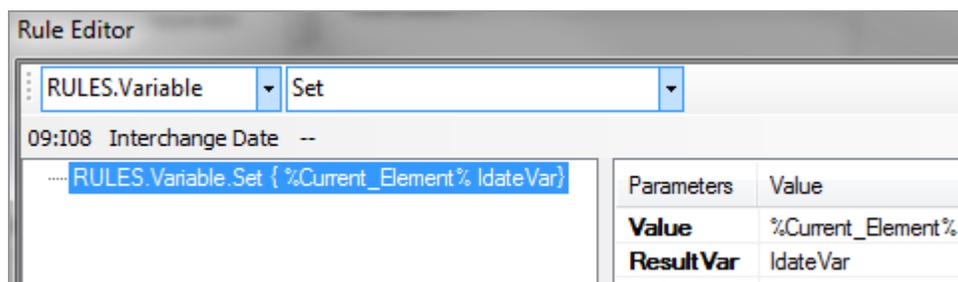
We want to add the ISA09 Interchange Date before it. This will require:

- A rule on the source ISA09 to capture its value.
- A rule on the target DATETIME to put the value in front of the time.

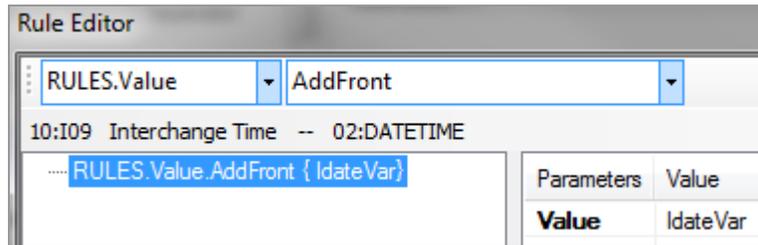
2. Right-click on Interchange Date and choose **Rules on the source**:



Create and save this rule to store the date in a variable called **Idate**:



- Open the rule editor for DATETIME and set up this target rule:



This adds the value in the Interchange date in front of the time that is already in DATETIME.

- Save and close the rule editor.
- Save the map.
- Test the rule. You should now have a date the time in the TRAN record:

TRAN 0410210944 000000001

The NAME Record

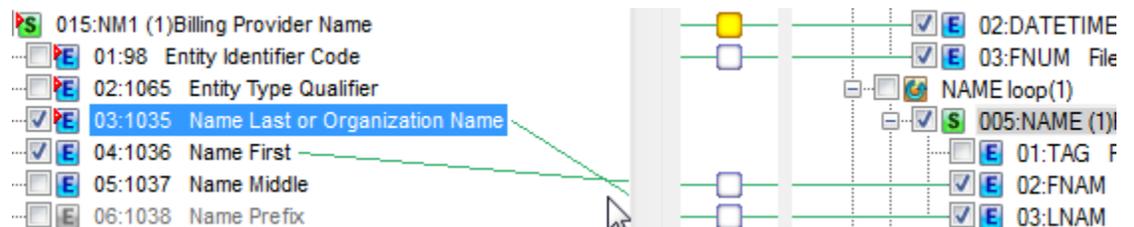
To populate the NAME record:

- On the source side, open the **GS**, **ST**, **2000A**, and **2010AA** loops and the **NM1** segment.

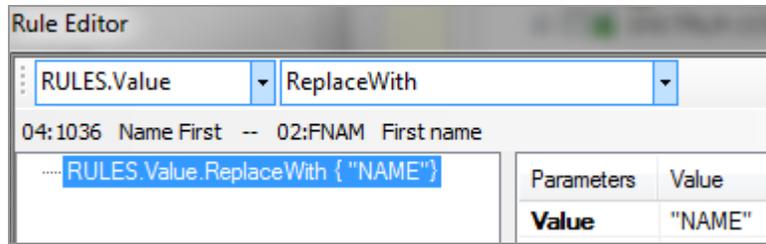
On the target side, open the NAME loop and the **NAME** record.

- Drag the **NM104** Name First to the **FNAM** field.
- Drag the **NM103** Name Last to the **LNAM** field.

The lines cross because the fields are in a different order in the source and target. This is OK since they are coming from the same segment.



- For the record tag, drag a line from the **TAG** field to the middle and add this rule. It always places the literal NAME in this field.



5. Save the rule and the map.

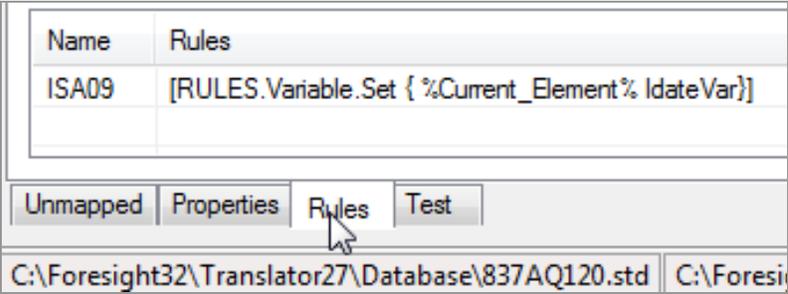
6. Test.

You should see a NAME record similar to this:

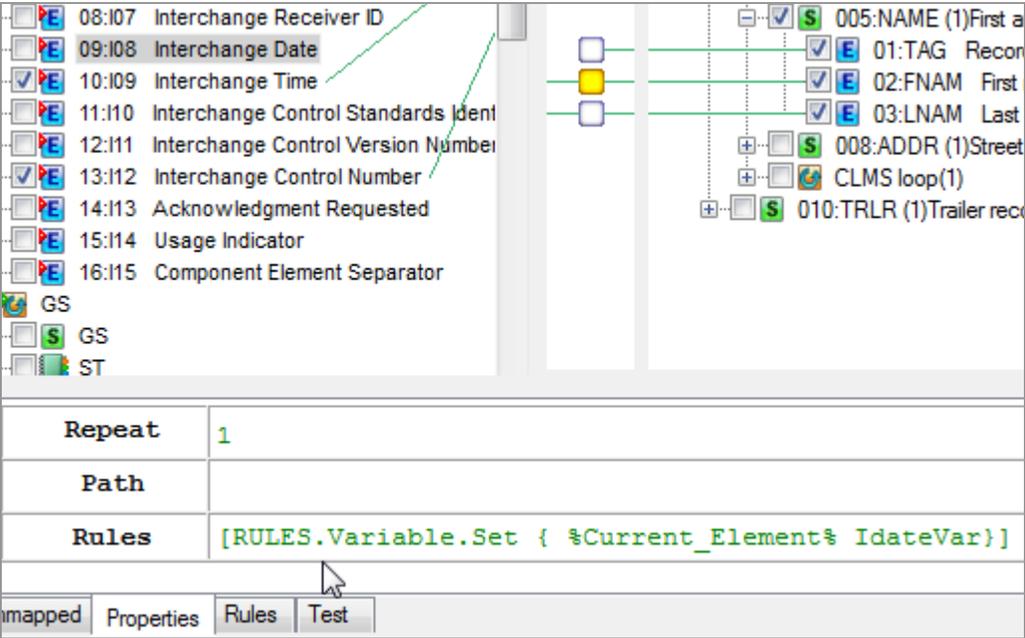
NAME MUFFY JONES

Listing Rules

1. To see a list of rules for all expanded levels of the guideline, click **Rules** at the bottom left of the properties pane:

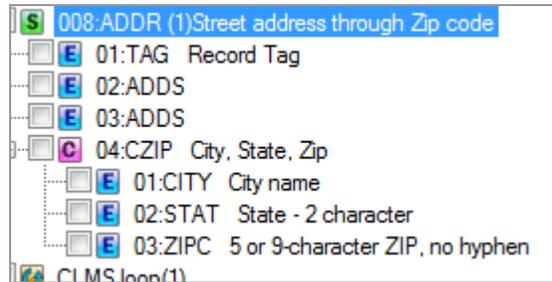


2. To see rules on a particular object, click on the object in the top pane. This displays its Properties tab at the bottom. By scrolling down, you can see its rules.



Mapping from Multiple Segments

Open the ADDR record and its CZIP group item:



The mappings for these are:

Source	Target
Literal inserted by [RULES.Value.ReplaceWith { ADDR}]	01 TAG
2010AA N301 Rules needed.	02 ADDS
2010AA N302 Rules needed.	03 ADDS
2010AA N401	04-01 CITY
2010AA N402	04-02 STAT
2010AA N403	04-03 ZIPC

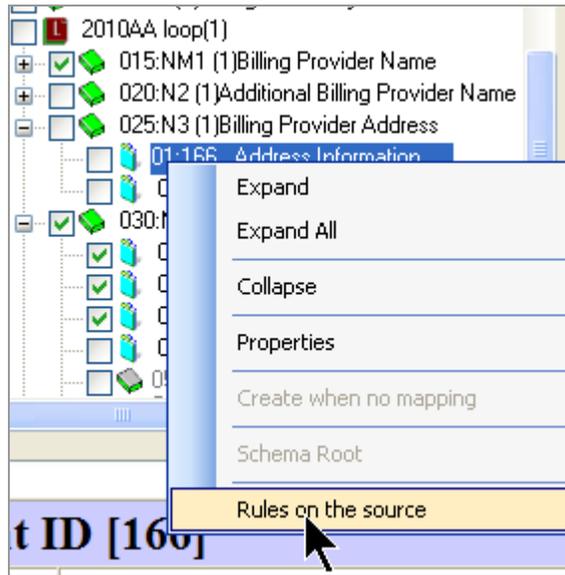
Because we have two different segments feeding into one segment, and because there is a group item within the target record, we will have additional steps to make all of this come out on one ADDR record.

First, do the easy parts:

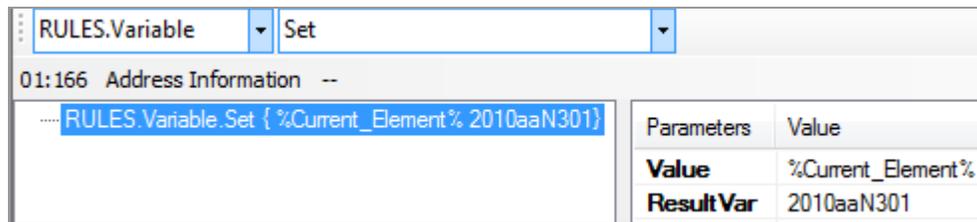
1. Put the rule on the **TAG** field.
2. Drag **N401**, **N402**, and **N403** to their respective fields.
3. Save.

Instead of mapping directly between the N301 and the 02 ADDS field, we put the value from the source into a variable and then use that variable to populate the ADDS field.

1. To create a variable on the **N301**, right-click on it and choose **Rules on the source**:



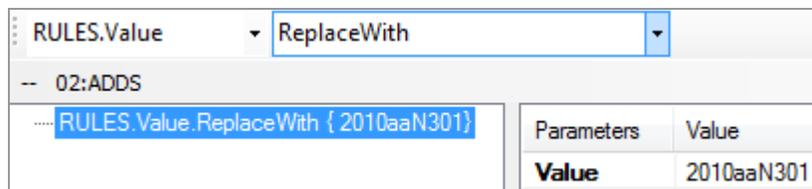
2. Create this rule:



We now have the value in the source's N301 going into a variable called 2010aaN301.

3. Put the contents of the variable into the target's 02 ADDS field:

Drag a line from **02 ADDS** out to the center and create this rule. Be sure the variable name is exactly the same as in the previous rule. This includes capitalization.



4. Create these rules:

Source's N302: [RULES.Variable.Set { %Current_Element% 2010aaN302}]

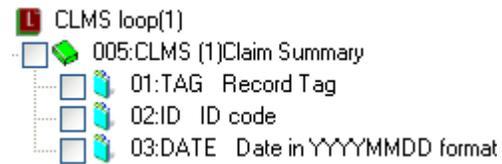
Target's 03 ADDS: [RULES.Value.ReplaceWith { 2010aaN302}]

5. Save and test to see a record similar to this:

ADDR PO BOX 123 157 WEST 57TH SDAMASCUS MD20872

The CLMS Record

Open the CLMS loop and record:



The mappings for this record include:

Source	Target
Literal inserted by [RULES.Value.ReplaceWith {CLMS}]	01 TAG
CLM01 in 2000B 2300 loop	02 ID
DTP03 Initial Treatment Date in 2000B 2300 loop	03 DATE

We again have data from different source segments feeding into the same target record, so we need to capture the one that is higher in the source hierarchy in a variable.

To map the CLMS record:

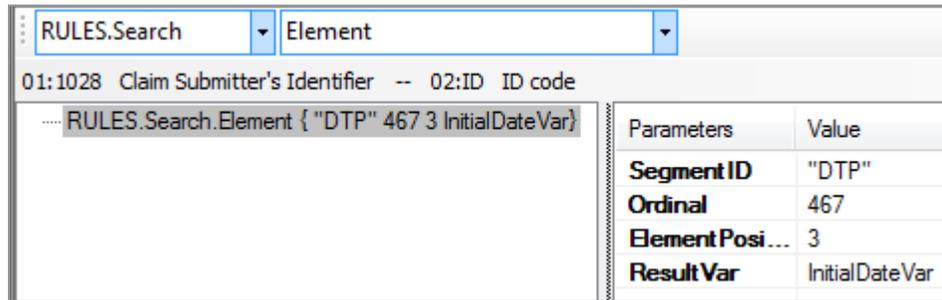
1. Put the usual rule on the **TAG** field.
2. Map from the **CLM01** to the ID field in the **CLMS** record.
3. We now need to get the Initial Treatment Date **DTP03** to the **DATE** field in the CLMS record. Since we have already mapped from a previous segment (CLM) to the CLMS, we need to use a rule rather than mapping directly.

We need to search down the file to grab the initial treatment date in a variable. To do this, we need to know its “ordinal” number. Click Initial Treatment Date DTP and look in its properties for the ordinal:

Segment ID [DTP]	
Description	Date - Initial Treatment
Purpose	To specify any or all of a
Ordinal	467
Repeat	1

Note that it is 467.

Put this rule on the source's **CLM01**:



We now have a variable InitialDateVar containing the date.

- Put this rule on the target's ID field:

```
RULES.Value.ReplaceWith {InitialDateVar}]
```

This puts the contents of InitialDateVar into the target's DATE field.

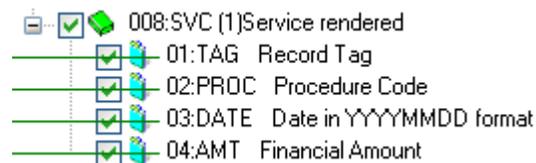
- Save the map and test. You should see a record like this:

```
CLMS 2235057 20040609
```

Since the Initial Treatment Date is optional, the date may show up as a blank in the output.

The SVC Record

Open the target's SVC record:



The mappings for this record include:

Source	Target
Segment ID of the SVC segment	01 TAG
SV101_02 Product/Service ID Rules needed to check the value in an external file and swap in text	02 PROC
DTP03 Service Date (2000B/2300/2400/455 DTP)	03 DATE
SV102 Monetary Amount Rules needed since the mapping is out of order	04 AMT

We again have data from different source segments feeding into the same target record, so we need to capture the one that is higher in the source hierarchy in a variable.

1. Create a rule to put the literal value `svc` in the target SVC's **TAG** field.
2. Map the **DTP03 Service Date** to the **03 DATE** field.
3. Capture the value in the **SV102 Monetary Amount** in a variable and then use it for the value in **03 DATE**:

Source rule on SV102:

RULES.Variable		Set
02:782 Monetary Amount --		
.....RULES.Variable.Set { %Current_Element% SV102AmtVar }		Parameters Value
		Value %Current_Element%
		ResultVar SV102AmtVar

Target rule on 04 AMT:

RULES.Value		ReplaceWith
-- 04:AMT Financial Amount		
.....RULES.Value.ReplaceWith { SV10tAmtVar }		Parameters Value
		Value SV10tAmtVar

4. Test.

Consulting an External File

The SV101_02 Product/Service ID contains a code representing a medical procedure. We want to use the corresponding description in the SVC record.

To do this:

1. Go to Foresight Translator's \DemoData directory and look in **ProcedureList.txt**:

CODE,PROCEDURE

99201,NEW PATIENT OFFICE VISIT

99211,ESTABLISHED PATIENT OFFICE VISIT

99354,LONG OFFICE VISIT

99241,OFFICE CONSULTATION

The first line contains column headings.

Each additional line contains a code that might appear in the source SV101_02 and then its description. If the source contains any code listed here, we want to use the description in the target's 02 PROC field.

2. Back in Translation Tool, create this rule on the **source SV101_02**:

The screenshot shows a rule configuration window. At the top, there are two dropdown menus: 'RULES.Variable' and 'Set'. Below this, the source is identified as '02:234 Product/Service ID --'. The main rule expression is 'RULES.Variable.Set { %Current_Element% ProcCodeVar}'. To the right, a 'Parameters' table is displayed:

Parameters	Value
Value	%Current_Element%
ResultVar	ProcCodeVar

3. Create these on the **target 02 PROC**:

The screenshot shows a rule configuration window. At the top, there are two dropdown menus: 'RULES.TableLookup' and 'SearchInTable'. Below this, the target is identified as '-- 02:PROC Procedure Code'. The main rule expression is 'RULES.TableLookup.SearchInTable { Procedure'. To the right, a 'Parameters' table is displayed:

Parameters	Value
TableName	ProcedureList.txt
InputColumn	CODE=ProcCodeVar
OutputColumn	ProcNameVar=PROCEDURE

This says:

- a. Look in a file called ProcedureList.txt, in Foresight Translator's \Bin directory.
- b. See if the CODE column contains the value in ProcCodeVar (which contains the value from the source's SV101_02).

If true, return the corresponding contents of the PROCEDURE column in a variable called **ProcNameVar**.

If not true, the **ProcNameVar** variable will contain the literal "ProcNameVar."

Conditional Rules

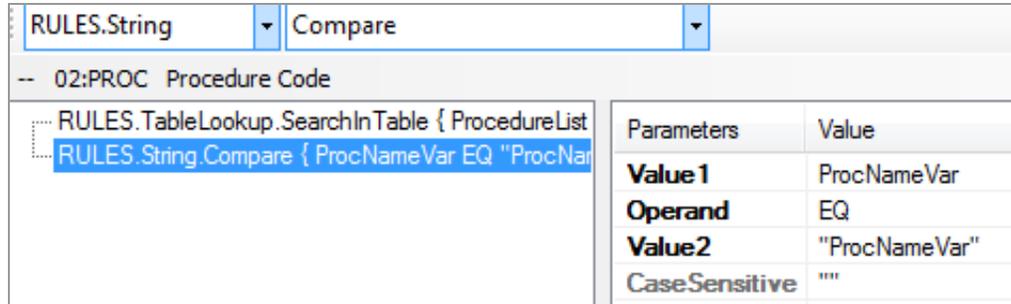
We have now checked an external file for a code, and a variable called **ProcNameVar** contains the corresponding text description. The final step is to use the returned value in 02 PROC.

But, we need to provide for the case that the code was not found in the external file.

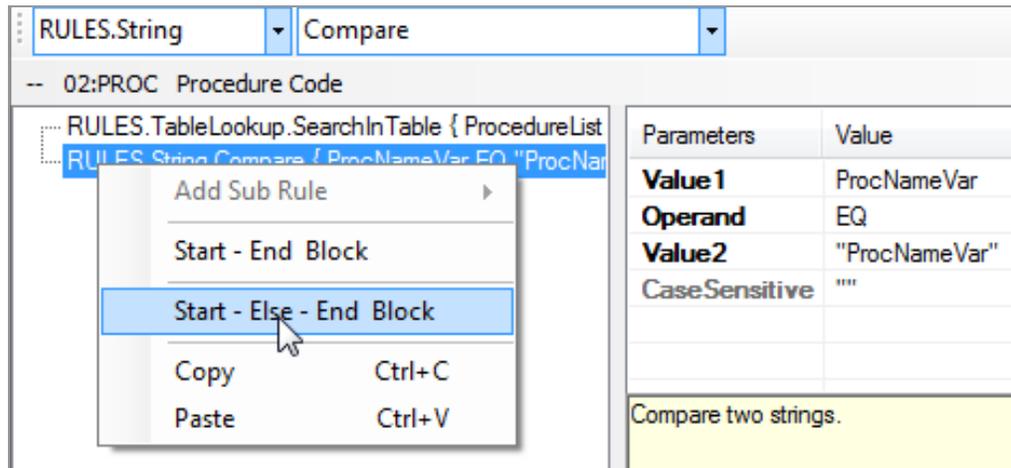
In that case, we want to insert the literal "NONE" in 02 PROC.

To do this:

1. Still in the rules on 02 PROC, add this String Compare rule:



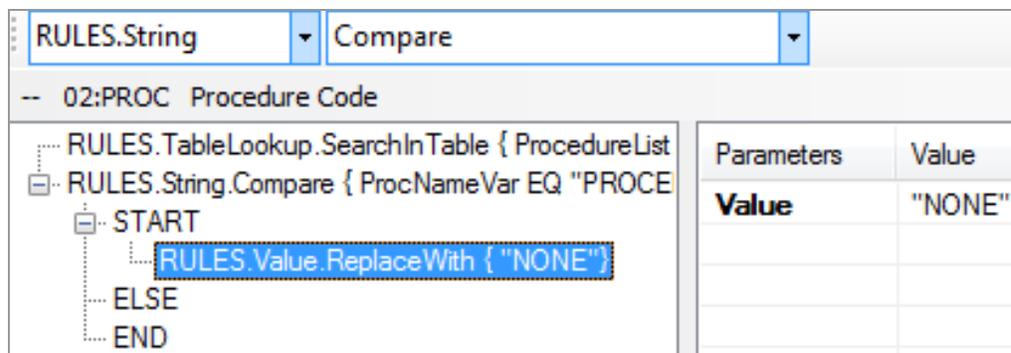
2. Right-click on the String Compare rule and choose **Start-Else-End Block**.



3. The **START** section will execute if the string compare succeeds (if the variable **ProcNameVar** contains the literal "ProcNameVar").

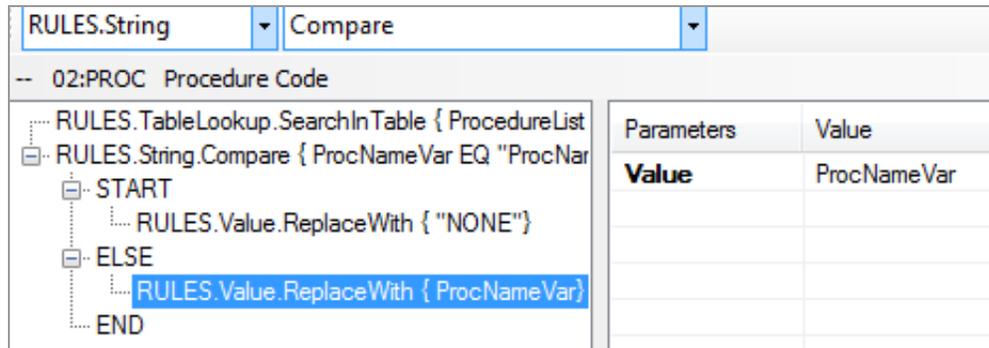
Right-click on **START** and choose Add sub rule | **RULES.Value** | **ReplaceWith**.

Click **RULES.Value.ReplaceWith** and type **"NONE"** for the Value parameter.



- Next, the rule that is to execute if the variable contains something else.

Right-click on **ELSE** and set up this rule:



- Save and test.

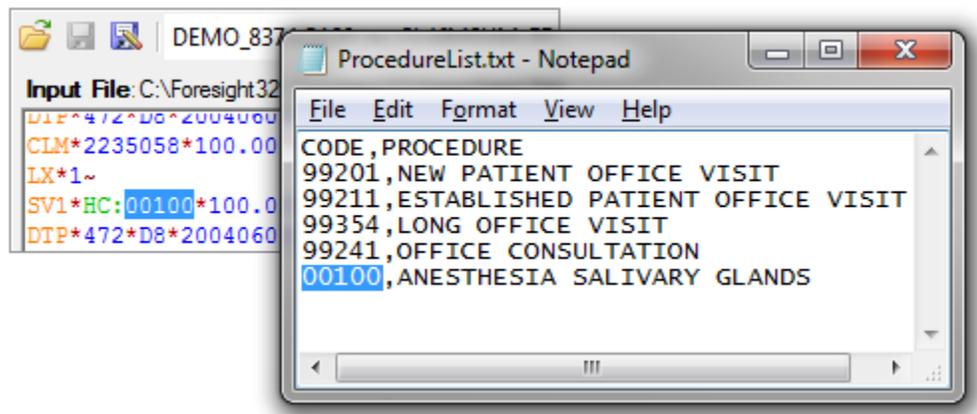
The SVC records should look like this, where the bold values were controlled by these rules. The first SVC01_02 was not found in the external file, so **none** appears.

```

TRAN 0410210944 000000001
NAME MUFFY JONES
ADDR PO BOX 123 157 WEST 57TH SDAMASCUS MD20872
CLMS 2235057 20040609
SVC NONE 20040609
CLMS 2235058 20040609
SVC ANESTHESIA SALIVARY 20040609
CLMS 2235059 20040609
SVC NEW PATIENT OFFICE V20040609
CLMS 2235060 20040609
SVC OFFICE CONSULTATION 20040609

```

Where did ANESTHESIA SALIVARY come from? Notice the code **00100** in the source data, and the same code in the external file that we consulted. This caused the corresponding contents of the PROCEDURE column to be placed in the **ProcNameVar** variable, where it was then used in the target data. The value was truncated due to the size of the target field.

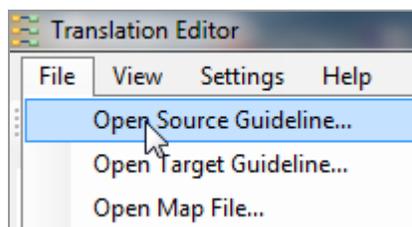


Mapping Tutorial 3 - EDI to EDI

Files used in Tutorial		
Purpose	File	Notes
Source	837AQ320.std (4010A EDI guideline)	In Foresight Translator's \DemoData directory Created by installation
Target	5010837I.std (5010 EDI guideline)	In Foresight Translator's \DemoData directory Created by installation
Map	DEMO_837I_4010A_5010.map	Create during tutorial. Backup in Foresight Translator's \DemoData directory
File to translate	837I_4010_H_1.txt	In Foresight Translator's \DemoData directory Created by installation

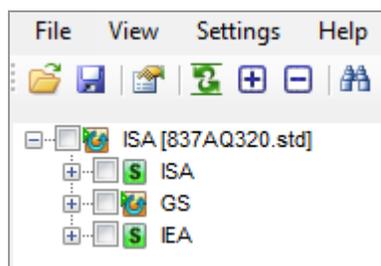
Importing Source and Target Guideline

1. Open **Translation Tool.exe** in Foresight Translator's \Bin directory.
2. Choose **File | Open Source Guideline**.



Choose **837AQ320.std** in Foresight Translator's \Database directory.

It will look like this in Translation Tool:



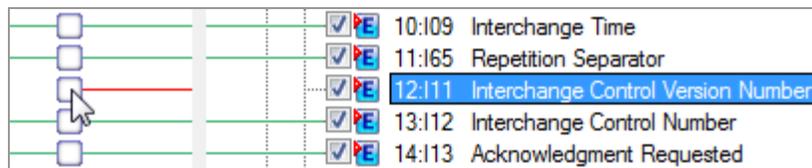
3. Choose **File | Open Target Guideline** and select **5010837I.std** from Foresight Translator's \Database directory.

Applying Rules

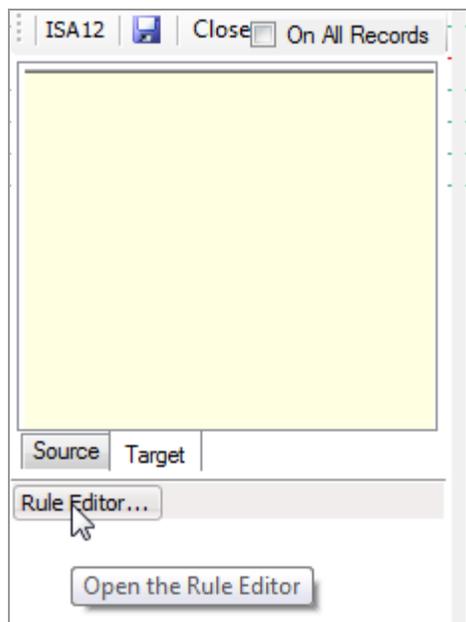
We need to supply a value of 00501 for the ISA12 Interchange Control Version Number. We can do this with a rule.

The entrance to the rule editor is through the little boxes in the center column.

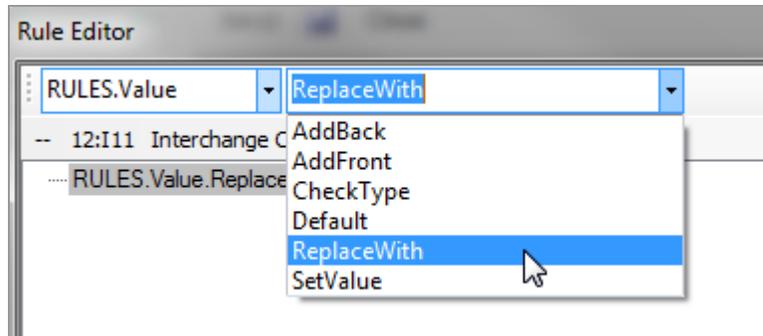
1. Click on the little box in the middle pane for the ISA12.



2. In the rule dialog box, click on **Rule Editor...** at the bottom left.



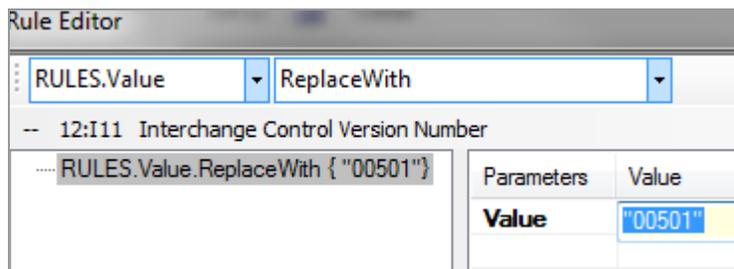
3. Drop down the first field at the top left and choose **RULES.Value**:
4. In the next field, choose **ReplaceWith**:



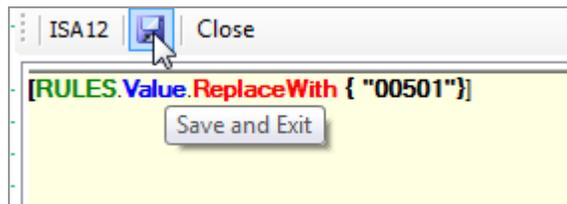
5. In the Parameters area, click at the right of the value field until it becomes editable and then type "00501" – the value to be put in the ISA11.

Click on the rule in the left pane to update it with the parameter.

It should look like this:



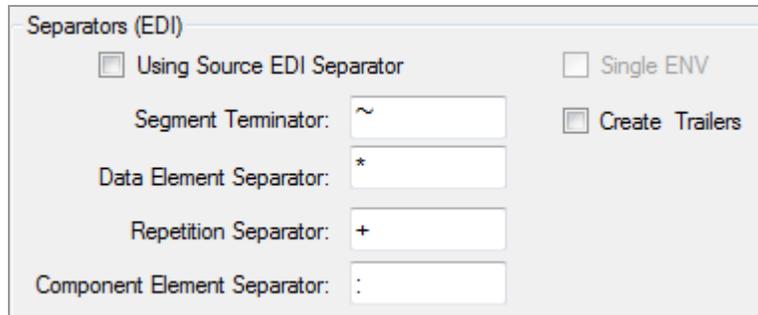
6. Click **OK** and **Save** to save the rule.



Specifying Delimiters

Tell Foresight Translator what delimiters to use when creating the output file.

1. Choose Settings | Target Guideline Settings
2. Specify these delimiters and click **OK**.

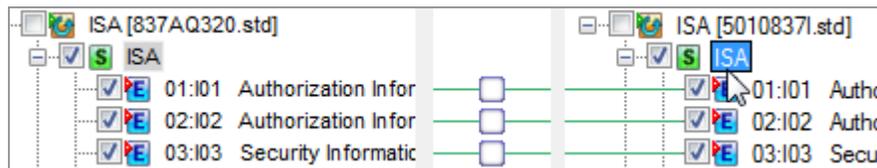


Mapping Data

1. Open the **ISA** segments for the source and the target.

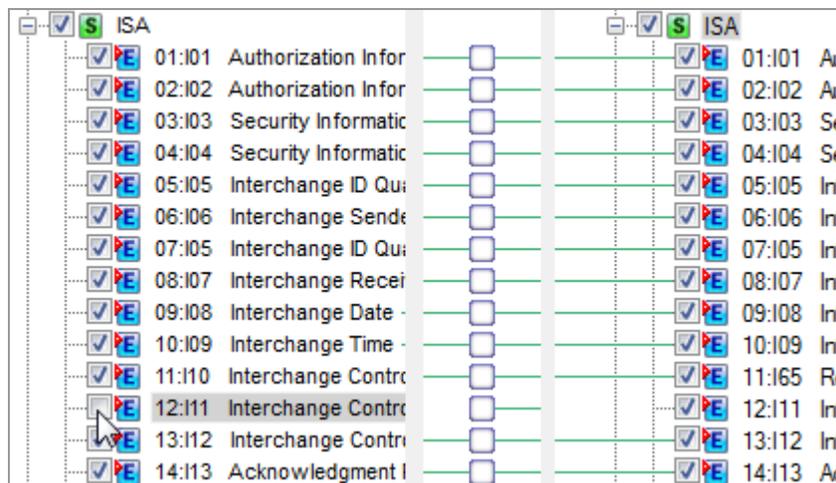
Since they have exactly the same structure, we can just drag between the segments to map all their elements.

2. Drag from the source **ISA** segment to the target **ISA** segment to connect all elements within the ISA at once.



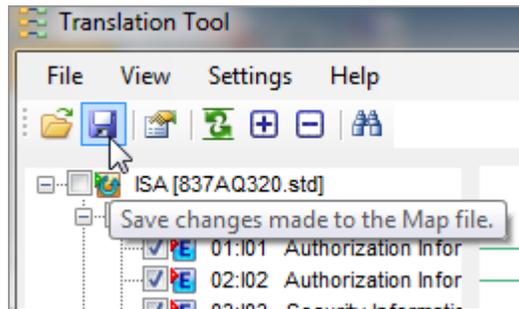
This tells Foresight Translator to put data from each ISA element from each source ISA element into the same element in the target.

3. We don't want the target's **ISA11** (Interchange Control Version Number) to be used in the source, so click its source checkbox to unmap it.



The line will remain from the source to the center of the pane. We will use this soon to create a rule that puts another value in this element in the target.

4. Save the map as **DEMO_837I_4010A_5010.map** in Foresight Translator's \Database directory.



Checking Properties

1. Look at the **ISA11**. The source and target seem to have different meanings.
2. Click on the **ISA11** for source (the left column).
3. If the Properties panel is not displaying at the bottom, click this toolbar button:



4. At the bottom of the Properties panel, be sure the **Properties** tab is selected.

Element ID [I10]	
Description	Interchange Control S
Purpose	Code to identify the
MinMax	01/01
Type	ID
Ordinal	11
Req.Des.	M (-)
Repeat	1
Unmapped Properties Rules Test	
C:\Foresight32\Translator27\Database\837AQ320.std C:	

Notice that the only code is U.

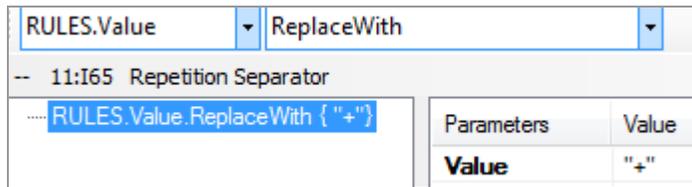
5. Now click on the **target's ISA11** in the top pane and again check the Properties tab.

Notice that it is the repetition separator and that it has no codes.

- Unmap the **ISA11**.
- To create a rule that will use a plus sign as the repetition separator in the target:
Drag a line from the target ISA11 to the middle so that you can make a rule.



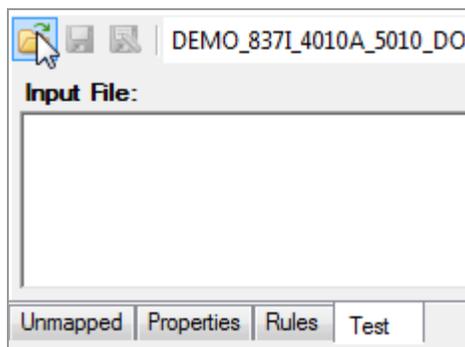
Click on the little box and create this rule:



- Save rule and the map.

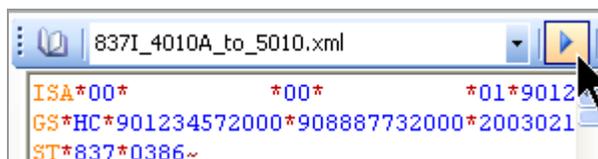
Checking your Map

- In the Properties panel, choose the **Test** tab.
- Click the **Open input file** button:



Select **837I_4010_H_1.txt** in Foresight Translator's \DemoData directory.

- Ensure that the correct map file name is showing to the right of the Open input file button.
- Click the **Run** button:



The fields that we have mapped should appear in the output area at the bottom right:

```
ISA*00*      *00*      *01*9012345720000 *01*9088877320000
*030212*0848*+**000000386*0*T*:~
```

More Mapping

1. In the top pane, open the GS loop and segment for the source and target.



2. Drag a connection between the two GS segments.
3. Check each GS element's properties.
4. Unmap the **GS08** and create a rule to have the target's GS08 always be 005010X223A2.

RULES.Value	ReplaceWith
-- 08:480 Version / Release / Industry Identifier Code	
.... RULES.Value.ReplaceWith { "005010X223A2" }	Paramet... Value
	Value "005010X223A2"

5. The ST segment has a different number of elements in the target, so we have to drag each element separately.

Drag connections from the source **ST01** to the target **ST01**.

Do the same for the **ST02**.

6. For the target **ST03**, drag a connection to the center and then create a rule to put **005010X223A2** in it.
7. Connect the **BHTs** and the **1000A** and **1000B** loops.
8. Save.
9. Again use the Test pane at the bottom to verify translation.
10. Continue mapping corresponding segments.

Where segments have different structures (like N4s, for example), open them and map the elements.

Optional target elements can be left unmapped if they have no corresponding source data.

Notice that the 2010BB source should be left unmapped and the 2010BC source mapped to the 2010BB target.

11. Do not map the SE01. Instead, create this rule to get a new count of the segments:

The screenshot shows the 'Rule Editor' window. At the top, there are two dropdown menus: the first is set to 'RULES.Value' and the second is set to 'ReplaceWith'. Below these, the rule text is '02:329 Transaction Set Control Number -- 02:329 Transaction Set Control Number'. The main text area contains the expression 'RULES.Value.ReplaceWith { %RECORDCOUNTER%}'. To the right of this text is a table with two columns: 'Parameters' and 'Value'. The table contains one row with 'Value' in the 'Parameters' column and '%RECORDCOUNTER%' in the 'Value' column.

Parameters	Value
Value	%RECORDCOUNTER%

12 Appendix B: Java API

Running the Java Demo

Before using the Java API

If you have installed 64-bit Foresight Translator on a Windows platform, be sure that your 64-bit registry has a “Translator” entry under the Foresight group.

Running the Java Demo

1. Go to Foresight Translator’s \API\JavaTransApi directory and compile the **.java** files by running **BuildJava.bat**. You should now have these additional files:

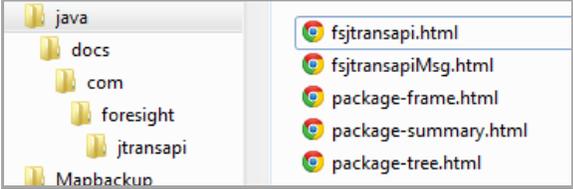
Main.class
APIDemo.class

2. Run the demonstration by executing **RunJavaSample.bat**.
3. Go to Foresight Translator’s \Output directory to see the translated files created.



4. Look in APIDemo.java to see how it was done.

Java Files

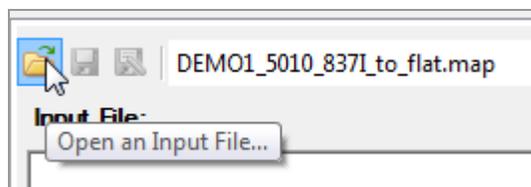
Foresight Translator's \API\JavaTransApi directory	
ApIDemo.java	Source code that you can edit and then compile into a class file.
BuildJava.bat	Batch file that compiles ApIDemo.java into ApIDemo.class and Main.java into Main.class.
RunJavaSample.bat	Batch file that sets up and executes ApIDemo.class, which runs Foresight Translator with various maps.
Foresight Translator's \Java directory	
fsjtransapi.jar	TIBCO Foresight's jar file for Foresight Translator's Java API.
Foresight Translator's \Java\docs\com\foresight\jtransapi directory	
fsjtransapi.html	Documentation for class fsjtransapi.  View fsjtransapi.html for field, constructor, and method summaries and details for use when altering ApIDemo.java.

Note: If you plan to change the sample code, copy the project to a different directory. Sample files get reinstalled with each release.

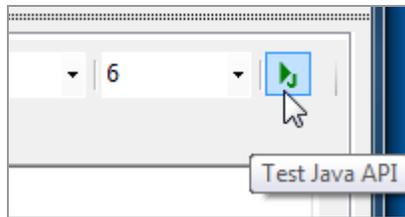
Building and Running Java from Translation Tool

To build and run Java from Translation Tool:

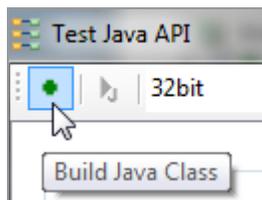
1. Open a map.
2. From the Properties pane, open the **Test** tab and choose a data file to translate.



3. On the far right above the Output File area, click the arrow:

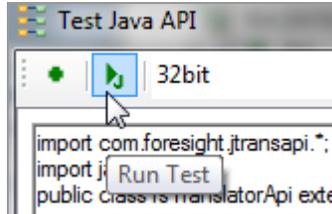


4. In the Test Java API dialog, type the path to your jdk\bin directory. Make any other changes you'd like. See [Test Java API Dialog](#) for more information about the available options.
5. Click the green button on the toolbar to build the class:

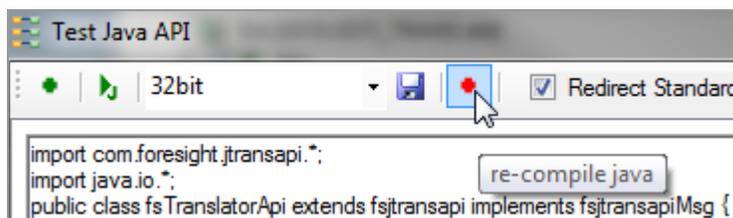


The **Source** tab should now show the contents of the class. You can type in this pane to make changes.

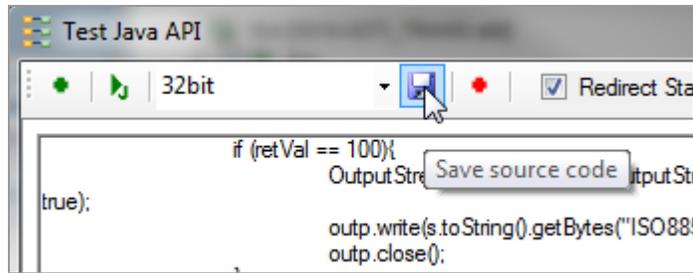
6. When finished with any changes, click the green **Run Test** button:



7. Check the **Log** tab for a return code of 100, meaning success.
8. The **Result** tab will show the translated data.
9. You can make further changes on the **Source** tab and then click the red **Re-compile Java** button:



10. To save the Java class, use the **Save** button:



Test Java API Dialog

Toolbar



	Meaning	Explanation
①	Build Java Class	Used to build the class. After the Java class has been built, the contents of class appear in the Source tab.
②	Run Test	Test the Java code.
③	Java Model	Drop down list; select 32- or 64-bit version of Java.
④	Save Java Source to File Save Result to File (Results Tab only)	Displays a save dialog allowing you to save the Java Source or Results file.
⑤	Re-compile Java	Available on the Test tab only. Used to recompile Java code after corrections have been made.
⑥	Redirect Standard Output	Unchecked = no output displayed. Checked = output displayed.
⑦	Text Box	Future Development.

Configure Tab

The **Configure** tab allows you to configure options for the Java API.

General

Java: C:\Program Files\Java\jdk1.7.0_09\bin

Map: 5010-8371_Deidentify.xml Memory

Source: 837-X223.std Memory

Target: 837-X223.std Memory

IN: C:\tibco\translator\3.3\DemoData\8371_5010_H_cle Memory

Out: Memory

Convert: EDIToEDI

Class name: fsTranslatorApi

Using callback select guideline(memory)

Variables

	Select	Name	Value
*	<input type="checkbox"/>		
...			

Detail

Call Buffer Size: 4096

Encoding: Default

Source delimiters

Record: 0x00 Sub-Field: 0x00

Field: 0x00 Over Write Delimiters

Target delimiters

Segment: Repetition:

Element: Word Wrap

Sub-Element:

EDIFACT delimiters

Create UNA

Component: 0x3a : Release: 0x3f ?

Element: 0x2b + Reserved: 0x20

Decimal: 0x2e . Segment:

Configure Source Log Result

Source Tab

The **Source** tab shows the contents of the Java class after it has been built. You can type in this pane to make changes.

```
import com.foresight.jtransapi.*;
import java.io.*;
public class fsTranslatorApi extends fsjtransapi implements fsjtransapiMsg {

    static String ROOTDIR="C:/tibco/translator/3.3";
    static String OUTPUTDIR = ROOTDIR + "/Output/";
    static String INPUTDIR = ROOTDIR + "/DemoData/";
    static String DatabaseDir=ROOTDIR + "/Database/";
    private StringBuilder s = new StringBuilder();private StringBuilder log = new StringBuilder();
    private String outputFileName="";
    private OutputStream outlog = null;

    public static void main(String[] args) {
        if (args.length > 0)
        {
            ROOTDIR = args[0];
            OUTPUTDIR = ROOTDIR + "/Output/";
            INPUTDIR = ROOTDIR + "/DemoData/";
            DatabaseDir = ROOTDIR + "/Database/";
        }
        fsTranslatorApi obj = new fsTranslatorApi();
        obj.test();
    }

    public void test(){
        try{
            this.init();
            s.delete(0,s.length());
            outlog = new FileOutputStream("C:/tibco/translator/3.3/Bin/fslog/APILOG.txt", true);

            inputMemoryBuffer = getBytesFromFile("C:/tibco/translator/3.3/DemoData/8371_5010_H_clean.txt");
            configuration_flag |= INPUTFILE_BYMEMORY;

            outputFile = "C:/tibco/translator/3.3/Bin/fslog/7ec1cb43-0d1b-418a-93de-cd3981b0bdd5.txt";
            configuration_flag |= OUTPUTFILE_BYMEMORY;
        }
    }
}
```

Configure Source Log Result

Log Tab

The **Log** tab provides a log of the Java activity.

```
C:\Program Files\Java\jdk1.7.0_09\bin\java -classpath .;C:/tibco/translator/3.3/bin;C:/tibco/translator/3.3/Java/fsjtransapi.jar fsTranslatorApi
java.lang.UnsatisfiedLinkError: C:/tibco/translator/3.3/bin\Translator.dll: Can't load IA 32-bit .dll on a AMD 64-bit platform
```

Result Tab

The **Results** tab provides results of the Java activity.

13 Appendix C: Packed and Unpacked Data

Default Handling of Packed Decimal Source Data

You may need to account for the differences in data type between some source and target fields – for example, N2 source data that must be R type in the output.

Packed decimal data is converted properly by default, including any necessary decimal point and minus sign.

What you need to do:

- If the source data is packed decimal, Foresight Translator will automatically convert it to the R type for output.
- If the source data is not packed decimal, you have to insert the decimal with a rule.
- If the source may or may not be packed decimal, you need to use a rule to check it and insert the decimal if it is not packed data.

N Source Data, R Target Data

N means an implied decimal. For example, N2 means 2 digits to the left of the last digit.

R means a number containing an explicit decimal point and signed.

N2 value		R value
3000	=	30.00
300	=	3.00
30	=	.30
3	=	.03

Example 1: Source not packed

Assume source is N2 type and target is R type. This input data has a number for the last digit, so it is not packed decimal:

```
Input File: C:\TIBCO\Translator\DemoData\ou
820000N EPS EDI37960 0033718617
820BPR0000C000000000000103CX12 040
820NTE0000ALLFREE FORM MESSAGE
```

With no rules, output value has not been transformed to include the decimal:

```
Output File:
ISA*00*1234567891*00*123456789
GS*RA*901234572000*9088877320
ST*820*TRANSACTIONCTRL!
BPR*C*103*C*X12**04*000466666
```

Because the last digit of the source value was numeric, the decimal is not packed.

Example 2: Source packed

Again assume source is N2 type and target is R type. This input data has a J for the last digit, so it is packed decimal:

```
Input File: C:\TIBCO\Translator\DemoDa
820000N EPS EDI37960 003371
820BPR0000C00000000000010JCX12
820NTE0000ALLFREE FORM MESSAGE
```

With no rules, the automatic unpacking process also adds the minus sign and the decimal:

```
Output File:
ISA*00*1234567891*00*123456789
GS*RA*901234572000*9088877320
ST*820*TRANSACTIONCTRL!
BPR*C*-1.01*C*X12**04*000466666
```

Example 3: Source might or might not be packed

What if the input value might or might not be packed decimal? This means the last digit of the source value combines a numeric value and a sign. For example, if the last digit is 1 and the sign is +, then the last digit will be **A**.

This example uses the IsUnPacked rule to determine if it has been automatically unpacked by Foresight Translator. If so, it uses the automatically unpacked value in the output. Otherwise, it formats the output with a SetDecimal rule.

```

RULES.Value.IsUnPacked { %Current_Element%
├── START
│   └── RULES.Value.ReplaceWith { %Current_Element% }
├── ELSE
│   ├── RULES.Format.SetDecimal { %Current_Element% "2" BPR02}
│   └── RULES.Value.ReplaceWith { BPR02}
└── END

```

Packed Decimal Chart

Zoned decimal conversion chart		
Last digit of value	Meaning of last digit	
	This number	This sign
{	0	+
A	1	+
B	2	+
C	3	+
D	4	+
E	5	+
F	6	+
G	7	+
H	8	+
I	9	+
}	0	-
J	1	-
K	2	-
L	3	-
M	4	-
N	5	-
O	6	-
P	7	-
Q	8	-
R	9	-

14 Appendix D: Return Codes

Seeing Return Codes

To display return codes when you run a script, put this line similar to this in the script right after running the program:

UNIX `echo "return code = " $?`

Windows `@echo [Return Code = %ERRORLEVEL%]`

This returns something like: `[Return Code=100]`

Virus Checking and TIBCO Foresight Products

Exclude all TIBCO Foresight workflow subdirectories from virus checking.

Foresight Translator Return Codes

Return Code	Meaning
100	Translation ran successfully.
105	Could not open the source schema.
106	Guidelines specified in the API are not the same guidelines used in the map.
110	Command line error or error loading INI file.
111	One of the documents being translated must be HL7 when using the HL7 plug-in.
112	A JNI EXECPTION was encountered when using the Java API.
117	The schema cannot be parsed or the XML data does not match it.
124	Configuration invalid. If this occurred when using the API, this is often because it cannot read the INI file. Check TRANSRoot.
131	Map doesn't match the convert call. (Example: Attempting to convert EDI to XML by using an XML-to-EDI map.)
135	Database call failed.

Return Code	Meaning
136	Database call failed when using ISIServer.
140	API return code for system failure.
154	Converting between this type of source and target is not supported.
155	Could not open the source guideline.
156	Could not open the target schema.
157	Could not open the target guideline.
158	Could not open the map file.
171	Configuration invalid. If this occurred when using the API, this is often because it cannot read the INI file. Check TRANSRoot.
172	Missing source guideline.
173	Missing target guideline.
174	Missing map.
175	Missing input data when using inputMemory while using API.
176	Cannot access input file when using inputByFile while using API.
177	The source file separators are invalid as specified by separator in the API.
178	The target file separators are invalid as specified by separator in the API.
180	Failed on OutputFile or Options while using API. Cannot write the output file, or options like CALLBACKBUFFER and UNA are the wrong format.
181	There is an error on the guideline itself. (Example: This might occur when using a guideline that was not created with TIBCO Foresight tools.)
199	The API is getting a JNI exception occurred and can't run Java.
201	There is an error in the input path. (Examples: An invalid or unreadable input file has been specified. A parameter, such as -t, has been left empty.)

Troubleshooting information	Notes
log file	Specified with -r command-line parameter. Log goes in the same file as the output.
log in Translation Tool	<p>Use the Log button on the Test pane's toolbar:</p>  <p>Time out: 10000 milliseconds   </p> <p>Or see the log in Foresight Translator's bin\fslog directory.</p>

See [Debugging and Logging](#) for more information.

15 Appendix E: XML Schemas

Introduction

An XML Schema defines the format and content of an XML document. XML schemas can be used as a source or a target in Foresight Translator when translating EDI data. Schemas are identified by their .XSD extension.

For the easiest and most direct use of Foresight Translator, we recommend creating a schema and associated maps by exporting a guideline using EDISIM Standards Editor as detailed in this section.

If you've created a schema using another source, mapping will be more complex and may require advanced business rules. Contact TIBCO Foresight Support for assistance.

For TIBCO BusinessConnect™ and TIBCO ActiveMatrix BusinessWorks™ Users

BusinessConnect™ and ActiveMatrix BusinessWorks™ users have EDISIM Standards Editor options that automatically output a schema in a format compatible with their system.

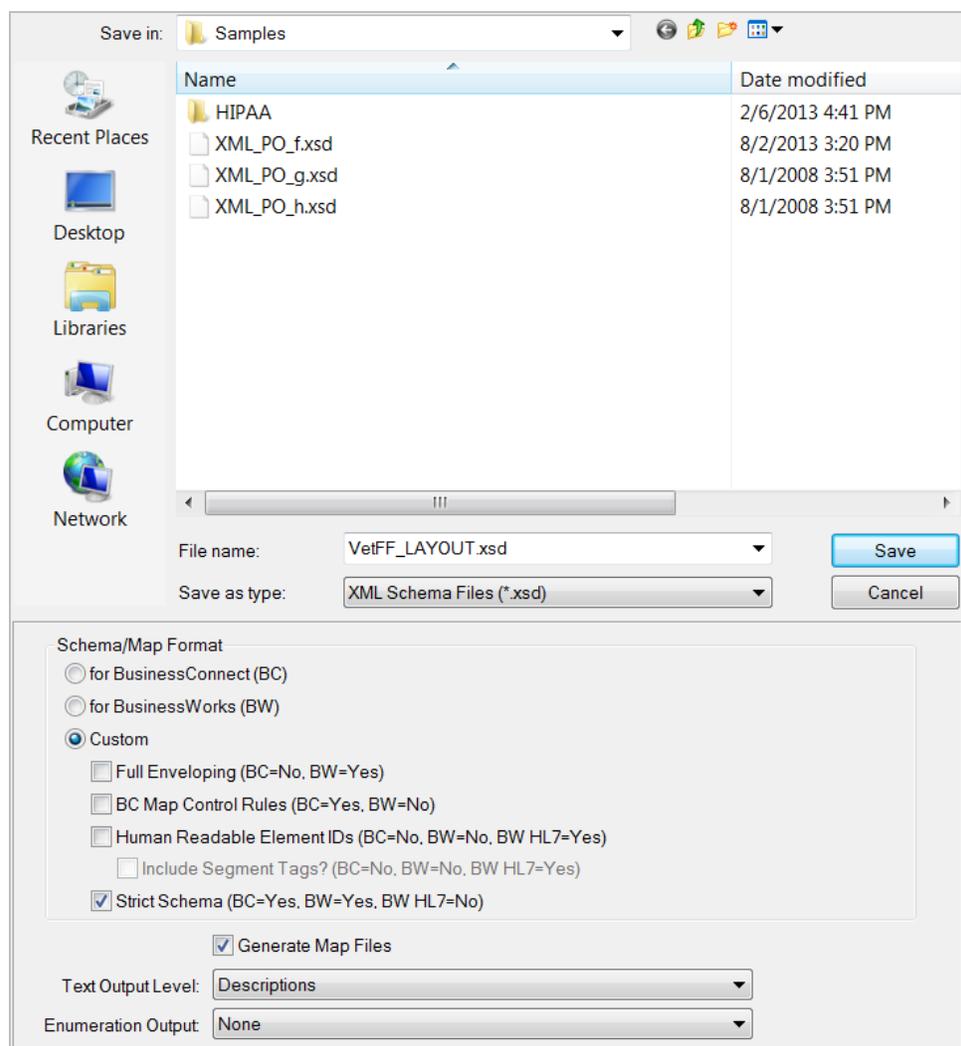
- For BusinessConnect, the format consolidates enveloping segments at the top of the schema and outputs rules to assign control numbers.
- For BusinessWorks, the format uses full formatting and preserves the required tree structure of the data.

Generating an XML Schema with EDISIM Standards Editor

Any EDI or Flat File guideline can be exported as an XML Schema file.

1. Open Standards Editor.
2. Open a transaction set (**File | Open**).
3. From a transaction set or message within a guideline:

Choose **File | Export | Export Current Guideline | To Schema...**



4. Choose a folder and name for the exported file.
5. Refer to [Schema/Map Formatting Options](#) on page 337 and select Schema/Map Format Options as options as needed. When done, continue with Step 6 of this procedure.
6. Click **Save** to export.

7. Copy the XSDs and MAP files to Foresight Translator's \Database directory.

You can now use the schema in Foresight Translator.

Schema/Map Formatting Options

Schema/Map Format

for BusinessConnect (BC)

for BusinessWorks (BW)

Custom

Full Enveloping (BC=No, BW=Yes)

BC Map Control Rules (BC=Yes, BW=No)

Human Readable Element IDs (BC=No, BW=No, BW HL7=Yes)

Include Segment Tags? (BC=No, BW=No, BW HL7=Yes)

Strict Schema (BC=Yes, BW=Yes, BW HL7=No)

Generate Map Files

Text Output Level:

Enumeration Output:

Schema/Map Format

Select from one of the following formats:

For BusinessConnect (BC):

Schema/Map Format

for BusinessConnect (BC)

When selected, Standards Editor outputs the schema in a format compatible with TIBCO BusinessConnect™. This format consolidates enveloping segments at the top of the schema and outputs rules to assign control numbers used by BusinessConnect.

For BusinessWorks (BW):

Schema/Map Format

for BusinessConnect (BC)

for BusinessWorks (BW)

When selected, Standards Editor outputs the schema in a format compatible with TIBCO BusinessWorks™. This format uses full formatting and preserves the tree structure of the data as required by BusinessWorks™.

Custom:

Schema/Map Format

for BusinessConnect (BC)

for BusinessWorks (BW)

Custom

Full Enveloping (BC=No, BW=Yes)

BC Map Control Rules (BC=Yes, BW=No)

Human Readable Element IDs (BC=No, BW=No, BW HL7=Yes)

Include Segment Tags? (BC=No, BW=No, BW HL7=Yes)

Strict Schema (BC=Yes, BW=Yes, BW HL7=No)

The Custom option allows you to specify formatting options for the schema:

- Full Enveloping. If selected, this option puts the envelope grouping around the transaction. If not selected, the enveloping is consolidated at the top of the schema.

This option is

- not compatible with BusinessConnect
- compatible with BusinessWorks.

- BC Map Control Rules. If selected, this option inserts the BusinessConnect control rules into the map files.

This option is

- compatible with BusinessConnect
- not compatible with BusinessWorks.

- Human Readable Element IDs

This option causes the exported schema and maps to use the segment, composite, and element names in the schema element naming.

Example:

Without Human Readable Element IDs selected:

```
S-BHT_2  
E-1005_2_01
```

With Human Readable Element IDs selected:

```
BHT.BeginningOfHierTransaction_2  
HierStructureCode_2_01
```

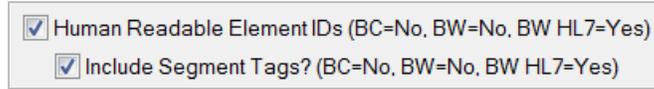
Note: Use of this option results in very large schemas.

This option is

- not compatible with BusinessConnect
- not compatible with BusinessWorks
- compatible with BusinessWorks HL7 data.

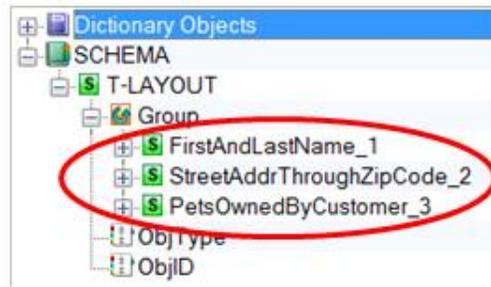
- Include Segment Tags?

When the Human Readable Element IDs option is selected, you can also opt to include segment tags.

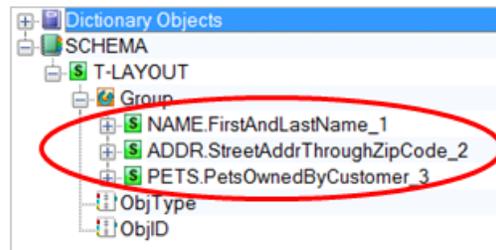


Checking the Include Segment Tags box will include the segment name as part of the schema element name. This is useful for additional naming information and makes the data more readable.

Without box checked:



With box checked:



- Strict Schema (EDI to Schema export only)

This option is used to specify if data types and limits in the base guideline should be carried over to the schema.

If Strict Schema is selected, the data types in the schema appear the same as the data types in the source guideline. This includes the length, repeat, and type limits from the source guideline.

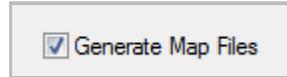
If Strict Schema is not selected, the data types in the source guideline become text in the schema. The schema will retain the the structure of the source guideline but the length, repeat, and type limits from the source guideline are removed (i.e., they are converted to text in the schema).

This option is

- compatible with BusinessConnect
- compatible with BusinessWorks
- not compatible with BusinessWorks HL7 data.

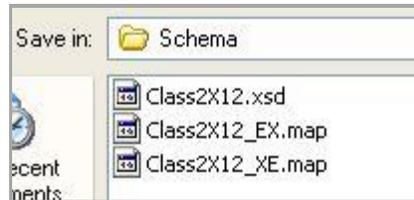
Generate Map Files

If desired, select Generate Map Files.



Map files map EDI to XML between the guideline and schema and vice versa and are used exclusively by Foresight Translator. This option is checked by default.

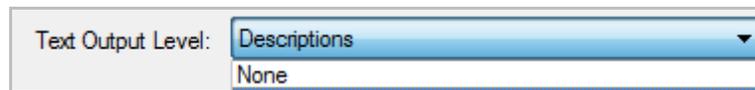
Two map files are generated and saved in the same location as the XML schema file.



<filename>_EX.map contains the EDI to XML mapping

<filename>_XE.map contains the XML to EDI mapping

Text Output Level



Specify the Text Output Level you would like included in the export:

- None
- Descriptions (default)

Enumeration Output



Specify the Enumeration Output Level you would like included in the export:

- None (default)
- Local Code Sets and Application Values

16 Appendix F: Foresight Translator Demos

Overview

This appendix lists demos that ship with Foresight Translator.

File locations:

- Maps and guidelines are in Foresight Translator's \Database directory.
- Input files are in Foresight Translator's \DemoData directory.
- Output goes to Foresight Translator's \Output directory.

Demos to run from a Script

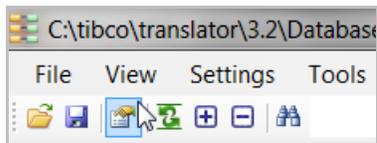
Execute these from Foresight Translator's \Scripts directory.

Demos to run from a batch file		
Translation	Script	Maps
EDI to XML XML to EDI (837I)	T_837I_4010_to_XML_and_back	DEMO3_EDI_to_XML_837I.map DEMO4_XML_to_EDI_837I.map
EDI to flat flat to EDI (837I)	T_837I_5010_to_flat_and_back	DEMO1_5010_837I_to_flat.map DEMO7_flat_to_5010_837I.map
HL7 to XML XML to HL7	T_ADTA05_HL7_to_XML T_XML_to_ADTA05_HL7	DEMO17_HL726_xml DEMO18_XML_HL726.map
EDI to flat (837I) <i>Note the encoding in source GS03</i>	T_837I_5010_targetUTF-8	DEMO1_5010_837I_to_flat.map
Translation	Script	Maps
EDI to flat (837I) Using partner automation	T_837I_5010_PartnerAutomation	DEMO1_5010_837I_to_flat.map <i>(selected by the partner automation file Translator_TPA.csv in Translator's DemoData directory)</i>
Any - you specify	TranslatorTestMap.bat	Any - you specify

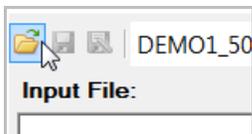
Demos to use with Translation Tool.exe

Translation Tool demos are listed on the next page. To use them:

1. Open Translation Tool.exe.
2. Open a map listed on the next page.
3. Display the Properties pane:



4. Choose the **Test** tab at the bottom.
5. Open the data file:



6. Click **Run**:



Translation	Map	Source / Target / Data file for testing
EDI to flat (837I- WPS)	DEMO1_5010_837I_to_flat.map	5010-837I_TRANS.STD 5010_flat_837I.STD 837I_5010_H_clean.txt
EDI to XML (850)	DEMO2_850_to_XML.map	DEMO_4010_850.STD DEMO_Order.xsd TranslatorDemo_850_4010_X_1.txt
EDI to XML (837I)	DEMO3_EDI_to_XML_837I.map	837AQ320.std 837AQ320.xsd 837Iclean.txt
XML to EDI (837I)	DEMO4_XML_to_EDI_837I.map	837AQ320.xsd 837AQ320.std 837Iclean_xml.xml
XML to EDI (EDIFACT Orders)	DEMO5_EDIFACT_XML_to_EDI.map	Edifact_Orders.xsd EDIFACT_orders.std Edifact_Orders_new.xml
Flat to EDI (270)	DEMO6_flatB_to_EDI_270.map	270_Layout.std 270AA120.std TranslatorDemo_flat_B_format.txt
flat to EDI (837I)	DEMO7_flat_to_5010_837I.map	5010_flat_837I.std 5010-837I_TRANS.STD 5010-MEDICAREA_testdata_flat.txt
EDI to EDI (837P)	DEMO8_837P_DBlookup.map Setup required. See _readme_ODBC.txt in Translator's DemoData\ODBC directory.	837AQ120.std 837-X222.std TranslatorDemo8_4010_837P.txt
XML to EDI (835)	DEMO9_XMLtoEDI_CreateTrailers.map Shows how to create GE and IEA segments when the source does not have them.	Trizet835.xsd 835-X221.std Trizet835_5010_XML_Sample.xml
EDI to flat (850)	DEMO10_850_FlatDelimited.map	DEMO_4010_850.std PODELIM.std 850_4010_X_1.txt
EDI to EDI (837I)	Demo_ICD_9to10_837I_Convert.map Installed with ICD Adapter. Shows ICD-9 to ICD-10 translations available with the TIBCO Foresight ICD-10 Conversion Adapter Version (a Translator add-in)	Demo12_41837I_ICD9_2000B_S.txt

Translation	Map	Source / Target / Data file for testing
EDI to one-record delimited flat	DEMO14_edi_flat.map Shows how to use the Lock and WriteTargetSegment rules to write out one record per transaction.	837-X223.std 837-223_flat_Layout.std TranslatorDemo8_4010_837P.txt
NCPDP to XML	DEMO15_NCPDP_to_xml.map	NCPDP5New.std NCPDP5New2-B1_BW.xsd NCPDP_Trans.txt
XML to NCPDP	DEMO16_NCPDP_xml_to_edi.map	NCPDP5New2-B1_BW.xsd NCPDP5New.std Ncpdp_B1.xml
HL7 to XML	DEMO17_HL726_xml.map	HL7_ADT_A05.std HL7_ADT_A05_BW.xsd ADT_A05_HL7_1.txt
XML to HL7	DEMO18_XML_HL726.map	HL7_ADT_A05_BW.xsd HL7_ADT_A05.std ADT_A05_HL7_1.xml
XML to HL7	DEMO19_XML_HL7_Base64.map In the source, E-98_6_03 Base64 data. It is converted to text and put in the NTE-03 in the target. Note the Base64decoding rule on the target.	BRP_O03_new.xsd BRP_O03_new.std BPR.xml
HL7 to XML	DEMO20_HL7_XML_Base64.map In the source, the NTE-03 contains unprintable characters. It is converted to Base64 data and put in E-98_6_03 in the target. Note the Base64encoding rule on the source.	BRP_O03_new.std BRP_O03_new.xsd HL7_BRP_O30d.txt
Stored procedures	Setup required. See _readme_TranslatorStoredProc.txt in Translator's DemoData\ODBC directory.	