

# TIBCO Foresight® EDISIM®

## Business Rules

*Version 6.20.0*

*May 2021*



# Contents

|   |               |
|---|---------------|
| <b>Introduction</b>                                   | <b>1</b>      |
| Document Purpose .....                                | 1             |
| Intended Audience .....                               | 1             |
| What you need before using Business Rules.....        | 2             |
| Big Picture .....                                     | 2             |
| Viewing TIBCO Foresight-supplied HIPAA Rules .....    | 3             |
| Using External Routines .....                         | 4             |
| *Call External Routine .....                          | 4             |
| Type the Rule.....                                    | 5             |
| Use the Parameters Grid.....                          | 6             |
| Registering new Business Rules .....                  | 8             |
| <br><b>Tutorials: Set up your own Rule</b>            | <br><b>9</b>  |
| HIPAA business rule tutorial .....                    | 9             |
| Overview.....   | 9             |
| Create your own error message.....                    | 10            |
| Set up a rule.....                                    | 10            |
| Copy the guideline from EDISIM to Instream .....      | 10            |
| Test the rule.....                                    | 11            |
| Merge your business rule with a HIPAA guideline ..... | 11            |
| X12 business rule tutorial .....                      | 11            |
| Overview.....   | 11            |
| Set up a rule.....                                    | 12            |
| Test the rule.....                                    | 12            |
| <br><b>Analyzer vs. other Validation Programs</b>     | <br><b>15</b> |
| Overview .....  | 15            |
| Local Variables .....                                 | 16            |
| Condition and Rule Definition Dialog Box.....         | 17            |
| External Routines .....                               | 18            |
| <br><b>Business Rules Reference</b>                   | <br><b>19</b> |
| Overview .....  | 19            |
| Reserved Variables .....                              | 20            |
| Current_Date.....                                     | 21            |
| Current_Delim.....                                    | 23            |
| Current_Element.....                                  | 24            |
| Current_ErrCount.....                                 | 25            |
| Current_LoopCount.....                                | 27            |
| Current_LoopKey .....                                 | 27            |
| Current_Row and Next_Row.....                         | 27            |
| Current_Time .....                                    | 28            |
| GLOBAL_FILENAME.....                                  | 29            |

|  |    |
|--|----|
| GLOBAL_FILEPATHNAME .....                            | 30 |
| Using Reserved Variables in a Message .....          | 31 |
| Literals.....  | 32 |
| Escape Character for Double Quotes .....             | 32 |
| Copying Business Rules .....                         | 33 |
| Printing Business Rules .....                        | 34 |
| Array Business Rules .....                           | 35 |
| Array Reserved Variables .....                       | 35 |
| CheckVarFromArray.....                               | 40 |
| ClearArray .....                                     | 41 |
| CreateArray .....                                    | 42 |
| DumpArray .....                                      | 43 |
| GetArrayCurrentRowIndex.....                         | 43 |
| GetArrayNextColumnIndex.....                         | 44 |
| GetArrayNextRowIndex .....                           | 45 |
| GetARowFromArray .....                               | 46 |
| GetVarFromArray .....                                | 48 |
| SearchVarsInArray.....                               | 49 |
| SearchConditionsInArray .....                        | 51 |
| SetArrayFromVar .....                                | 55 |
| UpdateArrayFromDate .....                            | 57 |
| Correct Coding Initiatives (CCI) Business Rules..... | 58 |
| CCIInit .....  | 59 |
| CCICollect.....                                      | 59 |
| CCIANalyze .....                                     | 60 |
| Code Lookup Business Rules.....                      | 61 |
| FindCode .....                                       | 61 |
| FindCodeWithDate.....                                | 63 |
| FindUserCode.....                                    | 65 |
| FindUserCodeWithDate .....                           | 65 |
| ValidateZipState .....                               | 66 |
| Core3 (Phase III CORE) Business Rules .....          | 67 |
| Custom Record Business Rules .....                   | 68 |
| DefineCustomRec .....                                | 70 |
| OutputCustomRec.....                                 | 72 |
| RemoveCustomRecord .....                             | 73 |
| Date and Time Business Rules .....                   | 74 |
| CheckDateInRange.....                                | 74 |
| CompareDate .....                                    | 76 |
| DateCalc .....                                       | 78 |
| GetGMTDate'Time.....                                 | 85 |
| ValidateDate'Time.....                               | 86 |
| ValidateDate'TimeUN and ValidateDate'TimeX12.....    | 88 |
| DBServer Business Rules .....                        | 89 |
| DBExecute.....                                       | 89 |
| DBQuery .....  | 92 |
| InvokeWebService.....                                | 94 |
| Exit Business Rules .....                            | 98 |

|  |     |
|--|-----|
| ClearExits .....                             | 98  |
| KeepOrder .....                              | 99  |
| SetCompositePreExit .....                    | 100 |
| SetElementPostExit .....                     | 101 |
| SetLoopPostExit .....                        | 102 |
| SetLoopPostInstanceExit .....                | 104 |
| SetSegmentPreExit .....                      | 105 |
| UserExitWithoutWait .....                    | 107 |
| UserExitWithWait .....                       | 108 |
| ICD Business Rules .....                     | 111 |
| List Business Rules .....                    | 111 |
| ClearList .....                              | 111 |
| InList .....                                 | 112 |
| ListCheck .....                              | 113 |
| ListContig .....                             | 115 |
| ListCount .....                              | 117 |
| ListGetVar .....                             | 118 |
| ListInsert .....                             | 119 |
| ListMinMax .....                             | 119 |
| Lookahead Business Rules .....               | 123 |
| Marking a Lookahead Range .....              | 127 |
| Creating Lookahead Business Rules .....      | 130 |
| Looping Business Rules .....                 | 134 |
| ForEach .....                                | 135 |
| Next .....                                   | 136 |
| ExitLoop .....                               | 136 |
| Extended Looping Example .....               | 137 |
| ODBC Business Rules .....                    | 138 |
| Setting up your ODBC Connection String ..... | 138 |
| DBOpen .....                                 | 140 |
| DBCclose .....                               | 142 |
| DBQuery .....                                | 143 |
| DBExecute .....                              | 145 |
| Run Business Rules .....                     | 146 |
| RunAlways .....                              | 146 |
| RunNoData .....                              | 148 |
| Substitute Business Rules .....              | 149 |
| DeleteSegment .....                          | 149 |
| InsertSegment .....                          | 150 |
| MakeKey .....                                | 150 |
| Substitute .....                             | 151 |
| SubstituteFind .....                         | 151 |
| SubstituteReplace .....                      | 152 |
| Utilities Business Rules .....               | 153 |
| AppendString .....                           | 153 |
| BuildString .....                            | 155 |
| ChangeCase .....                             | 157 |
| ChangeElmAttribute .....                     | 158 |

|  |     |
|--|-----|
| CheckFormat .....                          | 160 |
| CreateFSUID .....                          | 168 |
| DisplayErrorByNumber .....                 | 169 |
| FindString .....                           | 171 |
| GenerateFSUID .....                        | 173 |
| GetToken .....                             | 173 |
| Identify .....                             | 175 |
| IdentifierLookup .....                     | 177 |
| InsertIdentifier .....                     | 178 |
| Match .....                                | 178 |
| MatchApplList .....                        | 179 |
| Normalize .....                            | 181 |
| Numbers .....                              | 186 |
| OracleLookup and OracleLookupWithDate..... | 187 |
| OutputCTX .....                            | 189 |
| ReplaceChars .....                         | 189 |
| ReplaceString .....                        | 194 |
| SetCheckCTT and SetCheckCTTCount.....      | 196 |
| SetIdentifier .....                        | 197 |
| SubString.....                             | 198 |
| Trim .....                                 | 199 |
| TrimWhitespace .....                       | 200 |
| Variable Business Rules.....               | 202 |
| SetLocalVariable.....                      | 202 |
| SetVar .....                               | 203 |
| AddVar .....                               | 204 |
| Divide.....                                | 205 |
| DumpVars.....                              | 206 |
| Balance .....                              | 207 |
| CompareString and CompareStringNoCase..... | 209 |
| CompareNstring.....                        | 210 |
| CompareNumeric .....                       | 212 |
| Clear .....                                | 213 |
| ClearLocalVariable.....                    | 214 |
| FileTable Rules.....                       | 215 |
| GetInfo .....                              | 217 |
| GetLength .....                            | 219 |
| GetValueFromSegment .....                  | 220 |
| IsAlpha .....                              | 222 |
| IsAlphaNum.....                            | 223 |
| IsNum .....                                | 223 |
| SaveCurrentSegment.....                    | 226 |
| CheckCTT .....                             | 227 |
| FSVBExit.CheckDigit.....                   | 229 |
| X12 234-235 CheckDigit .....               | 230 |
| EDIFACT 3039-3055 CheckDigit .....         | 231 |
| Other CheckDigit Options .....             | 232 |
| User Defined Check Digit .....             | 232 |

|   |            |
|---|------------|
| DateTime.....   | 233        |
| FSVBExit.DisplayMessage .....                             | 234        |
| ProductUtilities.....                                     | 237        |
| <b>Appendix A: Variables</b>                              | <b>239</b> |
| Local Variables .....                                     | 239        |
| When to use Local Variables .....                         | 239        |
| Assigning a Local Variable .....                          | 239        |
| BusinessRules.Variable.....                               | 240        |
| When to use BusinessRules.Variable .....                  | 240        |
| Setting up BusinessRules.Variable .....                   | 240        |
| Good Variable Names .....                                 | 241        |
| Global Variables .....                                    | 242        |
| TIBCO Foresight-Defined Variables .....                   | 242        |
| Preprocessor Variables.....                               | 245        |
| Populating Variables with an External Variables File..... | 248        |
| Initializing and Clearing Variables and Lists .....       | 249        |
| Variable Maps.....  | 251        |
| <b>Appendix B: Validator Error Messages</b>               | <b>255</b> |
| Viewing TIBCO Foresight-Supplied Error Messages .....     | 255        |
| Creating and Viewing your own Error Messages.....         | 255        |
| Using your own Error Messages.....                        | 256        |
| Troubleshooting Custom Error Messages.....                | 259        |
| <b>Appendix C: Code Tables</b>                            | <b>261</b> |
| Setting up your own Code Tables.....                      | 261        |
| Example A: External Code Table File .....                 | 262        |
| Extending existing HIPAA Code Tables .....                | 263        |
| <b>Appendix D: Complicated Rules</b>                      | <b>265</b> |
| Simple Rules.....   | 265        |
| Complex Rules .....                                       | 266        |
| Example 1: Two Conditions create an Error Message.....    | 266        |
| Example 2: Using Rules in Loops .....                     | 268        |
| Example 3: Adding and Comparing Numeric Values .....      | 269        |
| <b>Appendix E: ODBC Examples</b>                          | <b>271</b> |
| ODBC Tutorials and Demos .....                            | 271        |
| ODBC Example 1.....                                       | 272        |
| Setting up a system DSN .....                             | 272        |
| Looking at the Database .....                             | 274        |
| Setting up the Error Message .....                        | 275        |
| Creating the Rules.....                                   | 276        |
| Testing the Rules .....                                   | 278        |
| ODBC Example 2.....                                       | 279        |
| Running the Demo in HIPAA Validator Desktop.....          | 280        |
| Running the Demo in Instream .....                        | 280        |

|   |            |
|---|------------|
| The Rules that made it happen .....                     | 281        |
| <b>Appendix F: Guideline Merge</b>                      | <b>285</b> |
| Overview .....  | 285        |
| <b>Appendix G: Debug</b>                                | <b>287</b> |
| EDISIM Validator Debug .....                            | 287        |
| HIPAA Validator Desktop and Instream Debug .....        | 287        |
| <b>Appendix H: Troubleshooting Checklist</b>            | <b>289</b> |
| <b>Appendix I: Processing Order</b>                     | <b>291</b> |
| <b>Appendix J: LookAhead and Array Extended Example</b> | <b>295</b> |
| Array, Lookahead, and Web Services Demos .....          | 295        |
| Summary of Rules – Top-Down .....                       | 296        |
| Annotated Summary of Rules in Execution Order .....     | 300        |
| <b>Appendix K: Building Business Rules</b>              | <b>313</b> |
| Overview .....  | 313        |
| Text Button.....  | 313        |
| Entry Examples .....                                    | 314        |
| <b>TIBCO Documentation and Support Services</b>         | <b>319</b> |
| How to Access TIBCO Documentation .....                 | 319        |
| Product-Specific Documentation .....                    | 319        |
| How to Contact TIBCO Support .....                      | 320        |
| How to Join TIBCO Community .....                       | 320        |
| <b>Legal and Third-Party Notices</b>                    | <b>321</b> |

# Introduction

---

## Document Purpose

This document describes the external routines used to create business rules with TIBCO Foresight® EDISIM®'s Standards Editor.

Business rule basics are in **TIB\_fsp\_edisim\_<n.n>\_fseditor.pdf** in EDISIM's Documentation directory.

## Intended Audience

This document is intended for advanced users who are familiar with:

- EDISIM Standards Editor.
- The EDI guidelines for which rules are to be developed.
- If the rules are to be used with EDISIM's Analyzer, then familiarity with that product is assumed.
- If the rules are to be used with TIBCO Foresight® Instream® or TIBCO Foresight® HIPAA Validator® Desktop, then familiarity with these products is assumed.

Before using this document, familiarize yourself with basics of business rules as described in the Standards Editor manual, **TIB\_fsp\_edisim\_<n.n>\_fseditor.pdf**.

You can open this manual from within Standards Editor by choosing **Help | View the Manual**.



# What you need before using Business Rules

You will need to install the following before using the business rules described in this document:

- EDISIM
- Instream or HIPAA Validator Desktop

## Big Picture

You can create your own business rules and have HIPAA Validator Desktop, Instream, or EDISIM Analyzer enforce them when it checks EDI files for compliance. These rules allow you to add additional checking beyond that done by TIBCO Foresight-distributed guidelines.

### Definitions

---

#### **TIBCO Foresight guidelines**

Guidelines that ship with EDISIM, Instream, and HIPAA Validator Desktop.

#### **HIPAA**

For those using TIBCO Foresight HIPAA validation products, these guidelines in EDISIM contain types 1 and 2 edits. HIPAA-based TIBCO Foresight guidelines in Validator include these same EDISIM guidelines plus additional guidelines with types 1-6 edits. Refer to **Guideline\_Reference\_Manual.pdf**.

#### **User guidelines**

Guidelines that you create in EDISIM Standards Editor. They contain your company's edits.

#### **Production guideline**

Guidelines for Instream and HIPAA Validator Desktop that you create by merging a user guideline with a TIBCO Foresight guideline. HIPAA-based Production guidelines contain type 1-7 edits.

### Major Steps for HIPAA validation users

---

To create a production guideline for Validator that includes both user *and* HIPAA rules:

1. Use EDISIM's Standards Editor to create a user guideline containing your own business rules.
2. Merge the user guideline containing your business rules with the corresponding TIBCO Foresight types 1-6 guideline in Instream or HIPAA Validator Desktop. This creates a production guideline with both user rules and HIPAA rules.
3. With HIPAA Validator Desktop or Instream, validate EDI data against the production guideline.

To create a guideline for Instream or HIPAA Validator Desktop that includes only *user* rules:

1. Use EDISIM's Standards Editor to create your own business rules in a user guideline.
2. Copy the user guideline's STD file from EDISIM's User Files\Public Guidelines directory and paste it into Instream or HIPAA Validator Desktop's **Database** directory.
3. With HIPAA Validator Desktop or Instream, test your own rules by checking EDI data against the new guideline.

### Major Steps for EDISIM Analyzer users

---

1. Use EDISIM's Standards Editor to create a user guideline containing your own business rules.
2. With EDISIM Analyzer, check EDI data against the guideline. You will see diagnostic messages created by the business rules.

## Viewing TIBCO Foresight-supplied HIPAA Rules

---

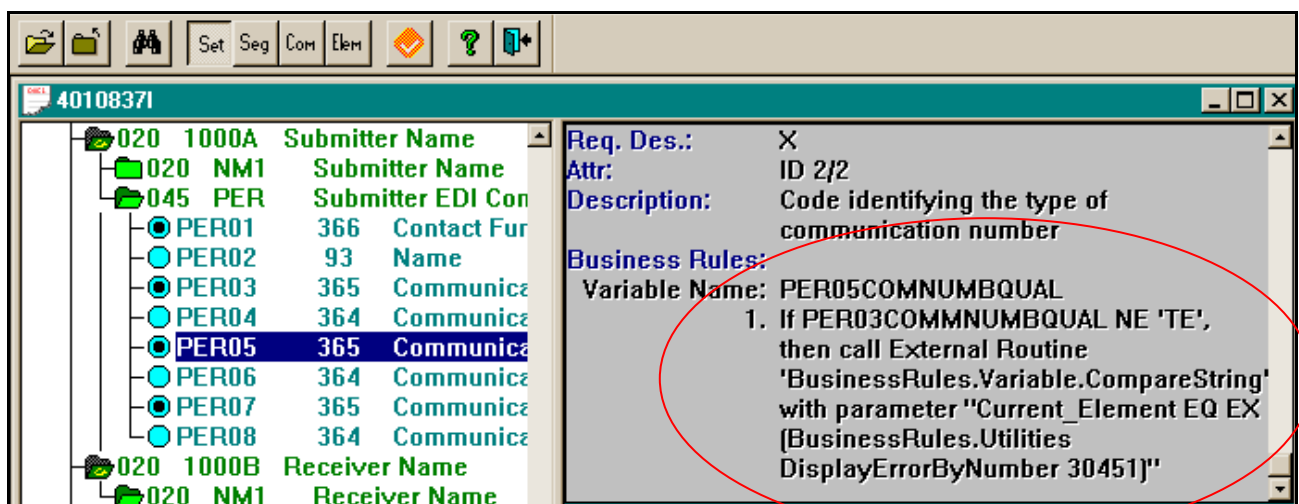
### Validator Only

---

TIBCO Foresight-supplied HIPAA business rules are not visible in Standards Editor, but you can see them in HIPAA Validator Desktop's Library. Click on the segment, composite, or element in Library's top left pane and look in the upper right corner.

For example:

1. Open Library with **Start | Programs | Foresight | Desktop | Library**.
2. Open the HIPAA guideline **4010837I**.
3. Open loop **1000A** and the **PER** segment at position 045 (Submitter EDI Contact Information) and click on element **365** (Communication Number Qualifier) at PER05.
4. Scroll down in the upper right pane to see this business rule:



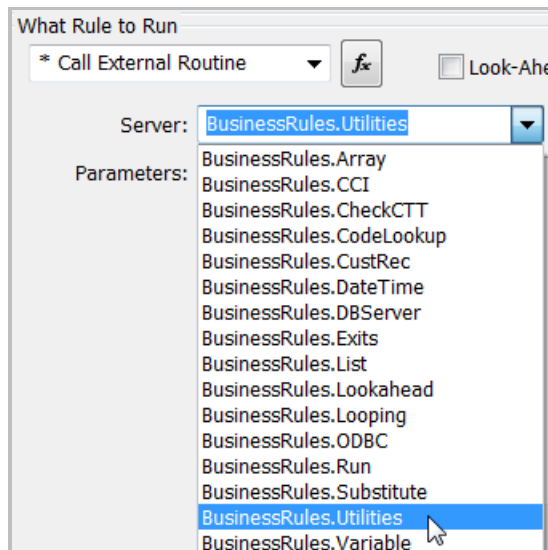
# Using External Routines

To create a rule that uses an external routine in Standards Editor, you have several options:

- \*Call External Routine See page [4](#)
- Type the Rule See page [5](#)
- Use the Parameters Grid See page [6](#)

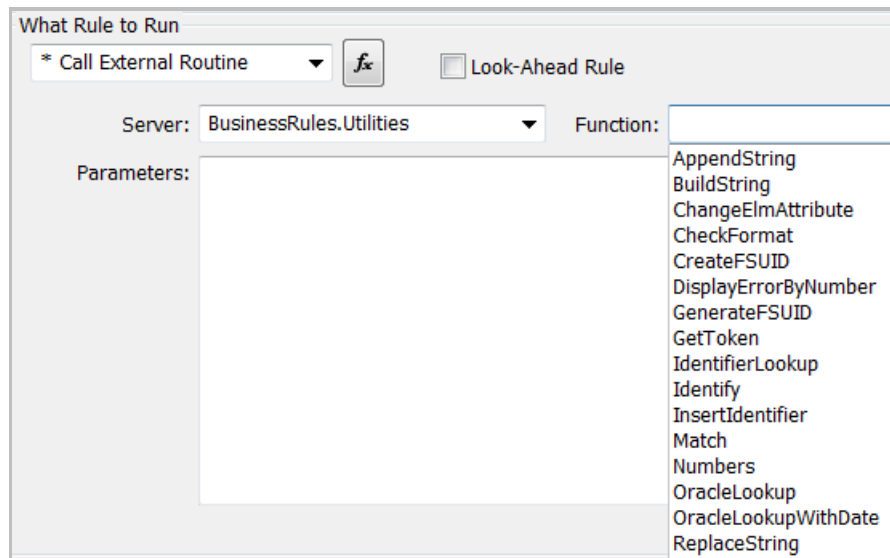
## \*Call External Routine

1. Open the business rules box (right-click on the segment, composite, or element and choose Business Rules).
2. Click **New** to bring up the Condition and Rule Definition dialog box. Set up the When to Run the Rule area at the top of the box, if needed.
3. In the What Rule to Run area, select **\*Call External Routine** in the drop box.
4. Click the down-arrow in the **Server** field and click the one of your choice. These are described in detail in the rest of this document.



If it is not in the list, type it in in the Server line.

- Click the down-arrow in the **Function** field and click the one of your choice.



If the function is not there, type it in the Function line.

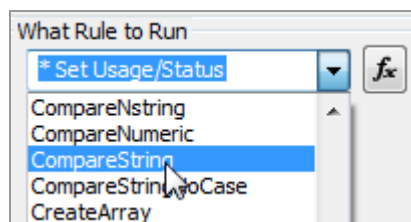
- If the function requires parameters, type them in the **Parameters** field.
- Click **OK**.

View the rule in the Business Rules dialog. If you need to change it, click **Edit** and make the changes.

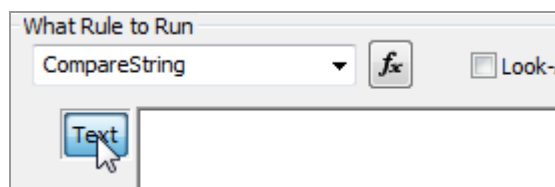
- Click **OK** until you have exited all dialog boxes.

## Type the Rule

- Open the business rules box (right-click on the segment, composite, or element and choose Business Rules).
- Click **New** to bring up the Condition and Rule Definition dialog box. Set up the When to Run the Rule area at the top of the box, if needed.
- In the What Rule to Run area, select the rule from the drop list:



- Click the **Text** button until you get an empty box where you can type the parameters:



- Referring to the parameters for the rule in Business Rules Reference, type the parameters, paying particular attention to capitalization and parentheses:

What Rule to Run

CompareString  ☐ Look-Ahead Rule

Current\_Element EQ "1" (BusinessRules.Utilities DisplayErrorByNumber 32001)

- Click **OK** until you have exited all dialog boxes.

## Use the Parameters Grid

- Open the business rules box (right-click on the segment, composite, or element and choose Business Rules).
- Click **New** to bring up the Condition and Rule Definition dialog box. Set up the When to Run the Rule area at the top of the box, if needed.
- In the What Rule to Run area, select the rule from the drop list:

What Rule to Run

\* Set Usage/Status

CompareNstring  
CompareNumeric  
CompareString  
CompareStringIgnoreCase  
CreateArray

- Click the **Text** button until you get a parameters grid:

What Rule to Run

CompareString  ☐ Look-Ahead Rule

| Parameter Name | Parameter Value |
|----------------|-----------------|
| Value1         | «Value1»        |
| Operator       | «Operator»      |
| Value2         | «Value2»        |
| ThenRule       |                 |

- Click in the first Parameter Value line and read the Parameter Notes to the right. Replace the contents of the line with the parameter value you want to use:


What Rule to Run

CompareString  ☐ Look-Ahead Rule

| Parameter Name | Parameter Value |
|----------------|-----------------|
| Value1         | Current_Element |
| Operator       | «Operator»      |
| Value2         | «Value2»        |


6. Repeat for the other parameter values:

What Rule to Run

CompareString  ☐ Look-Ahead Rule

| Text | Parameter Name | Parameter Value |
|------|----------------|-----------------|
|      | Value1         | Current_Element |
|      | Operator       | EQ              |
|      | Value2         | "1"             |
|      | ThenRule       |                 |

7. You cannot type in a line that has an **fx** to the right:

| Parameter Name | Parameter Value | LA  |
|----------------|-----------------|---|
| Value1         | Current_Element |   |
| Operator       | EQ              |   |
| Value2         | "1"             |   |
| ThenRule       |                 |  |

Instead, click the **fx** and then \*All, or one of the categories:

Select Rule

- \* Common Rules
  - \* All
  - Array
  - Builtin
  - CCI


Pick the sub-rule and click **Select**:

Select Rule

- DateCalc3
- DateCalc4
- DateCalc5
- DefineCustomRec
- DeleteSegment
- DisplayErrorByNumber
- Divide

You will then have additional parameters in the grid for that sub-rule:

What Rule to Run

CompareString  ☐ Look-Ahead Rule

| Text | Parameter Name | Parameter Value      |
|------|----------------|----------------------|
|      | Value1         | Current_Element      |
|      | Operator       | EQ                   |
|      | Value2         | "1"                  |
|      | ThenRule       | DisplayErrorByNumber |
|      | > ErrorNumber  | 32001                |
|      | > Severity     |                      |

The Parameter Names for a sub-rule are prefixed with > to indicate their depth:

|               |       |
|---------------|-------|
| > ErrorNumber | 32001 |
| > Severity    |       |
| > MessageText |       |

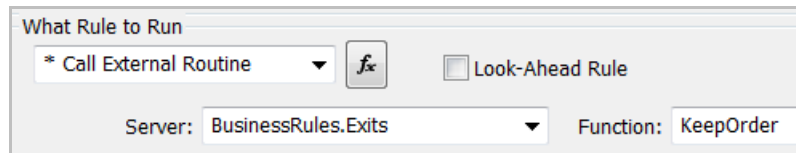
8. When finished, click **OK** until you have exited all dialog boxes.

The Look-Ahead Rule checkboxes are explained in Lookahead Business Rules on page [123](#).


## Registering new Business Rules

Two possibilities:

- Until new rules are registered, select \*Call External Routine at the top, and then type the Server and Function in the fields provided.



What Rule to Run

\* Call External Routine  ☐ Look-Ahead Rule

Server: BusinessRules.Exits Function: KeepOrder

- Copy FSBRD.dat (if present) from Instream's Bin directory into EDISIM's Bin directory, or request a copy from TIBCO Foresight Support. Then restart Standards Editor.

# Tutorials: Set up your own Rule

---

## HIPAA business rule tutorial

### Overview

This section shows you how to set up a rule that checks the transaction's date to be sure that it is not in the future. Where it is in the future, you'd like an error message to be displayed:

```
Transaction date cannot be in the future.
```

Like most rules, this one has two parts:

A **condition**     *If* the transaction date is in the future.

An **action**        *Then*, issue an error message.

To create an example business rule, we'll take these steps:

1. Create your own error message .....page [10](#)
2. Set up a rule .....page [10](#)
3. Copy the guideline from EDISIM to Instream .....page [10](#)
4. Test the rule.....page [11](#)
5. Merge your business rule with a HIPAA guideline.....page [11](#)



## Create your own error message

Use the instructions in Creating and Viewing your own Error Message on page [255](#) to create your own error message number 32210 with this text:

Transaction date cannot be in the future.

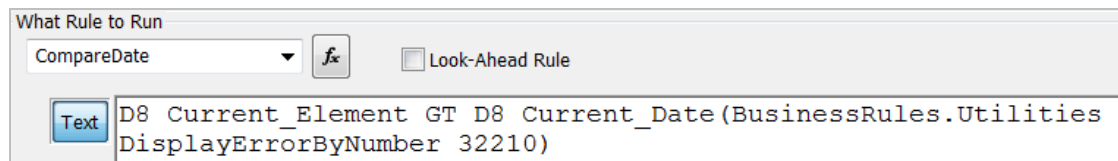
## Set up a rule

You will create a rule to:

- See if the transaction date is in the future.
- If so, issue error message 32210.

To create this rule, start an 837I guideline from 837AQ320 in Standards Editor and:

1. Click on the **BHT-04**.
2. Choose **Edit | Advanced | Business Rules | New | Always**.
3. In the What Rule to Run area, choose **CompareDate** from the drop box.
4. Fill out the bottom right as follows, using the same capitalization and spacing:



What Rule to Run

CompareDate  ☐ Look-Ahead Rule

D8 Current\_Element GT D8 Current\_Date (BusinessRules.Utilities DisplayErrorByNumber 32210)

Where:

D8 Current\_Element GT D8  
Current\_Date

See if the date in the current element is greater than today's date.

If true, continue by executing the rest of the rule.

(BusinessRules.Utilities  
DisplayErrorByNumber 32210)

Display the text of error number 32210.

5. Close the dialog boxes by clicking **OK** twice.
6. Save the guideline as **RULESNEW**.

## Copy the guideline from EDISIM to Instream

You can test the rule using EDISIM Validator.

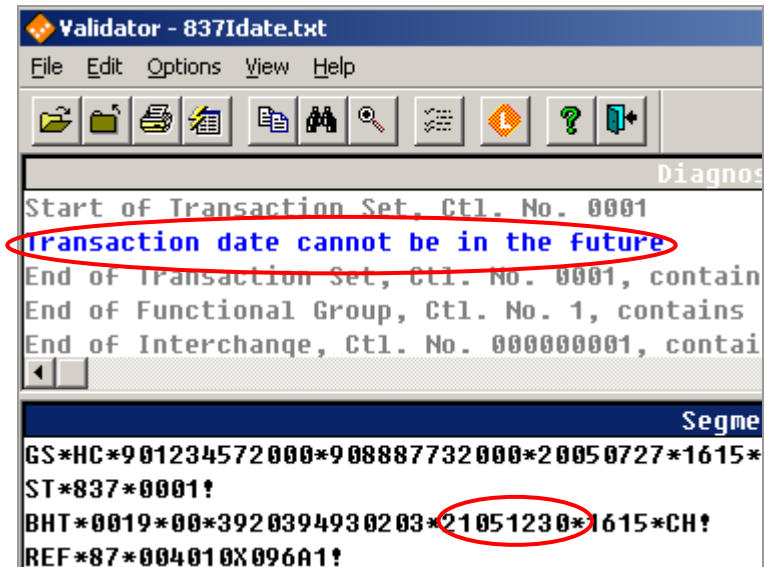
Instead, if you'd like, you can copy the **RULESNEW.STD** file in EDISIM's **User Files\Public Guidelines** directory and paste it in HIPAA Validator Desktop's or Instream's **Database** directory.

## Test the rule

When validating an EDI file against the RULESNEW guideline, your rule will be enforced.

To test this, validate **837I\_4010\_H\_futureBHT04.txt** or **837Idate.txt** in the DemoData directory of Instream or HIPAA Validator Desktop (Use the product that has your customized message 32210).

You should see the error message on the BHT segment, since the BHT-04 date is in the future.



## Merge your business rule with a HIPAA guideline

HIPAA users:

Once you have verified that your business rule is operating correctly, you can merge it with the corresponding HIPAA guideline, thus creating a guideline that will test both your rules and HIPAA types 1-6 rules at the same time.

See Appendix F: Guideline Merge on page [285](#) for details.

## X12 business rule tutorial

### Overview

You will create a rule for a 4010 850 transactions to:

- See if Table 1's REF-01 contains 06.
- If not, issue this error message:

This REF-01 must contain a value of 06.

## Set up a rule

To create this rule, start a 4010 850 guideline in Standards Editor and:

- 1 Click on the **050 REF-01**.
- 2 Choose **Edit | Advanced | Business Rules | New | Always**.
- 3 In the What Rule to Run area, choose **CompareString** from the drop box.
- 4 Fill out the rule input area as follows, using the same capitalization and spacing:

Where:

Current\_Element NE "06"

If the data in the current element is not 06, then continue by executing the rest of the rule.

(BusinessRules.Utilities  
DisplayErrorByNumber 0 0 "This REF-01  
must contain a value of 06)

Display this message during analysis.

- 5 Close the dialog boxes by clicking **OK** twice.
- 6 Save the guideline as **RULESNEW**.

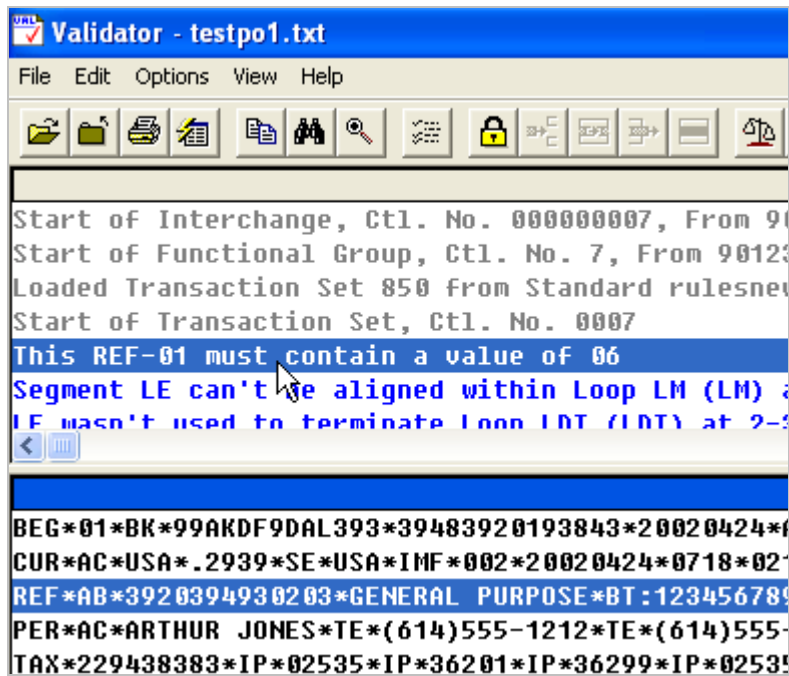
## Test the rule

When validating an EDI file against the RULESNEW guideline, your rule will be enforced.

To test this, open EDISIM's Validator and check **testpo1.txt** in EDISIM's Samples directory:



You should see the error message on the REF segment, since the REF-01 does not contain 06:





# Analyzer vs. other Validation Programs

---

## Overview

EDISIM Analyzer is a legacy product that supports validation of most, but not all of the standards supported by Instream, HIPAA Validator Desktop, and EDISIM Validator. For example, Analyzer does not validate HL7, XML, or Flat File data.

Analyzer does, however, support some standards that are not handled by the more recent products such as GENCOD and ODETTE. (Refer to **FileFormatsAtForesight.pdf** for a complete list.)

Due to these differences:

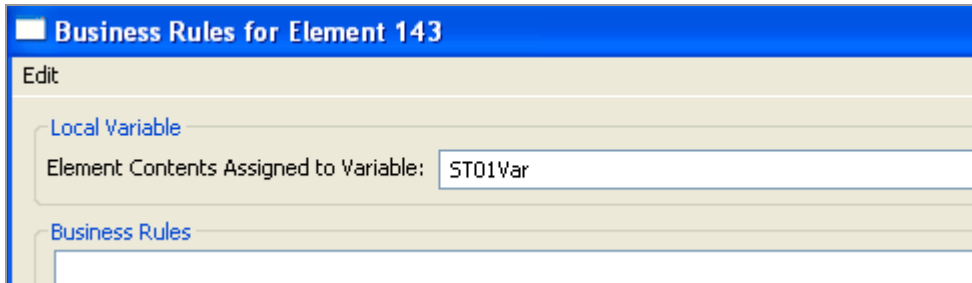
- not all Business Rule functionality can be used with Analyzer. For example, reserved variables cannot be used.
- not all Business Rules can be used with Analyzer. For example, Array Business Rules cannot be used.
- Some Business Rules **only** apply to Analyzer. For example, the rule CheckCTT is Analyzer-only.

The following sections describe what business rules can be used with Analyzer.

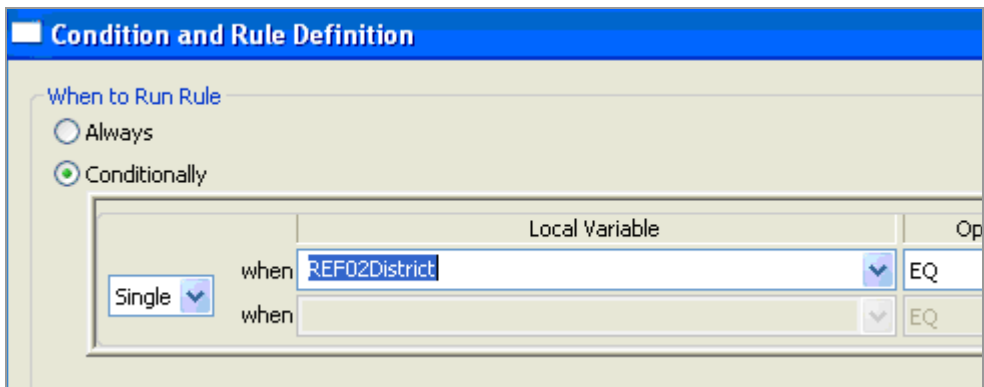
|   |         |
|---|---------|
| Local Variables .....                         | page 16 |
| Condition and Rule Definition Dialog Box..... | page 17 |
| External Routines .....                       | page 18 |

# Local Variables

Local variables are defined in the Local Variable area of the main business rules box. This appears when you are on an element.



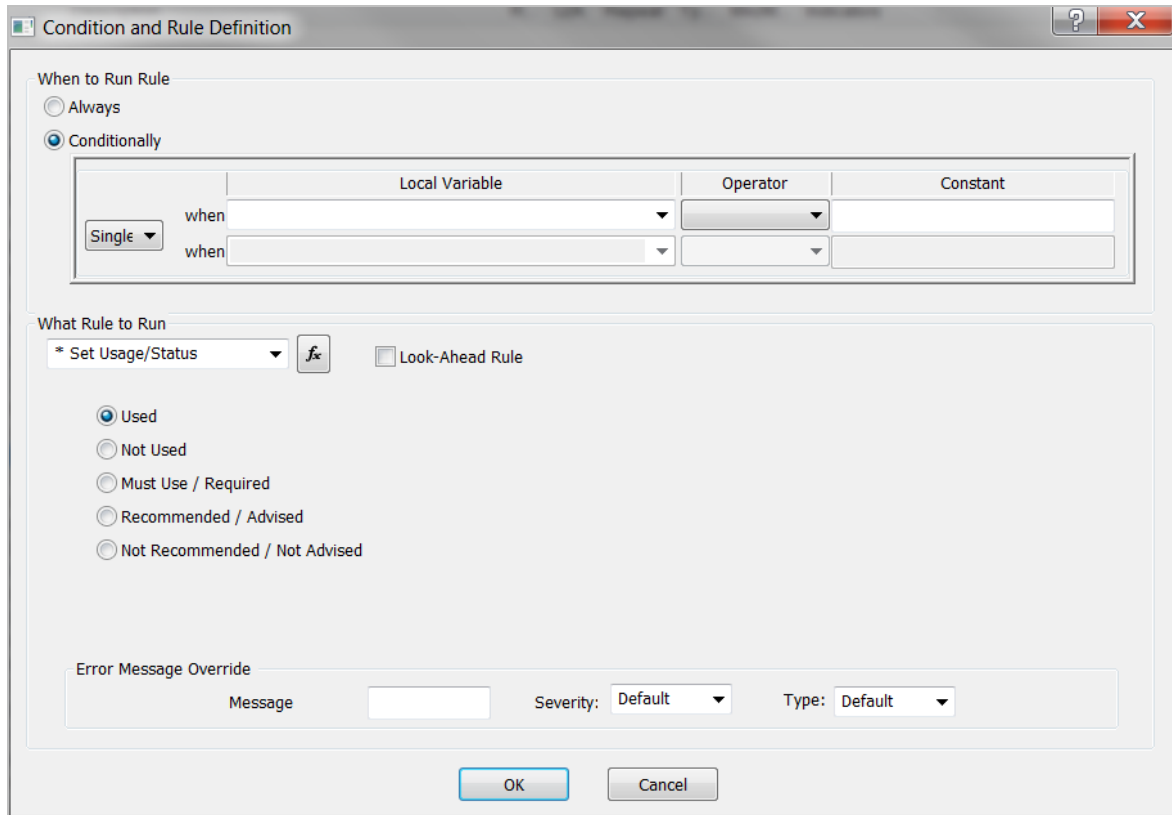
They work in business rules with all TIBCO Foresight validation programs:



See Local Variables on page [239](#) for details about where to use a local variable.

# Condition and Rule Definition Dialog Box

Features of this box are available with all TIBCO Foresight validation programs.



The dialog box is titled "Condition and Rule Definition". It contains the following sections:

- When to Run Rule:** Includes radio buttons for "Always" and "Conditionally" (selected). Below "Conditionally" is a table for defining conditions.
- What Rule to Run:** Includes a dropdown menu set to "\* Set Usage/Status", a formula icon, and a checkbox for "Look-Ahead Rule". Below these are radio buttons for rule status: "Used" (selected), "Not Used", "Must Use / Required", "Recommended / Advised", and "Not Recommended / Not Advised".
- Error Message Override:** Includes a "Message" text field, a "Severity" dropdown set to "Default", and a "Type" dropdown set to "Default".

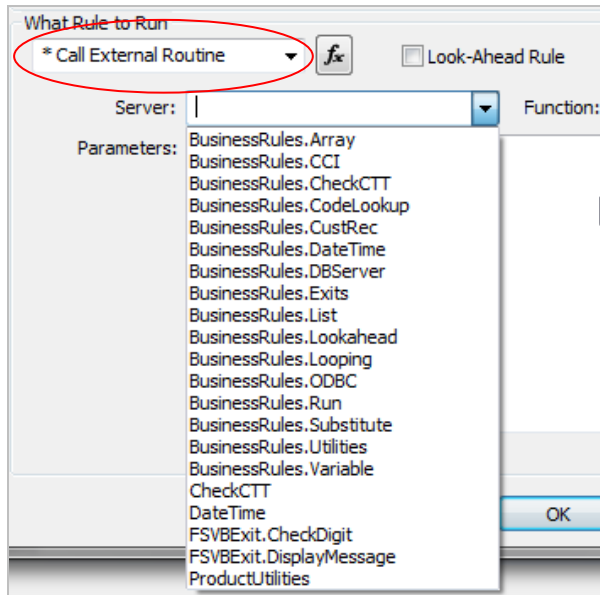
At the bottom are "OK" and "Cancel" buttons.

|          | Local Variable | Operator | Constant |
|----------|----------------|----------|----------|
| Single ▼ | when           |          |          |
|          | when           |          |          |



# External Routines

Most external routine business rules can be used with Analyzer.



The exceptions for Analyzer are as follows:

| External Routines |                         |  |
|-------------------|-------------------------|--|
| Rule Type         | Server Name             | Exception(s)   |
| Exits             | BusinessRules.Exits     | Not available:<br>UserExitWithWait<br>UserExitWithoutWait  |
| Utilities         | BusinessRules.Utilities | Note the following: <ul style="list-style-type: none"> <li>CheckFormat does not look up the first three digits of the SSN.</li> <li>DisplayErrorByNumber only supports explicit text.</li> <li>IdentifierLookup, OracleLookup, OracleLookupWithDate, Substitute rules, and SetIdentifier are ignored by Analyzer.</li> </ul> |
| Variable          | BusinessRules.Variable  | Not available:<br>DumpVars   |

# Business Rules Reference

---

## Overview

EDISIM Validator, HIPAA Validator Desktop, and Instream support all business rules except where noted otherwise in the Business Rules Reference section.

### ***EDISIM Analyzer vs. other Validation Programs***

EDISIM Analyzer is a legacy product that supports validation of many, but not all of the standards supported by Instream, HIPAA Validator Desktop, and EDISIM Validator. For example, Analyzer does not validate HL7, XML, or Flat File data.

Analyzer does, however, support some standards that are not handled by the more recent products such as GENCOD and ODETTE. (Refer to **FileFormatsAtForesight.pdf** for a complete list.)

Because of these differences:

- Not all Business Rule functionality can be used with Analyzer. For example, reserved variables cannot be used.
- Not all Business Rules can be used with Analyzer. For example, Array Business Rules cannot be used.
- Some Business Rules **only** apply to Analyzer. For example, the rule CheckCTT is Analyzer-only.

Throughout this section, when Analyzer support is different it is noted as follows:

---

**All validators except Analyzer**

---

or

---

**Analyzer Only**

---

# Reserved Variables

Reserved variables are ones that do not have to be assigned in order for the system to determine the value. They include

|                                |         |
|--------------------------------|---------|
| Current_Date .....             | page 21 |
| Current_Delim .....            | page 23 |
| Current_Element .....          | page 24 |
| Current_ErrCount .....         | page 25 |
| Current_LoopCount.....         | page 27 |
| Current_LoopKey.....           | page 27 |
| Current_Row and Next_Row ..... | page 27 |
| Current_Time .....             | page 28 |
| GLOBAL_FILENAME.....           | page 29 |
| GLOBAL_FILEPATHNAME.....       | page 30 |

Except where noted otherwise, these can be used in business rule parameters anywhere a variable can be used.

Do not attempt use these reserved names for other purposes.

## Current\_Date

---

### All validators except Analyzer

---

Represents today's date in your choice of formats.

#### Format

`Current_Date_format` (case sensitive)

#### Where:

`Current_Date_` Literal text.

`format` Optional date format containing any combination of:

CC century  
YY year  
MM month  
DD day  
HH hour on 24-hour clock  
mm minute (lower case mm)  
SS (second)

*separators* (slash, hyphen, or colon)


If format is omitted, date will be in CCYYMMDD format.

| In business rule:                              | Result if date is May 19, 2016<br>time is 1:45 PM) |
|--|--|
| <code>Current_Date</code>                      | 20160519   |
| <code>Current_Date_CCYY</code>                 | 2016   |
| <code>Current_Date_YYMMDD</code>               | 160519   |
| <code>Current_Date_ CCYY/MM/DD/HH/mm/SS</code> | 2016/05/19/13/45/00                                |

#### Example 1

If the value in the current element (which is a date in format D8) is later than the current date, then display error message 32210.

What Rule to Run

CompareDate 

Text D8 Current\_Element GT D8 Current\_Date (BusinessRules.Utilities DisplayErrorByNumber 32210)


### Example 2

This uses the current date in an error message by surrounding it with %. The message must be coded into the business rule rather than in the CustomerFSBRErrs.txt file.

|                      |   |
|----------------------|---|
| What Rule to Run     |   |
| DisplayErrorByNumber |  |
| Text                 | 0 0 "Order %Current_Element% was received on %Current_Date% at %Current_Time%"    |

### Example 3

This is similar to Example 2, but combines the date and time:

|                      |   |
|----------------------|---|
| What Rule to Run     |   |
| DisplayErrorByNumber |  |
| Text                 | 0 0 "Transaction was received on %Current_Date_CCYYMMDDHHMM%"                     |

If today is June 19, 2016 at 2:53 p.m., output will look like this:

Transaction was received at 201606191453

## Current\_Delim

---

### All validators except Analyzer

---

Returns the specified delimiter character.

If the character is a control character, then the corresponding number will be returned. For example, if NewLine is the segment terminator, then Current\_Delim will return '13'. If a delimiter is not assigned a value, '-1' will be returned.

#### Format

Current\_Delim(*“delimiter”*) *(case sensitive)*

#### Where:

Current\_Delim Literal text.

*delimiter* One of these; include the surrounding parentheses and double quotes:

|        |                                    |
|--------|------------------------------------|
| SEG    | <i>Segment terminator</i>          |
| ELM    | <i>Element delimiter</i>           |
| SUBELM | <i>Subelement delimiter</i>        |
| REPELM | <i>Repeating element delimiter</i> |
| DEC    | <i>Decimal point character</i>     |
| ESC    | <i>Escape character</i>            |

**Example.** This rule, which would typically be used in an EDIFACT message, sets variable DPCHAR to a period or a comma, depending on the current decimal place character as specified in the UNA Service String Advice segment.



This shows the decimal character that should be used:



The message shows that the decimal character is a period:

```
EMSG      4The decimal character is .
```

## Current\_Element

---

### All validators except Analyzer

---

Represents the value in the current element or field.

The rule should be applied to an item that directly holds data (EDI element, field, etc.) rather than a segment, record, or other structure.

#### Format

`Current_Element` *(case sensitive)*

**Examples:** This example places the value in the current element into variable HI0102IndustryCode:

The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "SetVar" selected and a button with a function symbol (fx). Below the dropdown is a text box containing "HI0102IndustryCode Current\_Element".

This example takes the first character of the value in the current element and places it in the variable IDCode:

The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "SubString" selected and a button with a function symbol (fx). Below the dropdown is a text box containing "IDCode Current\_Element 1 1".

## Current\_ErrCount

---

### All validators except Analyzer

---

Represents the current total number of errors, up to the point where the variable is used, for the specified severities or types.

#### Format

`Current_ErrCount(criteria)` *(case sensitive)*

#### Where:

|                               |  |
|-------------------------------|--|
| <code>Current_ErrCount</code> | Literal text.  |
| <code>(criteria)</code>       | Code defining the scope of the errors followed by the types or severities. This must be within parentheses and immediately follow ErrCount with no spaces. |

---

#### Criteria

---

|                |   |
|----------------|---|
| <b>B</b>       | Optional<br><br>For <b>Batch</b> . Errors in this batch (the entire file), up to this point, are counted.<br><br>If B is omitted, only errors in this transaction or message are counted.   |
| <b>T</b>       | Optional<br><br><b>T</b> ypes. This string of digits represents HIPAA type numbers.<br><br>If T is omitted, they are severity numbers.<br><br>T and B can be in any order, and either or both can be used or omitted.                         |
| <i>numbers</i> | A string of digits specifying which types or severities should be included in the counts. If preceded by a T, they are considered types and can be 0-8. Otherwise, they are severities and can be 0-6. Please see <b>APF.pdf</b> for details. |



### Examples:

|  |   |
|--|---|
| <code>Current_ErrCount("34")</code>      | The total number of severity 3 and 4 errors in the transaction up to this point.                  |
| <code>Current_ErrCount("T34")</code>     | The total number of type 3 and 4 errors in the transaction up to this point.                      |
| <code>Current_ErrCount("B34")</code>     | The total number of severity 3 and 4 errors in the batch up to this point.                        |
| <code>Current_ErrCount("BT34")</code>    | The total number of type 3 and 4 errors in the batch up to this point. (TB34 would work as well). |
| <code>Current_ErrCount("0123456")</code> | The total number of all severity errors in the transaction up to this point.                      |
| <code>Current_ErrCount("B12")</code>     | The total number of Type 1 and 2 errors in the batch up to this point.                            |

### Examples

This puts the count of the number of severity 3 and 4 errors encountered in the transaction set into variable ERRCOUNTA.

What Rule to Run

SetVar

Text: `ERRCOUNTA Current_ErrCount("34")`

This displays “Type 1 and 2 Errors Encountered!” if any Type 1 or 2 errors were encountered in the batch before this point:

What Rule to Run

CompareNumeric

Text: `Current_ErrCount("TB12") GT "0" (BusinessRules.Utilities DisplayErrorByNumber 0 0 "Type 1 and 2 Errors encountered!")`

This displays the total number of type 1 and 2 errors in the batch up to this point: “12 Errors Encountered!” Since `%Current_ErrCount%` does not have the criteria after it, the previous criterion of TB12 is assumed.

What Rule to Run

CompareNumeric

Text: `Current_ErrCount("TB12") GT "0" (BusinessRules.Utilities DisplayErrorByNumber 0 0 "Type 1 and 2 Errors Encountered!")`

## Current\_LoopCount

---

**All validators except Analyzer**

---

Represents the current iteration of the current loop in GetInfo business rules (see page [217](#)).

## Current\_LoopKey

---

**All validators except Analyzer**

---

Represents the loop ID and current iteration of the current loop in GetInfo business rules (see page [217](#)).

## Current\_Row and Next\_Row

---

**All validators except Analyzer**

---

Represents the current or next array row index in some array business rules. Please see Array Reserved Variables on page [35](#).

## Current\_Time

---

### All validators except Analyzer

---

Represents the current time in your choice of formats.

#### Format (*case sensitive*)

`Current_Time_format`

#### Where:

`Current_Time`      Literal text.

`format`      Optional time format containing any combination of:

HH

mm

SS

separators (slash, hyphen, or colon)



If format is omitted, time will be in HH:MM:SS format.


| In business rule                    | Result if time is 1:45 PM |
|-------------------------------------|---------------------------|
| <code>Current_Time</code>           | 13:45:00                  |
| <code>Current_Time_HHMM</code>      | 1345                      |
| <code>Current_Time_HHMMSS</code>    | 134500                    |
| <code>Current_Time_ MM/SS/HH</code> | 45/00/13                  |

#### Example

Put the current time in HHMMSS format into variable Potime:

What Rule to Run

SetVar  

 Potime Current\_Time\_HHMMSS

## GLOBAL\_FILENAME

---

### All validators except Analyzer

---

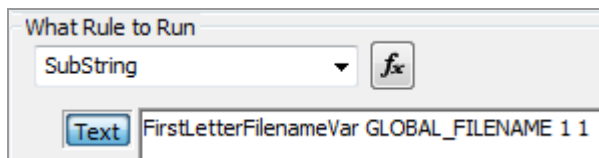
Represents the name of the file being validated. For example, this might contain the value `File.txt`.

**Format** (*case sensitive*)

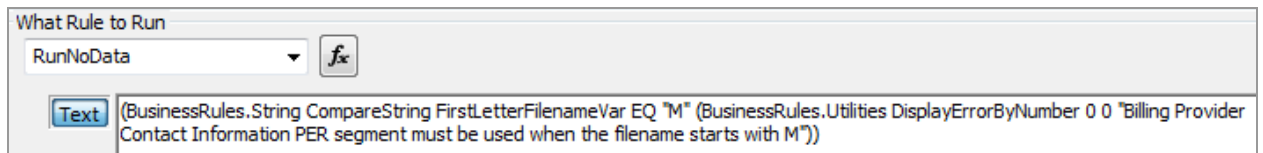
GLOBAL\_FILENAME

### Example

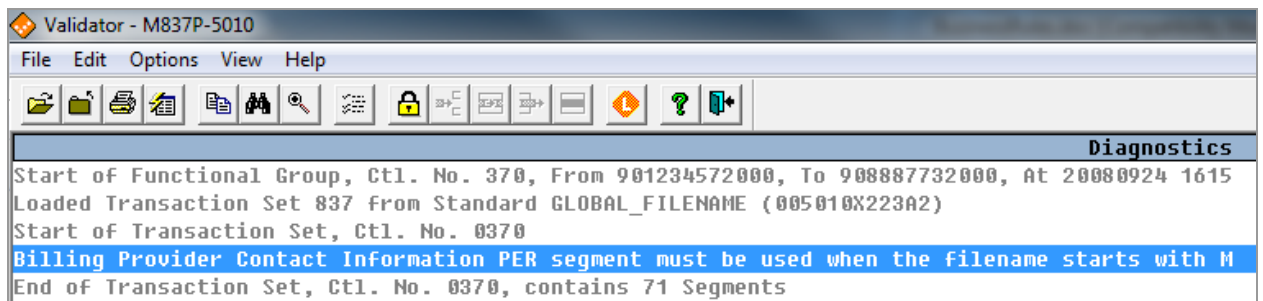
We store the first letter of the filename in a variable:



We put this on a certain PER segment. This displays a message if the segment is missing and the filename starts with **M**.



Result in HIPAA Validator Desktop when the PER segment is missing and the filename starts with M:



## GLOBAL\_FILEPATHNAME

---

### All validators except Analyzer

---

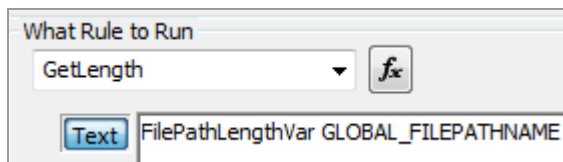
Represents the path and name of the file being validated. For example, this might contain the value `c:\837P\File.txt`.



**Format** (*case sensitive*)

GLOBAL\_FILEPATHNAME

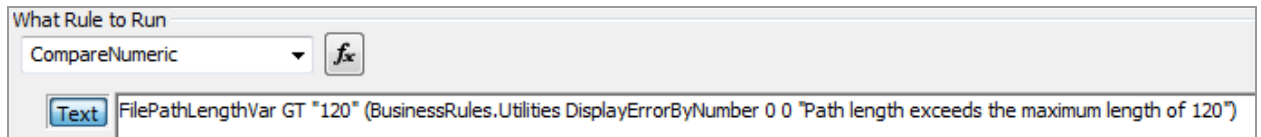
### Example



We store the length of the path in a variable:



What Rule to Run  
GetLength   
 FilePathLengthVar GLOBAL\_FILEPATHNAME

If it is longer than 120 characters, we issue a message:



What Rule to Run  
CompareNumeric   
 FilePathLengthVar GT "120" (BusinessRules.Utilities.DisplayErrorByNumber 0 0 "Path length exceeds the maximum length of 120")

## Using Reserved Variables in a Message

If you are hard-coding the message into the business rule, surround them with percent signs, like this:



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu set to "DisplayErrorByNumber" and a button with a function symbol (fx). Below this, there is a "Text" label and a text area containing the message: "0 0 "Order %Current\_Element% was received on %Current\_Date% at %Current\_Time%".

If the message is in CustomerFSBRErrs.txt:



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu set to "DisplayErrorByNumber" and a button with a function symbol (fx). Below this, there is a "Text" label and a text area containing the message: "32005".

Use # instead of % in the CustomerFSBRErrs.txt message:

```
32005 Order #Current_Element# received on #Current_Date# at #Current_Time#
```

# Literals

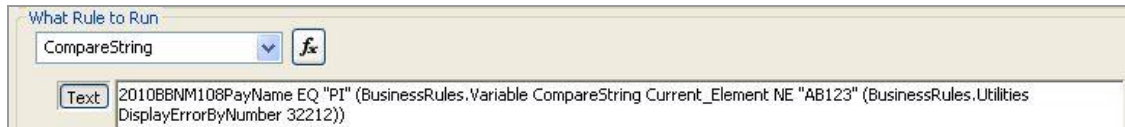
---

## All validation programs

---

You can use literal values to insert specific values into rules. Literal values are normally surrounded with double quotes.

**Example:** If the content of variable 2010BBNM108PayName equals the literal value **PI** and the content of Current\_Element does not equal the literal value **AB123**, then display error message 32212.



## Escape Character for Double Quotes

To use a double quote within a parameter, and have it considered a literal, precede it with a \ slash.

### Example

Incorrect:           Name EQ "John "Jack" Smith" (BusinessRules.Variable ...

Correct:             Name EQ "John \"Jack\" Smith" (BusinessRules.Variable ...

# Copying Business Rules

You can copy business rules and variable names between guidelines or within guidelines.

## Copying a business rule

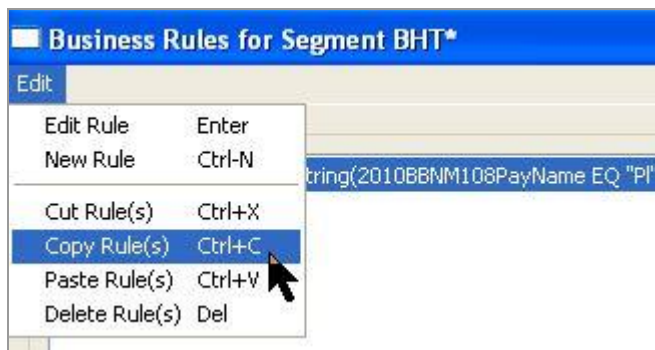
---

In Standards Editor:

1. Select the business rule in the main business rules box.



2. Use *Ctrl-c* or the Edit menu:



3. Go to the target business rules box and use *Ctrl-v* or the Edit menu to paste.

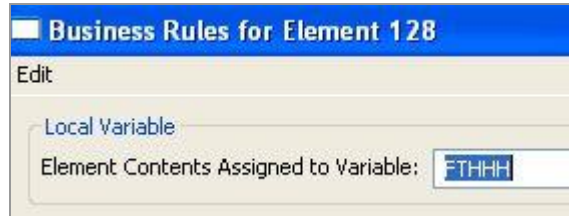


## Copying a variable

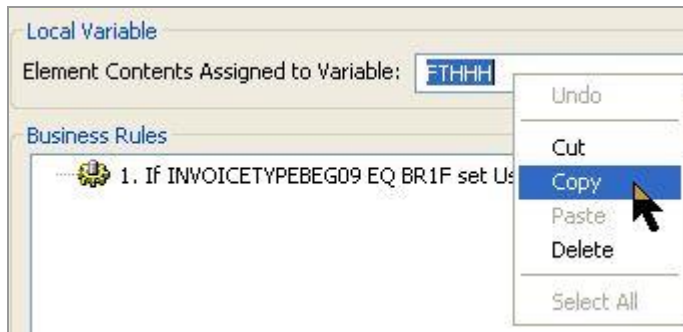
---

In Standards Editor:

1. In the main business rules box, select the entire variable name in the Local Variable text box.



2. Right-click on the text, and choose copy:



3. Go to the target location and use *Ctrl-v* or the Edit menu to paste. You can paste the variable name into any location that uses the Windows clipboard (a word processor document, spreadsheet, etc.).

## Printing Business Rules

In Standards Editor, use **File | Print | Print Rules** to get a text report of all business rules.

# Array Business Rules

---

## All validators except Analyzer

---

These rules set up and manipulate arrays, including those sent back by InvokeWebService rules.

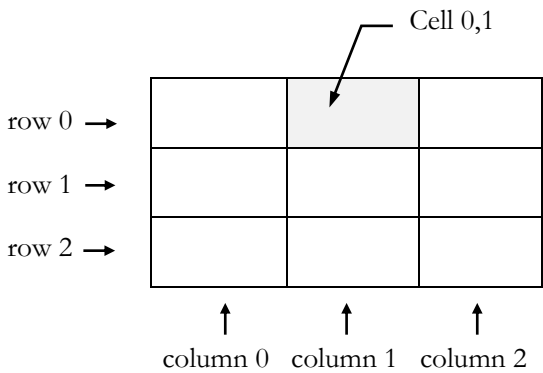
For a complete example, please see Appendix J: LookAhead and Array Extended Example on page 295.

---

### Array rows and columns

---

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



## Array Reserved Variables

**Current\_Row** The current array row index (Set or Get array rules)

**Next\_Row** The next array row index (Set or Get array rules)

**Last\_Row** The last array row in the array (Get array rules only)

Set array rules like SetArrayFromVar and Get array rules like GetArrayCurrentRowIndex maintain separate Current\_Row and Next\_Row pointers.

### Row pointer for “Set” array rules

SetArrayFromVar  
UpdateArrayFromDate

### Row pointer for “Get” array rules

CheckVarFromArray  
GetArrayCurrentRowIndex  
GetArrayNextColumnIndex  
GetArrayNextRowIndex  
GetARowFromArray  
GetVarFromArray  
UpdateArrayFromDate

## Example 1

---

### Filling the array:

1. SetArrayFromVar MyArray Current\_Row 0 “A” - Puts A in 0,0:

|          |  |  |
|----------|--|--|
| <b>A</b> |  |  |
|          |  |  |
|          |  |  |

2. SetArrayFromVar MyArray Current\_Row 1 “B” - Puts B in 0,1:

|   |          |  |
|---|----------|--|
| A | <b>B</b> |  |
|   |          |  |
|   |          |  |

3. SetArrayFromVar MyArray Next\_Row 0 “C” - Puts C in 1,0:

|          |   |  |
|----------|---|--|
| A        | B |  |
| <b>C</b> |   |  |
|          |   |  |

4. SetArrayFromVar MyArray Current\_Row 1 “D” - Puts D in 1,1:

|   |          |  |
|---|----------|--|
| A | B        |  |
| C | <b>D</b> |  |
|   |          |  |

5. SetArrayFromVar MyArray Next\_Row 0 “E” - Puts E in 2,0:

|          |   |  |
|----------|---|--|
| A        | B |  |
| C        | D |  |
| <b>E</b> |   |  |

### Getting values from array

6. GetVarFromArray MyArray Current\_Row 0 MyVar - Gets **A** from 0,0 and puts it into MyVar (Current\_Row is pointing to the first row for this Get array rule - it is not related to the Current\_Row for Set array rules):

|          |   |  |
|----------|---|--|
| <b>A</b> | B |  |
| C        | D |  |
| E        |   |  |

7. GetVarFromArray MyArray Current\_Row 1 MyVar1 - Gets **B** from 0,1 and puts it into MyVar1.

|   |          |  |
|---|----------|--|
| A | <b>B</b> |  |
| C | D        |  |
| E |          |  |

8. GetVarFromArray MyArray Next\_Row 0 MyVar2 - Gets **C** from 1,0 and puts it into MyVar2.

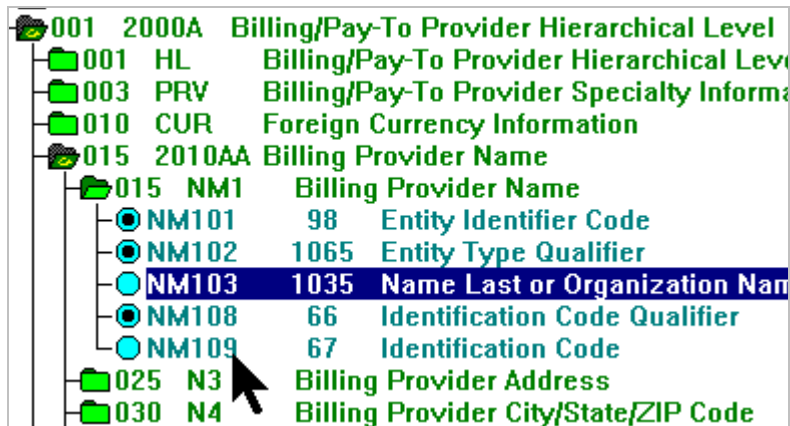
|          |   |  |
|----------|---|--|
| A        | B |  |
| <b>C</b> | D |  |
| E        |   |  |

9. GetVarFromArray MyArray Last\_Row 0 MyVar3 - Gets **E** from 1,0 and puts it into MyVar3. This will always get from the last row in the array, regardless of previous Get array rules. Last\_Row is only used with Get array rules.

|   |          |  |
|---|----------|--|
| A | <b>B</b> |  |
| C | D        |  |
| E |          |  |

## Example 2

Assume we are loading provider names (NM103) and IDs (NM109) into an array called ProvArray:



We want our array to contain a row for each billing provider:

|                    |                    |
|--------------------|--------------------|
| first NM103 value  | first NM109 value  |
| second NM103 value | second NM109 value |
| third NM103 value  | third NM109 value  |
| <i>etc.</i>        |                    |

We create the array on the ST segment:

| What Rule to Run |                |                 |
|------------------|----------------|-----------------|
| Text             | Parameter Name | Parameter Value |
|                  | arrayName      | "ProvArray"     |


Now, we load data into the array. We cannot hard-code a literal row number into our SetArrayFromVar business rule because we would continually overwrite the first row. Instead, we use Next\_Row to automatically keep the row index incrementing.


This rule on the NM103 will load up the first column (Column 0), using the next row down each time it executes:

| What Rule to Run |                |                 |
|------------------|----------------|-----------------|
| Text             | Parameter Name | Parameter Value |
|                  | arrayName      | "ProvArray"     |
|                  | rowIndex       | Next_Row        |
|                  | columnIndex    | "0"             |
|                  | resultVar      | Current_Element |

This rule on the NM109 will load up the second column (Column 1). Since we want this to be in the same row as the previous rule on the NM103, we use Current\_Row:

What Rule to Run

SetArrayFromVar 

 Parameter Name Parameter Value

|                  |                 |
|------------------|-----------------|
| arrayName        | "ProvArray"     |
| rowIndex         | Current_Row     |
| columnIndex      | "1"             |
| <b>resultVar</b> | Current_Element |

## CheckVarFromArray

Compare a value in a specific array cell to another value and perform an action if the two values do not match.

### Format of Parameters

*arrayName rowIndex columnIndex ValueA (Action)*

Where:

|                    |  |
|--------------------|--|
| <i>arrayName</i>   | Name of the array that contains the value to check. This can be a variable or a literal in double quotes.  |
| <i>rowIndex</i>    | Array row index. This can be a variable or a literal in double quotes, or one of the array reserved variables. Indexes start with 0. Please read <a href="#">Array Reserved Variables</a> on page 35 to see how current row is determined for “Get” array rules. |
| <i>columnIndex</i> | Array column index. This can be a variable or a literal in double quotes. Indexes start with 0.  |
| <i>valueA</i>      | The value to compare to the array value: a variable, literal in double quotes, <code>Current_Date</code> , or <code>Current_Element</code> .   |
| <i>(Action)</i>    | The action to be executed if the result is false.  |

### Examples

This rule checks the value in Array1BACK’s cell 0,4 (presumably returned from a web service that checked a database). If the cell does not contain “1”, a message is displayed.

This does the same thing, except the row index is the number in variable SubNum:

## ClearArray

Remove data from all or part of an array.

### Format of Parameter (three variations)

*arrayName*

*arrayName rowIndex*

*arrayName rowIndex columnIndex*

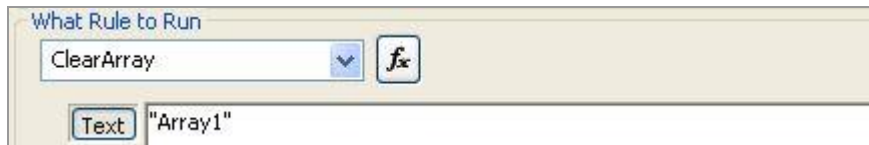
Where:

*arrayName* Name of the array where data is to be cleared. This can be a variable or a literal in double quotes.

*rowIndex* Clear data only from this row. This can be a variable or a literal in double quotes.

*columnIndex* Clear data only from this column. This can be a variable or a literal in double quotes.

**Example 1.** This clears all data from Array1:



What Rule to Run

ClearArray

Text "Array1"

**Example 2.** This clears all data from the row with index 2:



What Rule to Run

ClearArray

Text "Array1" "2"

|   | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 |   |   |   |   |   |
| 1 |   |   |   |   |   |
| 2 |   |   |   |   |   |



**Example 3.** This clears data from the cell at row index 2 and column index 4:

What Rule to Run

ClearArray

Text "Array1" "2" "4"

|   |   |   |   |   |   |
|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 |
| 0 |   |   |   |   |   |
| 1 |   |   |   |   |   |
| 2 |   |   |   |   |   |

## CreateArray

Creates an array with the name of your choice. This rule typically goes on the ST segment, where you can find it easily. However, it can go anywhere before it is used.

### Format of Parameters

*ArrayName*

Where:

*ArrayName*      Name of the array you are creating. This can be a variable or a literal in double quotes.

### Examples

This rule creates an array named Array1.

What Rule to Run

CreateArray

Text "Array1"

This rule creates an array named whatever is in the variable Array1.

What Rule to Run

CreateArray

Text Array1

## DumpArray

Displays all values in an array.

### Format of Parameters

*ArrayName*

Where:

*ArrayName*                      Name of the array containing the data you want to see. This can be a variable or a literal in double quotes.

### Examples

This rule displays the contents of Array1:



In this example output, each row contains 5 values:

```
EMSG      9**FS Debug DLL Invoking InvokeWebService :   Row 0 = [2][9999][888888][aaaa][1]
EMSG      9**FS Debug DLL Invoking InvokeWebService :   Row 1 = [2][22222][NA][bbbbbb][1]
```

## GetArrayCurrentRowIndex

Reports the index of the current row in the array.

As an alternative, use the `Current_Row` reserved variable.

### Format of Parameters

*arrayName indexVar*

Where:

*ArrayName*                      Name of the array containing the data you want to see. This can be a variable or a literal in double quotes.

*indexVar*                      A variable in which to return the current row's index. The current row is the last row that you inserted to. Please read [Array Reserved Variables](#) on page 35 to see how current row is determined for "Get" array rules.

See also the reserved variable `Current_Row` on page 35.

## GetArrayNextColumnIndex

Report the index of the column after the last column.

### Format of Parameters

*arrayName* *rowIndex* *indexVar*

Where:

*arrayName* Name of the array. This can be a variable or a literal in double quotes.

*rowIndex* The next column index of this row. This can be a variable or a literal in double quotes, or one of the array reserved variables. Indexes start with 0. Please read Array Reserved Variables on page [35](#) to see how current row is determined for “Get” array rules.

*indexVar* A variable in which to return the index.

### Example



What Rule to Run

GetArrayNextColumnIndex

Text "Array1" NextCol

|     |     |    |    |
|-----|-----|----|----|
| AAA | 55  |    | NO |
| BBB | 123 | A5 |    |
| XXX | 432 |    |    |

## GetArrayNextRowIndex

Report the index of the row after the last row.

### Format of Parameters

*arrayName index*

Where:

|                  |   |
|------------------|---|
| <i>arrayName</i> | Name of the array. This can be a variable or a literal in double quotes.  |
| <i>indexVar</i>  | A variable in which to return the index. Please read Array Reserved Variables on page <a href="#">35</a> to see how the next row is determined for “Get” array rules, and to read about the reserved variable Next_Row. |

### Example

What Rule to Run

GetArrayNextRowIndex

Text "Array1" NextRow

## GetARowFromArray

Populates one or more variables from a row in the array.

### Format of Parameters

*arrayName rowIndex varA varB varC ..*

Where:

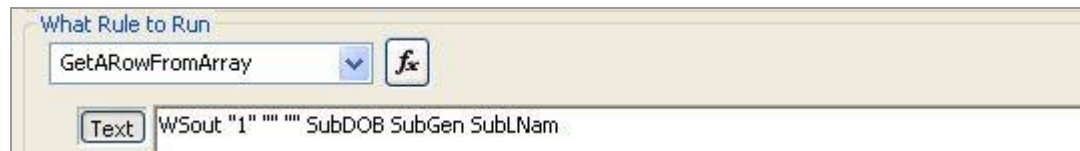
|                  |  |
|------------------|--|
| <i>arrayName</i> | Name of the array. This can be a variable or a literal in double quotes.   |
| <i>rowIndex</i>  | The row where the values are located. This can be a variable, a literal in double quotes, or one of the array reserved variables. Please read Array Reserved Variables on page 35 to see how to specify a row for “Get” array rules. |
| <i>varA</i>      | Variable to hold the contents of the first cell in the row. Use empty double quotes to skip any cell.  |
| <i>varB</i>      | Variable to hold the contents of the second cell in the row.   |
| <i>var C</i>     | Continue with variable names until you have all the data that you need from the row.   |

### Examples

Assume that array WSout contains this:

|         |   |           |                  |            |           |            |
|---------|---|-----------|------------------|------------|-----------|------------|
|         | 1 | SUB FNAME | SUB. DATEOFBIRTH | SUB GENDER | SUB LNAME | VALID DATE |
| row 1 → | 1 | PATRICK   | 19730203         | M          | WILSON    | 20010101   |
|         | 1 | KATHERINE | 19741125         | F          | WILSON    | 20010101   |

This business rule collects values from row 1:



With the array shown above:

- Columns 0 and 1 would be skipped.
- SubDOB would contain 19740203
- SubGen would contain M
- SubLnam would contain WILSON

This rule is similar and collects values from the current row:

The screenshot shows a configuration window titled "What Rule to Run". It features a dropdown menu with "GetARowFromArray" selected and a function icon (fx). Below this is a "Text" field containing the string: "WSout Current\_Row "" "" SubDOB SubGen SubLNam".

This rule is similar, and uses the row number stored in CurrentSearch\_Row (from a SearchVarsInArray or SearchConditionsInArray rule).

The screenshot shows a configuration window titled "What Rule to Run". It features a dropdown menu with "GetARowFromArray" selected and a function icon (fx). Below this is a "Text" field containing the string: "WSout CurrentSearch\_Row "" "" SubDOB SubGen SubLNam".

## GetVarFromArray

Populates a variable from a cell in an array.

### Format of Parameters

*arrayName rowIndex columnIndex varName*

Where:

|                    |  |
|--------------------|--|
| <i>arrayName</i>   | Name of the array that contains the value. This can be a variable or a literal in double quotes.   |
| <i>rowIndex</i>    | The row where the cell is located. This can be a variable or a literal in double quotes, or one of the array reserved variables. Indexes start with 0. Please read Array Reserved Variables on page <a href="#">35</a> to see how current row is determined for “Get” array rules. |
| <i>columnIndex</i> | The column where the cell is located. This can be a variable or a literal in double quotes.  |
| <i>varName</i>     | Variable that is to receive the value that was found in the array cell. This can be a variable or a literal in double quotes.  |

### Examples

These two rules show how you might check an array returned from InvokeWebService and then use a value it contains. In this example, we are assuming that the web service returns a value of 1 if a certain value is not in a database.

This rule copies the value in Array1BACK’s row 0 column 4 into the variable NM109BACK:

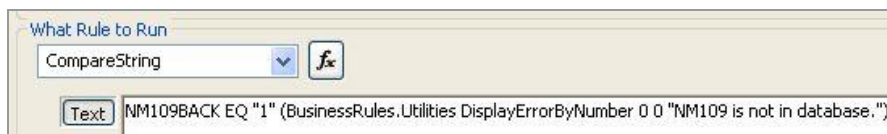


What Rule to Run

GetVarFromArray

Text "Array1Back" "0" "4" NM109BACK

This rule then checks this variable and displays a message if the value is 1.



What Rule to Run

CompareString

Text NM109BACK EQ "1" (BusinessRules.Utilities DisplayErrorByNumber 0 0 "NM109 is not in database.")

## SearchVarsInArray

Search an array for a row that contains certain values, and perform an action if the search fails.

Results will be put in a variable **CurrentSearch\_Row**. It will be set to the index of the row where the values were found, or to -1 if there is no match.

### Format of Parameters

*arrayName valueA valueB ... (FailedAction)*

Where:

*arrayName* Name of the array to search. This can be a variable or a literal in double quotes.

*valueA valueB ...* A list of values, each separated by a space. These can be any combination of variables, literals in double quotes, Current\_Date or Current\_Element. Please see the example below.

The list can contain one or more values. The first value will be located in the first column of the array, the second value will be located in the second column of the array, etc. To skip a column, use empty double quotes. See the example below for details. All values must be found in the same row for it to be considered a match.

*(Action)* The action to be executed no row contains all the values.

### Example

Assume that we want to search an array for a subscriber with a certain date of birth, gender, and last name. We have the array shown below (shading shows which columns are significant in this particular search).

Assume that the list of values in the parameter is: "1" "" DMG02 Current\_Element SLnam

The list maps to the columns like this:

|                | "1" | ""        | DMG02            | Current_Element | SLnam     |            |
|----------------|-----|-----------|------------------|-----------------|-----------|------------|
|                | ↓   | ↓         | ↓                | ↓               | ↓         |            |
|                | 1   | SUB FNAME | SUB. DATEOFBIRTH | SUB GENDER      | SUB LNAME | VALID DATE |
|                | 1   | PATRICK   | 19740203         | M               | WILSON    | 20010101   |
| matching row → | 1   | KATHERINE | 19741125         | F               | WILSON    | 20010101   |
|                | 1   | RITA      | 19221120         | F               | O'NEILL   | 20010101   |

Assume:

- DMG02 = 19741125
- Current\_Element = F




- SLnam = WILSON

The array is searched like this:

- All rows contain 1 in the first column, so all rows match so far.
- The second column is skipped because of the empty "".
- The third column contains one match to the contents of the variable DMG02 (19741125). Only one row matches now.
- In the fourth column of the row that still matches, the F matches the contents of Current\_Element.
- In the fifth column of the row that still matches, WILSON matches the contents of SLnam. Since we have a row that matches all values in the list, the *Action* in the business rule does not execute.

The rule looks like this:


What Rule to Run

SearchVarsInArray 

Text "Array1BACK "1" "" DMG02 Current\_Element SLnam (BusinessRules.Utilities.DisplayErrorByNumber 0 0 "Subscriber's last name+DOB+gender not found in Subscriber database.")

You can check CurrentSearch\_Row for results. In this example, if the search succeeded, we get the value from column two in the matched row and put the value into a variable SubDOB.

What Rule to Run

CompareString 

Text CurrentSearch\_Row NE "-1" (BusinessRules.Array.GetVarFromArray W5out CurrentSearch\_Row 2 SubDOB)

# SearchConditionsInArray

Search an array for a row that contains certain values, partial values, or combinations of values and perform an action if the search fails.

Results will be put in a variable **CurrentSearch\_Row**. It will be set to the index of the row where the conditions were found, or to -1 if there is no match.

## Format of Parameters

*arrayNam #matches Criteria Criteria ... (FailedAction)*

Where:

|                       |   |
|-----------------------|---|
| <i>arrayName</i>      | Name of the array to search. This can be a variable or a literal in double quotes.  |
| <i>#matches</i>       | Number of columns that must match.  |
| <i>Criteria ...</i>   | One or more sets of criteria, each separated by a space. Please see <b>matchCriteria format</b> below. The first criteria applies to the first column of the array, the second criteria applies to the second column of the array, etc. To skip a column, use empty double quotes. See the example below. |
| <i>(FailedAction)</i> | The action to be executed if no row matches the criteria.   |

## Criteria

---

This can have several formats:

- `""` Skip a column.
- Literal in quotes. Example: `"SMITH"`.
- Variable. Example: `SubscrLName`.
- Comparison The literal text COMP followed by a comparison of part or all of the column contents with a value from the EDI. See below.

Comparison format:

*COMP (EDIVar, EDIstart Arraystart, lengthToCompare, requirement)*

Where:

|                        |   |
|------------------------|---|
| <i>COMP</i>            | Literal text indicating a comparison.   |
| <i>EDIVar</i>          | Value from EDI to compare.  |
| <i>EDIstart</i>        | Position in EDI value to start comparison.  |
| <i>Arraystart</i>      | Position in array value to start comparison.  |
| <i>lengthToCompare</i> | Number of characters to compare.  |
| <i>requirement</i>     | <b>M</b> if values must match.<br><b>O</b> if they do not have to match as long as the #matches is met. |

### Example 1

---

Assume that we want to search an array for a subscriber who was born in November. If none are found, we want to display the message “No subscribers in this transaction were born in November.”

We are searching an array named Array1Back shown below (shading shows which column is significant in this particular search; the underlined parts of the values are being compared to “11”).

|         |        |           |                   |          |
|---------|--------|-----------|-------------------|----------|
| ANDREWS | ALAN   | 111222333 | 2001 <u>02</u> 12 | 20040212 |
| BROWN   | BENITA | 222333444 | 2001 <u>02</u> 12 | 20080212 |

The business rule would be:

What Rule to Run

SearchConditionsInArray

Array1Back 1 "" "" "" COMP("11",1,5,2,M) (BusinessRules.Utilities.DisplayErrorByNumber 0 0 "No subscribers in this transaction were born in November")

Where:

|                   |  |
|-------------------|--|
| <i>Array1Back</i> | Name of array to search.   |
| <i>1</i>          | Number of matches required.  |
| <i>"" "" ""</i>   | The three sets of empty double quotes mean to skip the first three columns in the array. |

This brings us to the fourth column, where we have the comparison *COMP ("11", 1, 5, 2, M)*

|             |  |
|-------------|--|
| <i>COMP</i> | Literal text meaning there is a comparison.      |
| <i>"11"</i> | Literal value to compare to value in array.      |
| <i>1</i>    | Start comparing at position 1 in the value “11”. |
| <i>5</i>    | Start comparing at position 5 in the array value |

2                                      Compare 2 characters.

M                                      Must match.

BusinessRules.Utilities...                                      Action to take if no match is found.

In this example, the comparison will fail and the message will be displayed.

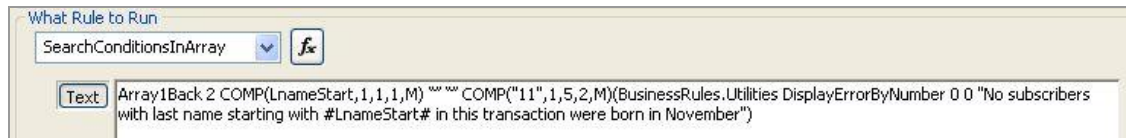
## Example 2

---

Assume that we want to search the same array for a subscriber:

- Whose last name starts with the letter in variable LnameStart
- AND, whose birth month was November

If the array does not contain a row where both conditions match, we display a message.



Where:

Array1Back                      Name of array to search.

2                                      Number of matches required.

COMP (LnameStart,1,1,1,M)

In the first column, we start comparing the first position in variable LnameStart, the first position in the array, one character , and they must match.

“” “”                                      We skip the next two columns in the array.

COMP ("11",1,5,2,M) This brings us to the fourth column, where we have the same comparison that we did in Example 1.

In this example, the comparison will fail because no subscribers were born in November, and both conditions are mandatory for success. The message will be displayed.

## Example 3

---

Assume that we want to search the same array for a subscriber:

- Whose last name starts with the letter in variable LnameStart
- OR, whose birth month was November

If the array does not contain a row where at least one condition matches, we display a message.

The pertinent changes are underlined and in bold:

```
Array1Back 1 COMP(LnameStart,1,1,1,O) "" ""  
COMP("11",1,5,2,O) (BusinessRules.Utilities DisplayErrorByNumber 0 0  
"No subscribers had a last name starting with #LnameStart# in this  
transaction and none were born in November")
```

The pertinent parts include;

The first 1 Means that only one match is needed: either the COMP in the first column or in the fourth.

The first O (Capital O for Optional) means that the first column's comparison is not required for success.

The second O Means that the fourth column's comparison is not required, either.

However, at least one of them is required due to the 1 immediately after the array name.

#### Example 4

You can check CurrentSearch\_Row for results. In this example, if the search succeeded, we get the value from column two in the matched row and put the value into a variable IDnum.

## SetArrayFromVar

Populates a cell in an array.

### Format of Parameters

*arrayName rowIndex columnIndex value*

Where:

|                    |   |
|--------------------|---|
| <i>arrayName</i>   | Name of the array you are populating. This can be a variable or a literal in double quotes.   |
| <i>rowIndex</i>    | The row where the cell is located. This can be a variable, a literal in double quotes, or one of the array reserved variables. Please read <a href="#">Array Reserved Variables</a> on page 35 to see how to specify a row for “Set” array rules. |
| <i>columnIndex</i> | The column where the cell is located. This can be a variable or a literal in double quotes.   |
| <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> .  |

### Examples

This rule puts the value 0 into cell 0,5:

What Rule to Run

SetArrayFromVar

Text "Array1Back" "0" "5" "0"

This rule puts the value in the variable SubNM109 into cell 0,5 in Array1:

What Rule to Run

SetArrayFromVar

Text "Array1Back" "0" "5" SubNM109

This rule puts the value in variable SubNM109 into an array named by the contents of a variable called Array1Back. It will go in row 0. The column index is in variable SubNum.

|                  |   |
|------------------|---|
| What Rule to Run |   |
| SetArrayFromVar  |  |
| Text             | Array1 SubNM109 "0" Subnum  |

|                  |   |
|------------------|---|
| What Rule to Run |   |
| SetArrayFromVar  |  |
| Text             | Array1Back "0" SubNum SubNM109  |

# UpdateArrayFromDate

Compares two dates (one in the array) and updates the one in the array if the result is true.

## Format of Parameters

*arrayName rowIndex columnIndex operand dateV DateFormat*

Where:

|                    |  |
|--------------------|--|
| <i>arrayName</i>   | Name of the array that contains the date. This can be a variable or a literal in double quotes.  |
| <i>rowIndex</i>    | The array row where the date is located. This can be a variable, a literal in double quotes, or one of the array reserved variables. Please read Array Reserved Variables on page 35 to see how current row is determined for “Set” array rules. |
| <i>columnIndex</i> | The array column where the date is located. This can be a variable or a literal in double quotes.  |
| <i>operand</i>     | <b>EQ, NE, GT, GE, LT, or LE.</b>  |
| <i>dateV</i>       | Location of the date in the data: a variable, literal in double quotes, Current_Date, or Current_Element.  |
| <i>DateFormat</i>  | D8 (for YYYYMMDD)<br>D6 (for YYMMDD)<br><br>The dates being compared must have the same format. This can be a literal in quotes or a variable.   |

## Example

If the date in the current element is less than the date in cell 0,2 of Array1, replace the array value with the current element’s value. Format of both dates is YYYYMMDD.

What Rule to Run

UpdateArrayFromDate



Text

"Array1" "0" "2" LT Current\_Element D8



# Correct Coding Initiatives (CCI) Business Rules

---

## Instream and HIPAA Validator Desktop

### HIPAA only

---

These rules enforce:

- Correct Coding Initiatives (CCI) for Part B Medicare Carriers from the Centers for Medicare and Medicaid Services (CMS).
- National Correct Coding Initiative (NCCI) edits for Hospital Outpatient Prospective Payment System (OPPS).

This initiative consists of a large number of CPT pairs, where the second CPT code:

- Will not be paid if it occurs in the same claim and on the same service date as the first CPT code.
- Or, will not be paid if it occurs in the same claim and on the same service date unless a modifier exists for the second code.

TIBCO Foresight has already added the rules for CCI checking in the PDSA837P and B41A837P guidelines. You can turn on and off checking with a setting in the APF file (described below).

You can add your own rules to your 837I guidelines to check for compliance to NCCI edits for OPPS. TIBCO Foresight-supplied guidelines do not have these rules pre-built. See the sections below on CCIInit, CCICollect, and CCIAalyze.

---

### Turning on CCI checking during validation

---

To validate with CCI or NCCI checking:

- Use a guideline that has CCI or NCCI business rules set up.
- Set `CheckCCIEdits=1` in the APF file being used for validation.

The three CCI rules are:

- CCIInit
- CCICollect
- CCIAalyze

## CCIIInit

---

### Instream and HIPAA Validator Desktop HIPAA only

---

Clears the collection of data to prepare for another claim.

**CCI Part B Medicare:** This rule is already attached to the CLM segment for PDSA837P and B41A837P guidelines.

The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "CCIInit" selected, a blue downward arrow, and a button with a function symbol (fx). Below this is a "Text" label and an empty text input field.

**NCCI for Outpatient 837I:** On the CLM segment of your own 837I guideline, use the O parameter for NCCI for outpatient tables.

The screenshot shows the same "What Rule to Run" dialog box. The dropdown menu still shows "CCIInit". The "Text" input field now contains the letter "O".

## CCICollect

---

### Instream and HIPAA Validator Desktop HIPAA only

---

Collects information needed for the analysis. This rule goes on these segments that have data to collect:

| Segment  | Collects ...                         |
|--|--------------------------------------|
| 2400 LX  | Line count for use in error messages |
| 2400 SV1, SV2, or SV3  | Procedure codes and modifiers        |
| 2400 DTP for Service Date or<br>2300 DTP for Statement Dates | Date of service                      |

#### Example

The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "CCICollect" selected, a blue downward arrow, and a button with a function symbol (fx). Below this is a "Text" label and an empty text input field.

## CCIAalyze

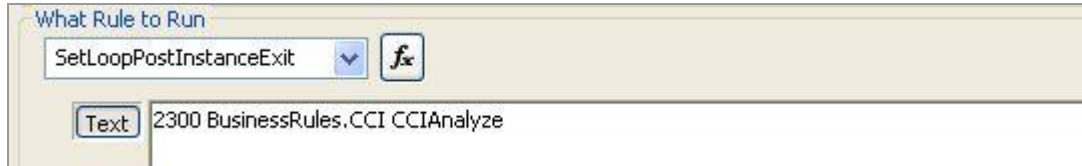
---

### Instream and HIPAA Validator Desktop HIPAA only


---

Analyze the data in the CCI and NCCI collection against the CCI and NCCI tables, and report errors. The CCIAalyze rule is usually set up on the ST segment to run at the end of each claim loop.

This example rule runs at the end of any 2300 loop.



What Rule to Run

SetLoopPostInstanceExit ▼ 

Text 2300 BusinessRules.CCI CCIAalyze

Only run the CCIAalyze rule for NCCI checking if the CLM05-01 is an outpatient code.

CCI/NCCI error messages are in the range 31031 to 31070 and are listed in FSBRErrs.txt in the Bin directory for Instream and HIPAA Validator Desktop.

# Code Lookup Business Rules

## FindCode

---

### Instream and HIPAA Validator Desktop

---

Issues an error message if the code is not valid. It looks up the code:

- First, it checks code tables your company has set up (see Appendix C: Code Tables on page 261).
- Then, for HIPAA guidelines, it checks for the presence of the code in the TIBCO Foresight-distributed external HIPAA code tables.

#### Format of Parameters

*CodeTable* *CodeValue* (*ifNotFoundAction*) (*ifFoundAction*)

Where:

|                             |  |
|-----------------------------|--|
| <i>CodeTable</i>            | Name of the external code table.   |
| <i>CodeValue</i>            | Value to be checked. If omitted, the value in the current element is assumed.  |
| ( <i>ifNotFoundAction</i> ) | Action to be taken if the value is <b>not</b> found in the table. The parentheses must be included. To take no action, use (BusinessRules.Utilities DoNothing) |
| ( <i>ifFoundAction</i> )    | Optional. Action to be taken if the value <b>is</b> found in the table. The parentheses must be included.  |

#### Examples

What Rule to Run

FindCode

Text CPT4

What Rule to Run

FindCode

Text CPT4 VarA (BusinessRules.Utilities DoNothing) (BusinessRules.Utilities DisplayErrorByNumber32201)

Example parameters:

CPT4                      **CPT4** is the code table. Since *CodeValue* and *ifNotFoundAction* are omitted, this rule will use the code value in the current element and issue a generic error message.

CPT4   A                      **A** is a variable.  
Since this has a *CodeValue* of A, see if the value in variable A is in table CPT4. If the value is not found in table CPT4, issue a generic error message.

CPT4   "A"                      **"A"** is a literal.  
Since this has a *CodeValue* of "A", check for the presence of code A in table CPT4. If A is not found, issue a generic error message.

CPT4   Current\_Element   (BusinessRules.Utilities   DisplayErrorByNumber   32201)

                                 If the value in the current element is not found in table CPT4, display the text in error number 32201.

                                 We must include the second parameter, *CodeValue*, since we are using the third parameter, (*ifNotFoundAction*).

CPT4   Current\_Element   (BusinessRules.Utilities   DoNothing)  
                                 (BusinessRules.Utilities   DisplayErrorByNumber   32202)

                                 If the value in the current element is not found in table CPT4, do nothing.

                                 If the value is found in table CPT4, display the text in error number 32202.

                                 We must include all previous parameters, since we are using the last parameter, (*ifFoundAction*).

## FindCodeWithDate

---

### Instream and HIPAA Validator Desktop

---

Issues an error message if the code is not valid on a specific date. It looks up the code:

- First, it checks code tables your company has set up (see Appendix C: Code Tables on page 261).
- Then, for HIPAA guidelines, it checks for the presence of the code in the TIBCO Foresight-distributed external HIPAA code tables.

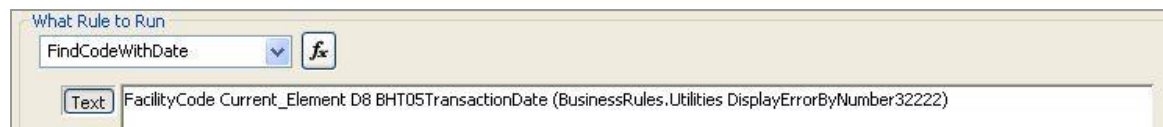
#### Format of Parameters

*CodeTable CodeValue DateFormat DateToCheck (ifNotFoundAction) (ifFoundAction)*

Where:

|                           |  |
|---------------------------|--|
| <i>CodeTable</i>          | Name of the external code table.   |
| <i>CodeValue</i>          | Value to be checked. It can be a literal in double quotes, a variable, or <code>Current_Element</code> .   |
| <i>DateFormat</i>         | The format of the date, which must match the format of <code>DateToCheck</code> . This can be a variable or a literal. If the date can be in more than one format, assign a variable to the format element and then call <code>FindCodeWithDate</code> . |
| <i>DateToCheck</i>        | The date: a variable that has been assigned to a date element, a literal in double quotes (such as "20030625"), or <code>Current_Element</code> if it is a date.   |
| <i>(ifNotFoundAction)</i> | The action to be taken if the value is <b>not</b> effective on that date. If omitted, a generic error message appears. The parentheses must be included. To take no action, use <code>(BusinessRules.Utilities DoNothing)</code>                         |
| <i>(ifFoundAction)</i>    | Optional. Action to be taken if the value <b>is</b> effective on that date. The parentheses must be included.  |

**Example 1.** This rule checks to see if the facility code is valid for the transaction date. If not, it displays error 32222.



Where:

|                              |  |
|------------------------------|--|
| <code>FacilityCode</code>    | The table.                                       |
| <code>Current_Element</code> | Value of the element where the rule is attached. |

D8                                      Date format for BHT04TransactionDate.

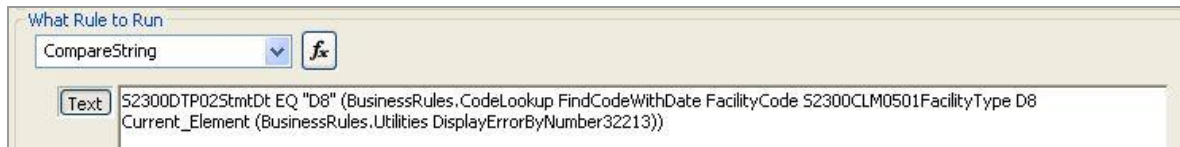
BHT04TransactionDate

Variable assigned to the transaction date element.

(BusinessRules.Utilities DisplayErrorByNumber 32222)

Specifies that error message 32222 should be displayed if the code is not valid for the transaction date.

**Example 2.** This rule on the statement date element checks to see if the date format is D8. If so, it checks the facility code to see if it is valid on that date.



Where:

CompareString checks the element that has the BusinessRules.Variable S2300DTP02StmntDt assigned to see if it contains D8.

If so, it executes the FindCodeWithDate function inside the outer parentheses:

FacilityCode            The table.

S2300CLM0501FacilityType

Variable assigned to the facility type element that is being checked.

D8                                      Date format.

Current\_Element    Value of the element where the rule is attached - the statement date.

(BusinessRules.Utilities DisplayErrorByNumber 32213)

Specifies that error 32213 should be displayed if the code is not valid for the transaction date.

## FindUserCode

---

### Instream and HIPAA Validator Desktop

---

Like FindCode (see page [61](#)) but checks your own external code table only. It does not check TIBCO Foresight's table at all. See Appendix C: Code Tables on page [261](#).

## FindUserCodeWithDate

---

### Instream and HIPAA Validator Desktop

---

Like FindCodeWithDate (see page [63](#)) but checks your own external code table only. It does not check TIBCO Foresight's table at all. See Appendix C: Code Tables on page [261](#).



## ValidateZipState

---

### Instream and HIPAA Validator Desktop HIPAA only

---

Checks the zip code and issues an error message if it is not valid for the state code.

If the zip code includes the four-digit extension (as in 43017-1111), only the first five digits will be validated against the state code.

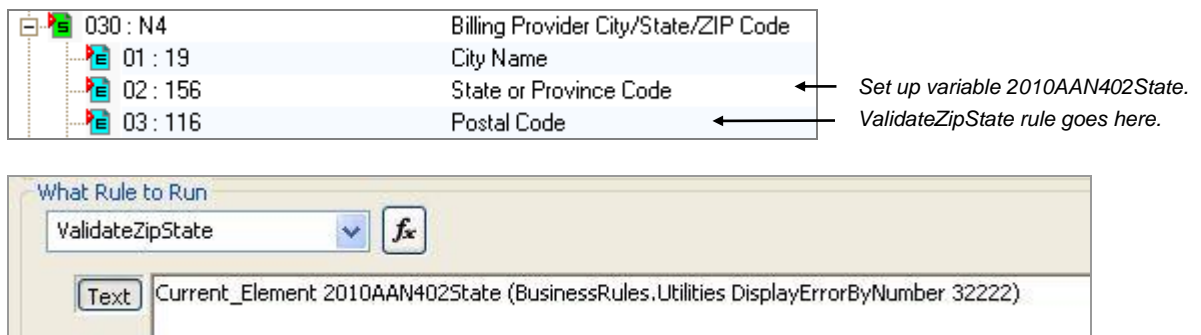
#### Format of Parameters

*ZipCode StateCode ifNotMatchAction*

Where:

|                           |  |
|---------------------------|--|
| <i>ZipCode</i>            | Zip code element. It can be a variable or <code>Current_Element</code> .   |
| <i>StateCode</i>          | <code>Current_Element</code> , literal in double quotes, or a variable pointing to the state element.  |
| <i>(ifNotMatchAction)</i> | Optional. Action to be taken if the zip code is not valid in the state. If omitted, the following message will be displayed: “ <i>ZipCode</i> ” is not valid for the State “ <i>StateCode</i> .” |

**Example.** This rule is applied to the N403 to see if the zip code it contains is valid for the state contained in the N402.



Where:

|   |  |
|---|--|
| <code>Current_Element</code>                                      | Value of the zip code element where the rule is attached.                                  |
| <code>2010AAN402State</code>                                      | Variable assigned to the state element.  |
| <code>(BusinessRules.Utilities DisplayErrorByNumber 32222)</code> | Specifies that error 32222 should be displayed if the zip code is not valid for the state. |

# Core3 (Phase III CORE) Business Rules

---

## Instream and HIPAA Validator Desktop

### HIPAA only

---

The Phase III CORE 360 Uniform Use of Claim Adjustment Reason Codes and Remittance Advice Remark Codes (835) Rule checks for certain valid combinations of Claim Adjustment Group Codes (CAGC) with Claim Adjustment Reason Codes (CARC). This rule also checks for valid combination of Claim Adjustment Reason Codes (CARC) with the Remittance Advise Remark Codes (RARC).

In general:

- A CARC can be used only with certain CAGCs.
- A CARC/CAGC pair may have a list of valid RARC(s), at least one of which must be used.

---

### Turning on Phase III CORE checking during validation

---

To validate with Phase III CORE checking:

- Use a guideline that has Core3 business rules set up. PDSA5010835.std and 5010-835.std guidelines incorporate the Phase III CORE 360 Rule.
- Set `CheckCore3Edits=1` in the APF file being used for validation.

The Core3 rules are:

- Core3CheckCARC
- Core3CheckMissingRARC
- Core3Init
- Core3TrackRARC

At this time, Core3 rules are for internal TIBCO Foresight use only.

# Custom Record Business Rules

## Instream and HIPAA Validator Desktop

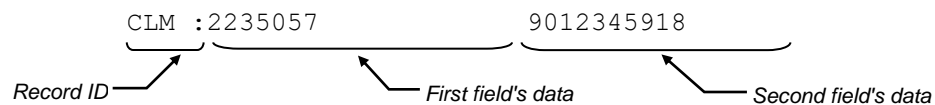
A custom record places the contents of actual data in the validation output stream. The name and contents of the record are customizable. You might wish to display a patient ID number or claim number in the output, for example.

You can set up custom records to be generated each time a message is placed in the output stream, or on demand.

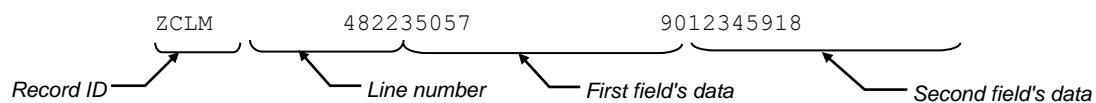
The layout of a custom record is:

| Field               | Notes   |
|---------------------|---|
| Record ID           | 1-4 alphanumeric characters. Avoid starting your Record ID with a <b>Z</b> , <b>00</b> , <b>S0</b> , or <b>P0</b> , or with any ID listed under Custom Record IDs to Avoid on page 69.<br><i>Instream</i> : Occupies positions 1-5, including a Z that is automatically added to the beginning of the record.<br><i>HIPAA Validator Desktop</i> : Occupies positions 1-5. Includes a trailing colon. No Z added to beginning. |
| Line #              | <i>Instream only</i> . Automatically generated during validation: the number of the segment line in the input file that caused the message to be generated.<br>Occupies positions 6-15.   |
| Field 1 Data        | Contents of the specified BusinessRules.Variable name.<br><i>Instream</i> : Starts at position 16.<br><i>HIPAA Validator Desktop</i> : Starts at position 6.  |
| Field <i>n</i> Data | Immediately after previous field  |

HIPAA Validator Desktop example custom record with two values:



Instream example - same custom record



Three functions let you create, output, and remove custom records that contain data of your choosing:

- DefineCustomRec
- OutputCustomRec
- RemoveCustomRec

To set up a custom record:

1. Assign variables for the data that will be output. Use SetVar, AddVar, or other BusinessRules.Variable (see page 240). You can assign these before or after defining the record. Local variables will not work in custom records.
2. Define the record layout using DefineCustomRec (see page 70).
3. Output the record using OutputCustomRec (see page 72). This is not necessary for Automatic records (see page 70).

---

## Setting up variables for use by custom records

The example in the next two sections creates a record that contains the submitter number and claim number, like this:

|                                |            |                              |
|--------------------------------|------------|------------------------------|
| CLM :2235057                   | 9012345918 | (HIPAA Validator<br>Desktop) |
| ZCLM <i>linenumber</i> 2235057 | 9012345918 | (Instream)                   |

We need to define BusinessRules.Variable for these two elements. To do this, we can use SetVar to assign:

- Variable 1000ANM109SubmitterNum to the submitter name.
- Variable 2300CLM01ClaimNum to the claim number.

---

## Custom Record IDs to Avoid

In general, avoid record names starting with X or Z unless you are using them with TIBCO Foresight® Transaction Insight®.

Do not use these IDs for your own custom records.

|      |      |      |      |      |         |         |
|------|------|------|------|------|---------|---------|
| BGNS | GSSG | ISA1 | SBA2 | PTHD | STST    | XN00-99 |
| CLSP | HCPN | ISNM | SBEL | RMRA | TRSE    | XS00-00 |
| CLSS | HDDT | ISPN | SBNM | RMRB | TS2     |         |
| DPA1 | HLDP | ISST | SBST | RQAA | TS3     |         |
| DPA2 | HLIR | MIA  | SBSV | RQNM | UMA1    |         |
| DPEL | HLIS | P009 | SBTN | RQST | UMA2    |         |
| DPNM | HLRQ | P010 | SPAA | S009 | UMNM    |         |
| DPST | HLSB | P011 | SPNM | S010 | UMST    |         |
| DPSV | HLSP | SREF | SPST | S011 | XTID    |         |
| DPTN | HLSS | SSAA | SSTC | SSST | XTIA    |         |
| DSTC | HLUM | S012 | P012 | SSTN | XD00-99 |         |
| DSVC | IRNM | S020 | P020 | SSVC | XF00-00 |         |
| DTRN | IRST | SBA1 | PRST | STRN | XM00-99 |         |

## DefineCustomRec

---

### Instream and HIPAA Validator Desktop

---

Defines the layout of a custom record. This rule is normally attached to the ST segment.

#### Format of Parameters

*ID Flag VarInfo*

Where:

|                |  |
|----------------|--|
| <i>ID</i>      | An ID for the record: 1 to 4 alphanumeric characters. (Instream output adds a <b>Z</b> to the beginning of the ID.)  |
| <i>Flag</i>    | <b>M</b> , <b>D</b> , or <b>A</b> .<br><br>M (Manual) <b>HIPAA Validator Desktop and Instream</b> . The record is output when an OutputCustomRecord rule calls it. No accompanying detail record is output.<br><br>D (Manual with Detail). <b>HIPAA Validator Desktop and Instream</b> . The custom record is output when an OutputCustomRecord rule calls it. A detail record is also output.<br><br>A (Automatic) <b>Instream only</b> . All automatic records output whenever an error is encountered. No OutputCustomRecord rule is needed. This might be useful if you want to show the billing provider name, the claim number, etc., for each error. A rule with a flag of <b>A</b> is ignored by HIPAA Validator Desktop.  |
| <i>VarInfo</i> | Variables to be included in the output record. This is a list of variable name and width pairs in the format <i>variable/length</i> . The variable name and the length are separated by a slash. Commas separate multiple <i>variable/width</i> pairs.<br><br>The variables have been set up with business rules like SetVar or AddVar before being output. Undefined variables appear as blanks in the record.<br><br>The length need not be the same as the length of the data in the variable. It can be preceded with <b>L</b> to left-justify (the default) or <b>R</b> to right-justify the data.<br><br>Example: VarOne/12, VarTwo/R5, VarThree/L10<br><br>This includes three variables: VarOne is left-justified in a field 12 characters wide, VarTwo is right-justified in a field 5 characters wide, and VarThree is left-justified in a field 10 characters wide. The L is actually not needed in the last pair, since it is the default. |

**Example.** This rule displays a record that contains the submitter number and claim number as described on page 69.

What Rule to Run

DefineCustomRec

Text CLM M 2300CLM01ClaimNum/20,1000ANM1095SubmitterNum/10

Where:

|                           |   |
|---------------------------|---|
| CLM                       | ID of the custom record being defined.  |
| M                         | Flag indicating Manual output (meaning the record is output when called by a OutputCustomRec rule in either HIPAA Validator Desktop or Instream). |
| 2300CLM01ClaimNum         | Variable containing data that is to be included in the record (in this case, the claim number).   |
| /                         | Separator between variable and its length.  |
| 20                        | Include the first 20 characters of the data from the variable.  |
| ,                         | Comma separates this <i>variable/length</i> pair from the next.   |
| 1000ANM109SubmitterNum/10 | A second <i>variable/length</i> pair, this one showing the first 10 characters of the contents of the Submitter NM109.                            |

When output with OutputCustomRec, this message might look like:

```
CLM :2235057          9012345918      (HIPAA Validator Desktop)
or
ZCLM      482235057          9012345918      (Instream)
```

With Instream, a Z appears at the beginning of the record to flag it as a custom record. The line number field is automatically added to the Instream record (in this example, eight spaces plus 48).

## Fine-tuning the appearance of the custom record

You can add literal text to your message by setting up variables containing the literal text. Then use these variables in the DefineCustomRec.

## OutputCustomRec

---

### Instream and HIPAA Validator Desktop

---

Places custom records in the validation output.

#### Format of Parameters

*ID ID ID ...*

Where:

*ID* ID of the record: 1 to 4 alphanumeric characters. This ID was set up with a DefineCustomRec rule. If the ID is omitted, all custom records are output.  
Optional. Multiple record IDs can be included. Separate each with a space. The custom records can include those defined with flags of M, A, or D.

**Example.** This rule displays the custom record **CLM**, defined in the example under DefineCustomRec (see page 70). We can attach this rule to the CLM01, after the SetVar for the variable used in the record definition.



The screenshot shows a dialog box titled "What Rule to Run". It contains a dropdown menu with "OutputCustomRec" selected, a small blue square button with a white "v" icon, and a small square button with a white "fx" icon. Below this, there is a "Text" button and the text "CLM".

With Instream, each OutputCustomRec command will first generate a DTL record, to identify the location of the custom record in the data.

## RemoveCustomRecord

---

### Instream and HIPAA Validator Desktop

---

Removes one or all custom record definitions and their outputs.

It is good practice to remove all custom records on the SE segment. This prevents the definitions from remaining during further analyses for data in the same functional group.

You may also find this command useful for Instream. You can attach it to a location where you no longer wish to see the automatic custom record.

#### Format of Parameters

*ID ID ID ...*

Where:

*ID* ID of the record: 1 to 4 alphanumeric characters. This record was set up with a DefineCustomRec rule. If the ID is omitted, all custom records are removed.

Use a space to separate multiple record names.

**Example.** This rule removes the custom record CLM:



What Rule to Run

RemoveCustomRec

Text CLM



# Date and Time Business Rules

## CheckDateInRange

---

### All validation programs

---

Verifies whether a date falls within a range and takes an action if the test is true.

#### Format of Parameters

*DateToCheckFormatCode* *DateToCheck* *Operand* *DateRangeFormatCode* *DateRange* (*IfTrueAction*)

Where:

*DateToCheckFormatCode* The format of the date to check: a variable, literal in double quotes, or `Current_Element`. Possible formats are:

|    |                  |
|----|------------------|
| D6 | for YYMMDD       |
| D8 | for YYYYMMDD     |
| DT | for YYMMDDHHMMSS |

*DateToCheck* The date that may or may not be in range. This can be a variable, literal in double quotes, `Current_Date`, or `Current_Element`.

*Operand*

`InRange` - Date is within the range, and is not the starting or ending date of the range.

`OutOfRange` - Date is outside the range, and is not the starting or ending date of the range.

`InRangeEqual` - Date is within the range or is the starting or ending date of the range.

`OutOfRangeEqual` - Date is outside the range, or is the starting or ending date of the range.

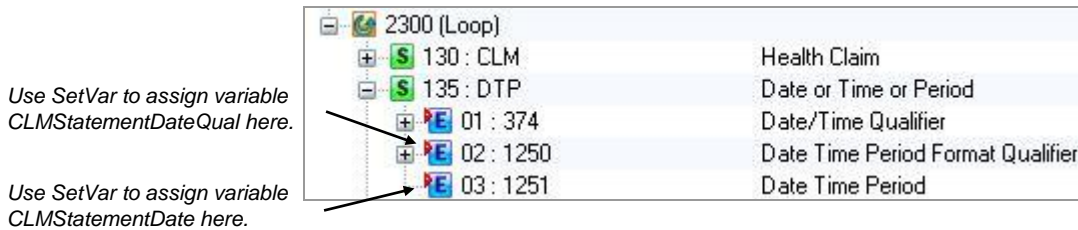
*DateRangeFormatCode* The format of the date range, always RD8 for YYYYMMDD-YYYYMMDD format.

*DateRange* A variable, literal in double quotes, or `Current_Element`.

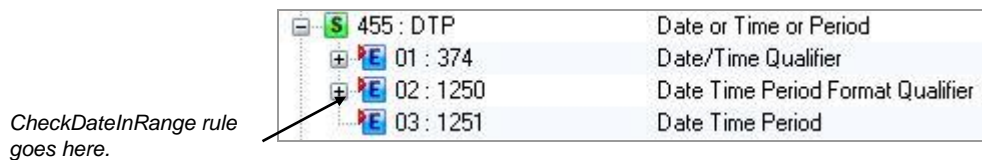
(*IfTrueAction*) Optional. The action to be executed if the result is true. If omitted, a default message will be displayed.

**Example.** This example checks that service line dates are within the range specified in the DTP Statement Dates segment in the CLM loop. If not, it issues an error message.

To set this up, we need to put variables on the Statement Dates DTP02 and DTP03:



We will also need a variable on the Service Line Date DTP02, which offers a choice of two codes.



We can then use these variables in a rule on the Service Line Date DTP03:

What Rule to Run

CheckDateInRange

Text ServiceDateQual Current\_Element OutRange RD8 CLMStatementDate (BusinessRules.Utilities.DisplayErrorByNumber 32232)

If the rule is to be used with Analyzer, the DisplayErrorByNumber format is slightly different.

If it is possible that other date formats would appear in the data, then the rule would need to become more complex, with CompareString used to check the formats of CLMStatementDateQual and conditionally execute the rule above.

## CompareDate

---

### All validation programs

---

Compares two dates based on the operand and performs an action if the result is true.

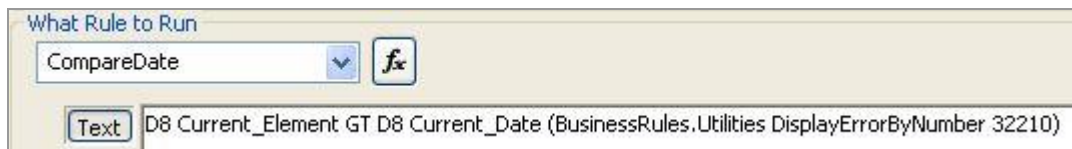
#### Format of Parameters

*DateFormatCodeA* *DateVarA* *Operand* *DateFormatCodeB* *DateVarB* (*IfTrueAction*)

Where:

|                        |   |                  |
|------------------------|---|------------------|
| <i>DateFormatCodeA</i> | D6  | for YYMMDD       |
|                        | D8  | for YYYYMMDD     |
|                        | DT  | for YYYYMMDDHHMM |
| <i>DateVarA</i>        | A variable, literal in double quotes, <i>Current_Date</i> , or <i>Current_Element</i> .                     |                  |
| <i>Operand</i>         | EQ, NE, GT, GE, LT, or LE.  |                  |
| <i>DateFormatCodeB</i> | D6  | for YYMMDD       |
|                        | D8  | for YYYYMMDD     |
|                        | DT  | for YYYYMMDDHHMM |
| <i>DateVarB</i>        | A variable, literal in double quotes, <i>Current_Date</i> , or <i>Current_Element</i>                       |                  |
| <i>(IfTrueAction)</i>  | Optional. The action to be executed if the result is true. If omitted, a default message will be displayed. |                  |

**Example.** This example checks the service line date to see if it is in the future. If this is true, it displays custom error message 32210, which might be "Service cannot have been performed in the future."



If the rule is to be used with Analyzer, the DisplayErrorByNumber format is slightly different.

|             |              |                        |
|-------------|--------------|------------------------|
| 2400 (Loop) |              |                        |
| +           | \$ 365 : LX  | Assigned Number        |
| +           | \$ 375 : SV2 | Institutional Service  |
| +           | \$ 385 : SV4 | Drug Service           |
| +           | \$ 420 : PwK | Paperwork              |
| -           | \$ 455 : DTP | Date or Time or Period |
|             | +            | E 01 : 374             |
|             | +            | E 02 : 1250            |
|             |              | E 03 : 1251            |

Rule goes here. →

## DateCalc

---

### All validation programs

---

DateCalc can be used in several ways:

Date Ranges:

- Calculate how many days or months are in a date range and put the result in a variable.
- Take action (such as displaying an error) based on the relationship between two dates.

Individual Dates:

- Put the results of a calculation between two dates into a variable.
- Display an error based on the calculation between two dates.

#### Example

| Dates                 | Number of days | Number of months |
|-----------------------|----------------|------------------|
| 20070922-20071021     | 21             | 1                |
| 20070922 and 20050922 | 730            | 24               |

---

### Date Ranges: Calculate how many days or months are in an RD8 date range

---

DateCalc can calculate how many days or months are between the two dates in a date range. This element would have a RD8 qualifier that contains a value like 20071028-20071204.

#### Format of Parameters

**BYMONTH RD8** *DateToCheck* *ResultVariable*

Where:

|                       |   |
|-----------------------|---|
| <b>BYMONTH</b>        | Literal. The difference between the two dates is to be calculated in months. If omitted, the difference will be calculated in days. |
| <b>RD8</b>            | Literal.  |
| <i>DateToCheck</i>    | The location of the date range: a variable, literal in double quotes, Current_Date or Current_Element.                              |
| <i>ResultVariable</i> | A variable to hold the result of the calculation.   |

### Example 1 - number of days in a date range

This example calculates the number of days in the range contained in the current element and places the result in variable NumOfServiceDays.

What Rule to Run

DateCalc1

Text: RD8 Current\_Element NumOfServiceDays

Another rule might then check the value in NumOfServiceDays and take some action:

What Rule to Run

CompareNumeric

Text: NumOfServiceDays GT "365" (BusinessRules.Utilities.DisplayErrorByNumber 0 0 "The number of service days is more than 365")

### Example 2 - number of months in a date range

This example is the same as the previous, but it calculates the number of months in the range

What Rule to Run

DateCalc1

Text: BYMONTH RD8 Current\_Element NumOfServiceM

### Example 3 - checking the qualifier first

Set a variable on the element Date Time Period Format Qualifier (in this example, we will use DateFormat).

It then checks the variable to see if it contains RD8. If so, it calculates the number of days in the range and places the result in variable NumOfServiceDays.

When to Run Rule

☐ Always

☒ Conditionally

|        | Local Variable | Operator | Constant |
|--------|----------------|----------|----------|
| Single | DateFormat     | EQ       | RD8      |
|        |                | EQ       |          |

What Rule to Run

DateCalc1

Text: RD8 Current\_Element NumOfServiceDays

#### Example 4: Creating a 6-month date range

This example creates a 6-month date range on the fly. The FutureDate variable will contain the date 6 months from the Current\_Date. It will be in D8 format. You can then use the FutureDate to create a date range and check a date against that range.



What Rule to Run

DateCalc1

Text BYMONTH D8 Current\_Date + 6 FutureDate

#### Date Ranges: Take action based on the relationship between two RD8 dates

DateCalc can take action based on the relationship between a date range and another number.

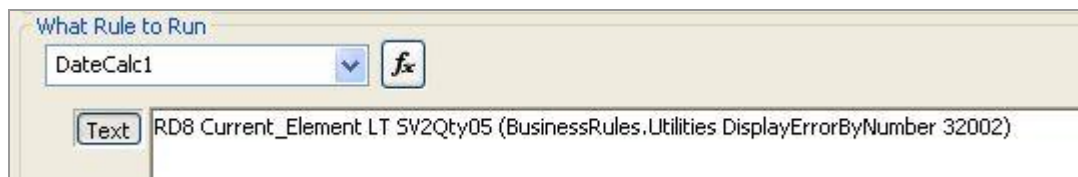
##### Format of Parameters

**BYMONTH RD8** *DateToCheck Operand IntegerVar (action if true)*

Where:

|                         |  |
|-------------------------|--|
| <b>BYMONTH</b>          | The value is to be calculated in months. If omitted, the difference will be calculated in days.        |
| <b>RD8</b>              | Literal.   |
| <i>DateToCheck</i>      | The location of the date range: a variable, literal in double quotes, Current_Date or Current_Element. |
| <i>Operand</i>          | Any of these: <b>EQ</b> , <b>NE</b> , <b>GT</b> , <b>LT</b> , <b>GE</b> , <b>LE</b>                    |
| <i>IntegerVar</i>       | A variable containing an integer, or a literal in double quotes (such as "5").                         |
| <i>(action if true)</i> | Action to be taken if the calculation is true.   |

**Example.** This displays an error message if the number of days in the current date range is less than the integer in the variable SV2Qty05.



What Rule to Run

DateCalc1

Text RD8 Current\_Element LT SV2Qty05 (BusinessRules.Utilities DisplayErrorByNumber 32002)

If the rule is to be used with Analyzer, the DisplayErrorByNumber format is slightly different.

## Individual Dates: Store the results of a calculation between dates in a variable

---

DateCalc can calculate how many days or months are between dates in two separate elements and put the result in a variable.

### Format of Parameters

**BYMONTH** *DateFormat1* *Date1* - *DateFormat2* *Date2* *ResultVariable*

or

**BYMONTH** *DateFormat1* *Date1*  $\pm$  *Integer* *ResultVariable*

Where:

|                       |   |
|-----------------------|---|
| <b>BYMONTH</b>        | The value is to be calculated in months. If omitted, the difference will be calculated in days.                                       |
| <i>DateFormat1</i>    | The format of the first date:<br><br>If BYMONTH: <b>D6</b> , <b>D8</b> , or <b>DT</b> , or a variable containing one of these values. |
| <i>Date1</i>          | The location of the first date: a variable, literal in double quotes, <i>Current_Date</i> or <i>Current_Element</i> .                 |
| -                     | A literal minus sign surrounded by spaces.  |
| <i>DateFormat2</i>    | The format of the second date: <b>D6</b> , <b>D8</b> , or <b>DT</b> , or a variable containing one of these values.                   |
| <i>Date2</i>          | The location of the second date: a variable, literal in double quotes, <i>Current_Date</i> , or <i>Current_Element</i> .              |
| <i>ResultVariable</i> | A variable to store the number of days between the two dates.   |
| $\pm$                 | A plus sign or a minus sign surrounded by spaces.   |
| <i>Integer</i>        | A literal in double quotes or a variable containing an integer.   |

**Example.** This example calculates the number of days between the date in variable BHT04StatementDate and the date in the current element. The result goes into variable DelayInSubmitting.

What Rule to Run

DateCalc1

Text D8 BHT04StatementDate - D8 Current\_Element DelayInSubmitting



## Individual Dates: Display an error based on the calculation between two dates

---

DateCalc can calculate how many days are between dates in two separate elements and put the result in a variable.

### Format of Parameters

**BYMONTH** *DateFormat1* *Date1*  $\pm$  *DateFormat2* *Date2* *Operand* *IntegerVar* (*action if true*)

Where:

|                           |  |
|---------------------------|--|
| <b>BYMONTH</b>            | The value is to be calculated in months. If omitted, the difference will be calculated in days.                          |
| <i>DateFormat1</i>        | The format of the first date: <b>D6</b> , <b>D8</b> , or <b>DT</b> , or a variable containing one of these values.       |
| <i>Date1</i>              | The location of the first date: a variable, literal in double quotes, <i>Current_Date</i> , or <i>Current_Element</i> .  |
| $\pm$                     | A plus sign or a minus sign surrounded by spaces.  |
| <i>DateFormat2</i>        | The format of the second date: <b>D6</b> , <b>D8</b> , or <b>DT</b> , or a variable containing one of these values.      |
| <i>Date2</i>              | The location of the second date: a variable, literal in double quotes, <i>Current_Date</i> , or <i>Current_Element</i> . |
| <i>Operand</i>            | Any of these: <b>EQ</b> , <b>NE</b> , <b>GT</b> , <b>LT</b> , <b>GE</b> , <b>LE</b> .                                    |
| <i>IntegerVar</i>         | A variable containing an integer, or a literal in double quotes (such as "5").   |
| ( <i>action if true</i> ) | Action to be taken if the calculation is true.   |

**Example.** This example calculates the number of days between the date in variable BHT04StatementDate and the date in the current element. If the result is more than 365, display custom error message 32003.

What Rule to Run

Rule: DateCalc1

Text: D8 BHT04StatementDate - D8 Current\_Element GT "365" (BusinessRules.Utilities DisplayErrorByNumber 32003)

If the rule is to be used with Analyzer, the DisplayErrorByNumber format is slightly different.

### Complete example

---

This set of rules in a HIPAA 837 will display an error if the transaction date and the date of service are more than 4 months apart.

First, capture the transaction date on the BHT04:

What Rule to Run

Rule: SetVar

Text: BHT04TransactionDate

Go to the 2400 DTP03 for service date. In the first rule, capture the first 8 digits of the date:

What Rule to Run

SubString

Text: DateOfService Current\_Element 1 8

In the second rule, capture the number of elapsed months:

What Rule to Run

DateCalc1

Text: BYMONTH D8 BHT04TransactionDate - D8 DateOfService NumberOfMonths

In the third rule, display an error if the number of months is greater than 4:

What Rule to Run

CompareNumeric

Text: NumberOfMonths GT "4" (BusinessRules.Utilities.DisplayErrorByNumber 32001)

## Add or Subtract Hours and Adjust Date Accordingly

BYHOUR rules tell you how many days would have to be added or subtracted when you add or subtract hours from a time. It also reports the new time.

For example, if the data contained a time of 1800 and you added 10 hours, it would report that the time would be 0400, and one day would be added.

### Format of Parameters

**BYHOUR** *DateFormat1* *Date1* *HourChange* *DateFormat2* *Date2* *DaysAdded*

Where:

|                    |   |
|--------------------|---|
| BYHOUR             | Literal. Specifies that the difference between the two dates is to be calculated in hours.  |
| <i>DateFormat1</i> | <p>The format of the time, one of these: TSDD, DT, TSD, TM or TS.</p> <p>If unknown, use XX and validation will look at the value's length to pick one of these formats:</p> <ul style="list-style-type: none"> <li>8 TSDD = HHMMSSDD</li> <li>12 DT = CCYYMMDDHHMM</li> <li>7 TSD = HHMMSSD</li> <li>4 TM = HHMM</li> <li>6 TS = HHMMSS</li> </ul> |
| <i>Date1</i>       | The original time: a variable, literal in double quotes, Current_Time, or Current_Element.  |

|                    |  |
|--------------------|--|
| <i>HourChange</i>  | Hours to be added or subtracted - a variable or literal.<br><br>Examples:<br>8<br>-8<br>TimeChangeVar                    |
| <i>DateFormat2</i> | This should be the same as <i>DateFormat1</i> .  |
| <i>Date2</i>       | A variable to contain the new date and time, after the hours have been added to or subtracted from <i>Date1</i> .        |
| <i>DaysAdded</i>   | A variable containing the number of days that would need to be added or subtracted to a date because of the time change. |

**Example.** This example calculates the date when 9 hours is added to the time in the GS05. If the new time runs into the next day, this information has to be captured in a variable in place of the date in the GS04.

GS\*BE\*901234572000\*908887732000\***20100926**\***1615**\*3466\*X\*004010X095A1~

1. Set a variable on the GS04 (a date):

| What Rule to Run |                |                 |
|------------------|----------------|-----------------|
| SetVar           |                |                 |
| Text             | Parameter Name | Parameter Value |
|                  | VarToAssign    | GS04Var         |
|                  | Value          | Current_Element |

This captures **20100926** in GS04Var.

2. Set up this rule on the GS05 (a time), which:
  - Adds 9 hours to the value in the GS05 and stores the result in variable NewGS05Var.
  - Determines that a time of 1615 + 9 hours = 2515. This means it is actually 0115 one day later. The 1 is stored in DayAddedVar.

DateTime.DateCalc:BYHOUR TM Current\_Element 9 TSDD NewGS05Var DaysAddedVar

3. Calculate the new date with this rule, also on the GS05:

DateTime.DateCalc:BYDAY D8 GS04Var DaysAddedVar D8 NewGS04Var

## Add Days to an existing Date

---

BYDAY rules add days to an existing date and put the result in a variable.

### Format of Parameters

**BYDAY** *DateFormat1* *OriginalDate* *DaysAdded* *DateFormat2* *NewDate*

Where:

|                     |   |
|---------------------|---|
| <b>BYDAY</b>        | Literal. The difference between the two dates is to be calculated in days.  |
| <i>DateFormat1</i>  | D6, D8, or DT.  |
| <i>OriginalDate</i> | Input date. A literal in double quotes, a variable, Current_Element, or Current_Date.   |
| <i>DaysAdded</i>    | A literal in double quotes or a variable containing the number of days to add. If the value is negative, it will be subtracted from OriginalDate. |
| <i>DateFormat2</i>  | This should be the same as <i>DateFormat1</i> .   |
| <i>NewDate</i>      | A variable to hold the new date.  |

**Example.** See the `DateTime.DateCalc:BYDAY` rule above.

## GetGMTDateTime

Reports the current GMT date and time.

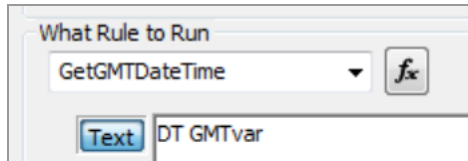
### Format of Parameters

*format* *resultVar*

Where:

|                  |   |                               |
|------------------|---|-------------------------------|
| <i>format</i>    | Format of the output, one of these:             |                               |
|                  | <b>Format</b>                                   | <b>Output will look like:</b> |
|                  | RTS   | CCYYMMDDHHMMSS                |
|                  | DT  | CCYYMMDDHHMM                  |
|                  | TSD   | HHMMSSD                       |
|                  | TSDD  | HHMMSSDD                      |
|                  | MM-DD-CCYY                                      | MM-DD-CCYY                    |
|                  | HH:MM:SS  | HH:MM:SS                      |
| <i>resultVar</i> | Variable to hold the current GMT date and time. |                               |

**Example:** This rule puts the current GMT date and time in variable GMTvar, using format CCYYMMDDHHMM. GMTvar might contain something like this: 201104012142



## ValidateDateTime

---

### All validation programs

---

Validates any value or element against the specified date and time format.

#### Format of Parameters

*Formatstring value*

Where:

*Formatstring*                      Format for date and time. This value can be an X12 or EDIFACT code (e.g., DT or 10) or a format you create using the allowable format strings. See information below for **Allowable Format Strings, X12 Date/Time Codes, and EDIFACT Date/Time Codes**.

*value*                                Value to be checked (constant, variable, internal keyword, etc.).

#### Examples

Validate the value against EDIFACT Date Code 102:

```
ValidateDateTime "102" "20190111"
```

Validate the value against X12 Date Code D8:

```
ValidateDateTime "D8" "20190111"
```

Validate the value against the specified date format:

```
ValidateDateTime "CCYYMMDD" "20190111"
```

Validate the value against the specified date format:

```
ValidateDateTime "CCYYMMDD+" "20190111abc"
```

#### Allowable Format Strings

Format string can be made up of these character strings:

- CC = Century
- YY = Year
- MM = Month (01-12)
- DD = Day (01-31 depending on Month)
- hh = Hour (00-23)

- mm = Minute (00-59)
- ss = second (00-59)
- Trailing '+' = extra characters are accepted

### **X12 Date/Time Codes**

D8 = CCYYMMDD

DT = CCYYMMDDHHMM

D6 = YYMMDD

DTS = CCYYMMDDHHMMSS

RD6 = YYMMDD-YYMMDD

RD8 = CCYYMMDD-CCYYMMDD

RDT = CCYYMMDDHHMM-CCYYMMDDHHMM

RTS = CCYYMMDDHHMMSS-CCYYMMDDHHMMSS

TM = HHMM

TS = HHMMSS

TSD = HHMMSSD

TSDD = HHMMSSDD

### **EDIFACT Date/Time Codes**

2 = DDMMYY

3 = MMDDYY

4 = DDMMCCYY

5 = DDMMCCYYHHMM

10 = CCYYMMDDTHHMM

101 = YYMMDD

102 = CCYYMMDD

106 = MMDD

201 = YYMMDDHHMM

202 = YYMMDDHHMMSS

203 = CCYYMMDDHHMM

204 = CCYYMMDDHHMMSS

205 = CCYYMMDDHHMMZHHMM

401 = HHMM

402 = HHMMSS

713 = YYMMDDHHMMYYMMDDHHMM

717 = YYMMDDYYMMDD

718 = CCYYMMDDCCYYMMDD

719 = CCYYMMDDHHMMCCYYMMDDHHMM

## ValidateDateTimeUN and ValidateDateTimeX12

---

### All validation programs

---

|                            |  |
|----------------------------|--|
| <b>ValidateDateTimeX12</b> | Place this rule on an X12 element 1251. It checks the date and time in the current element to see if it follows the format specified in the preceding element 1250. Be sure to customize the code values for the qualifier in element 1250 |
| <b>ValidateDateTimeUN</b>  | Place this rule on an EDIFACT element 2380 to see if it follows the format specified in the following element 2379. Be sure to customize the code values for the qualifier in element 2380.  |

### Format of Parameters

*<falseRule>*

Where:

*falseRule* (Optional) The rule to be executed if ValidateDateTimeX12 or ValidateDateTimeUN check fails. This parameter must be another rule to run. If omitted, a default message is displayed.

**Example.** This rule checks X12 element 1251.

The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "ValidateDateTimeX12" selected, a button with a function symbol (fx), and a checkbox labeled "Look-Ahead Rule" which is unchecked. Below this is a "Text" button and a text field containing "(<falseRule>.Original)".

**Example.** This rule checks EDIFACT element 2380. If the check fails, Error 32001 is displayed.

The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "ValidateDateTimeUN" selected, a button with a function symbol (fx), and a checkbox labeled "Look-Ahead Rule" which is unchecked. Below this is a "Text" button and a text field containing "(BusinessRules.Utilities DisplayErrorByNumber 32001)".

# DBServer Business Rules

---

## Instream on UNIX

---

### Important

Validating with these rules requires additional setup outside of the guideline. Please see **TIB\_instream\_<n.n>\_isiserver.pdf** for details.

These functions interact with Oracle or SQL Server databases from Instream that is running on AIX:

- The **DBExecute** function runs a stored procedure. See below.
- The **DBQuery** function sends a query. See page [92](#).

The **InvokeWebService** function sends an array to a web service and receives one back.

## DBExecute

Executes a stored procedure in an Oracle or SQL Server database.

### Format of the Parameters

*DBRef* *ReturnCode* "StoredProc params" {var1=1 {var2=2 ...}} {(business rule)}

Where:

*DBRef* Name of a database connection specified in the ISIsServer.config's [ORACLE] or [SQL] sections (see **TIB\_instream\_<n.n>\_isiserver.pdf**).

Example:

The *DBRef* is ORACLEDB in this business rule parameter:

**ORACLEDB** RetVal "check\_NPI" NPIactive=1

It is also ORACLEDB in the ISIsServer.config:

**ORACLEDB**=DATABASE{192.168.1.74:1521/or10};USER{ISUsr};PWD{Q1W2E3}

*ReturnCode*

The name of a variable to contain the success of the DBExecute rule. It will always contain 1. Therefore, please check the output variables from the stored procedure to determine what action to take.

If the rule failed, the DTL file will have more information like this:  
EMSG 10SQL ERROR : [BRDatabase::DBExecute -  
SQLExecDirect Failed. [-1]]



### *StoredProc params*

The stored procedure to execute followed by a space and its input parameters, each separated by one space. You can include business rule variables in the stored procedure's name in the form *%var%* where *var* is the name of a business rule variable. Before the command is processed, *var* will be replaced with the contents of the specified variable.

In the stored procedure, always put the output parameters first, like this example:

```
CREATE PROCEDURE verifyXXXx
(@variableout varchar(50) out,
@variablein   varchar(50)
)
```

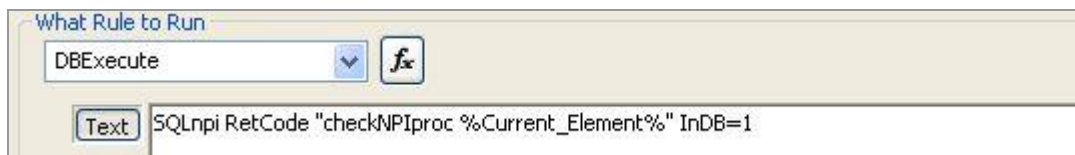
*{var1=1 {var2=2 ...}}*

Variables to contain the values returned from the procedure. The contents of these variables are automatically set to null strings before the business rule executes the procedure. The “=1” means it is the first parameter returned from the procedure, the “=2” means it is the second parameter returned, etc.

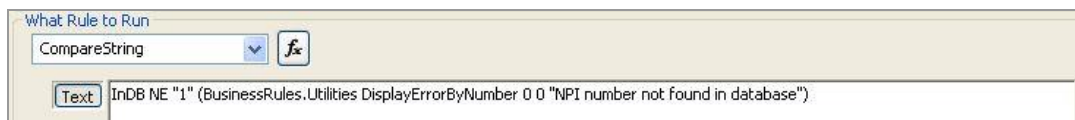
*{{(business rule)}}*

Optional business rules to run at the end. See example 2 below. This is especially useful in end of loop rules, which run in reverse order.

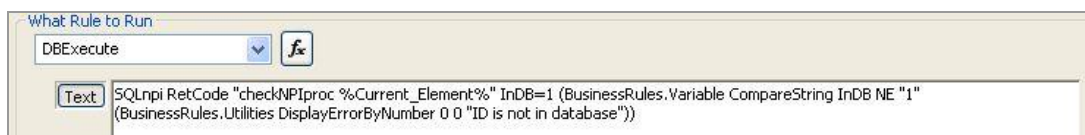
**Example 1:** This rule executes a stored procedure called `checkNPIproc` in the database referenced by `SQLnpi` in `ISIServer.config`. It sends the value in the current element as the proc's one input parameter. The only returned item is stored in variable **InDB**.



This rule then checks the procedure's output parameter `InDB` and displays an error if it equals 1:



**Example 2:** This rule executes the `checkNPIproc`, checks the procedure's `InDB` output value, and displays a message if it doesn't contain 1.



## ***Connecting to Stored Procedures using DBExecute***

Note the following when connecting to stored procedure using DBExecute:

- The first parameter in the stored procedure **must** be an output parameter and, when calling the stored procedure from TIBCO Foresight, there **must** be a value in that output parameter.
- All input parameters must be VARCHAR.
- Any stored procedure that TIBCO Foresight uses **must** have the SQL statement SET NOCOUNT ON at the top of the procedure. If not, an “Invalid cursor state” error is generated.
- The ODBC DBQuery command is used to call a stored procedure even when there are no return values.

## DBQuery

Performs a SQL or Oracle query on a specified database.

### Format of the Parameters

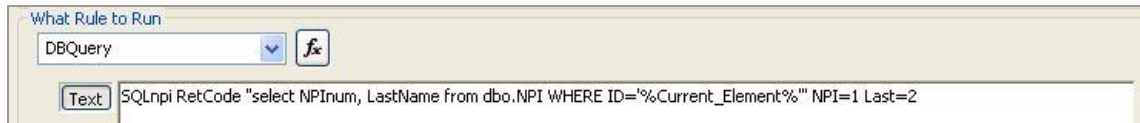
```
DBRef ReturnCode "SqlStatement" {var1=1 {var2=2 ...}}
```

Where:

|                              |   |
|------------------------------|---|
| <i>DBRef</i>                 | Name of a database connection specified in the ISIservr.config's [ORACLE] or [SQL] sections.  |
| <i>ReturnCode</i>            | <p>The name of a variable to contain the success of the <a href="#">database</a> query. It will contain one of these:</p> <p><b>0</b> = database query failed (check database setup)<br/><b>1</b> = database query succeeded</p> <p>If the query failed, the DTL file will have an EMSG with more information.</p> <p><b>Note:</b> This return code does not indicate success or failure of the business rule. It indicates success/failure of the database query. Success or failure of the business rule is indicated in {var1=1 {var2=2 ...}}.</p> |
| <i>"SqlStatement"</i>        | <p>SQL command to execute. The SQL command must return a recordset. The SQL command can contain business rule variables in the form %var% where <i>var</i> is the name of a business rule variable. Before the SQL command is processed, variables in the SQL string will be replaced with the contents of the specified variable.</p>  |
| <i>{var1=1 {var2=2 ...}}</i> | <p>Variables to contain the values returned from the query. The contents of these variables are automatically set to null strings before the business rule executes the query. The "=1" means it is the first value returned from the procedure, the "=2" means it is the second query returned, etc.</p>   |
| <i>{{business rule}}</i>     | <p>Optional business rules to run at the end. See example 3 below. This is especially useful in end of loop rules, which run in reverse order.</p>  |

## Example 1

This rule queries a database to see if the current element is in table NPI in the database with connection name SQLnpi.

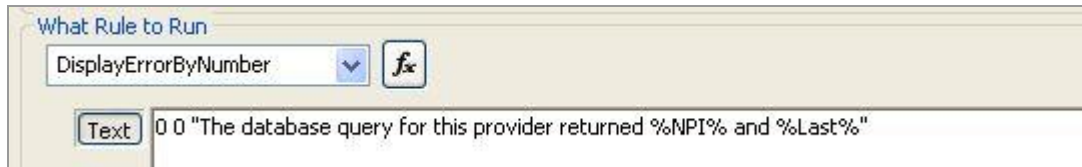


What Rule to Run

DBQuery

Text: SQLnpi RetCode "select NPInum, LastName from dbo.NPI WHERE ID="%Current\_Element%" NPI=1 Last=2"

This rule then uses the values in the returned variables NPI and Last:



What Rule to Run

DisplayErrorByNumber

Text: 0 0 "The database query for this provider returned %NPI% and %Last%"

DTL file results:

```
EMSG          10The database query for this provider returned 123456789 and
Qian
```

The Server-Thread log results in ISIservers fslog directory show success:

```
SQL : connection=DRIVER={SQL Server};SERVER=TI-TEST;DATABASE=
SQL : open success on DB=SQLnpi;SERVER=TI-TEST;DATABASE=LindaTest
getDBRef[SQLnpi] success.
SQL : select NPInum, LastName from dbo.NPI WHERE LastName='Q
Return value = [9      1      1234567894      2      Qian]
LookUp In : DB=SQLnpi;SERVER=TI-TEST;DATABASE=LindaTest
LookUp Success
```

## Example 2

This rule captures the value in the NM103 into this variable:

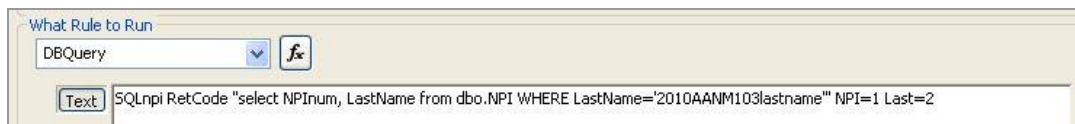


What Rule to Run

SetVar

Text: 2010AANM103lastname

We then use this value in our DBQuery:



What Rule to Run

DBQuery


Text: SQLnpi RetCode "select NPInum, LastName from dbo.NPI WHERE LastName='2010AANM103lastname' NPI=1 Last=2"

### Example 3

---

This example ends with a rule that checks the return code and displays an error message if the query failed.

What Rule to Run

DBQuery  ☐ Look-Ahead Rule

**Text** SQLnpi RetCode "select NPInum, LastName from dbo.NPI WHERE ID=%Current\_Element%" NPI=1 Last=2 (BusinessRules.Variable CompareString RetCodeEQ "0" (BusinessRules.Utilities DisplayErrorByNumber 0 0 "Database call failed"))

## InvokeWebService

---

### HIPAA validation programs

---

In addition to a business rule developer, this requires:

- A Java or web services developer to create a Java class.
- Someone to install and configure TIBCO Foresight's ISIServer.

Please see **TIB\_instream\_<n.n>\_isserver.pdf** for instructions on these steps.

For an extended example, see Appendix J: LookAhead and Array Extended Example on page [295](#).

TIBCO Foresight web services business rules give you a standardized way to send information out to your own components. Instream acts as the client to your external web service.

You will need to create a Java class to serve as the client to your web service. At runtime, we will call the class and invoke it according to the contract.

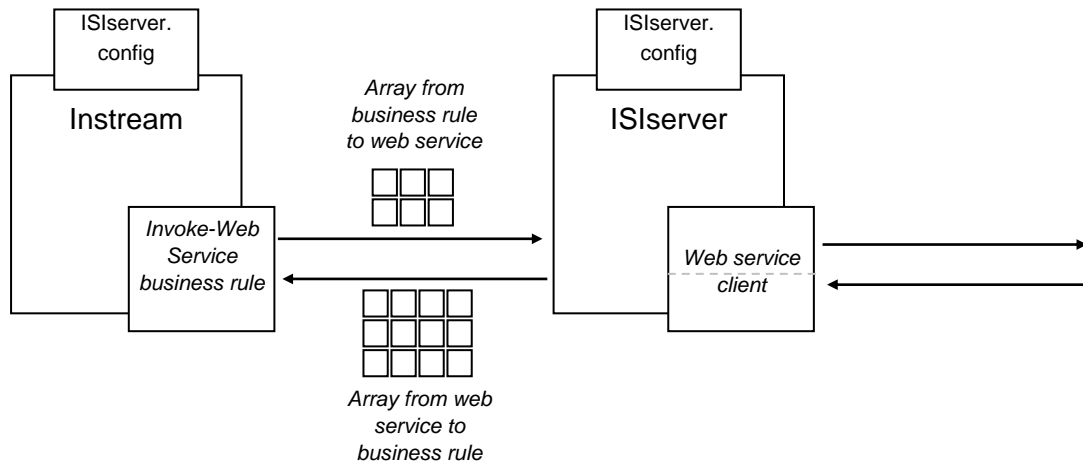
For instance, you might send a subscriber ID and the corresponding service line dates to a web service. The web service might then execute a database lookup and return information about whether the subscriber was covered on those dates. Another business rule could then check the results and display an error message if they were not covered.

InvokeWebService is compliant with WS-I version 1.1 and tested in Java and .NET environments. It sends an array to your web service and receives an array in return.

## Overview

---

InvokeWebService sends a business rule array to TIBCO Foresight's ISIs server program, which passes it on to your own web service. It receives a response array that your guideline can use with array business rules.



## During validation

---

1. Before starting Instream or HIPAA Validator Desktop validation, start ISIServer.exe.
2. Instream or HIPAA Validator Desktop reaches an InvokeWebService business rule during validation.
3. This creates an instance of the Java class and sends an array of information to it.
4. The class can then perform any operations it requires to work with the data.
5. The web service returns another array to Instream or HIPAA Validator Desktop.

## Format of Parameters

---

*WSName inputArrayName outputArrayName (action)*

Where:

*WSName* Web service reference name; must match the name in ISIServer.config's[WEBSERVICES] section:

```
[WEBSERVICES]
:Connection of web services
:Format : CLASSNAME{"classname"};CLASSPA
:wschecksub=CLASSNAME{"com.foresightcorp.
chksubsrv=CLASSNAME{"com.foresightcorp.De
```

*inputArrayName* Name of the array being sent to the web service. This must be a literal in double quotes. It was defined and populated with Array business rules (see page [34](#)).

*outputArrayName* Name of the array being returned from the web service. This must be a literal in double quotes. It was defined with a CreateArray business rule (see page [42](#)).


*action* Optional. The action to be executed if the call fails.

## Examples

---

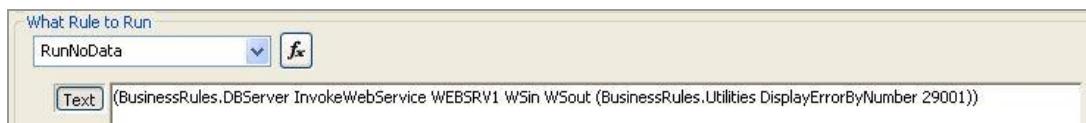
These all invoke a web service called ChkSubSrv. They send an array called ArrayTOsrv and receive back an array called ArrayFROMsrv. If the web service fails to start, a message displays.

**Example 1.** This is the simplest way to invoke a web service.



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu set to "InvokeWebService" and a button with a function icon (fx). Below this is a "Text" field containing the expression: `WEBSRV1 WSin WSout (BusinessRules.Utilities DisplayErrorByNumber 29001)`.

**Example 2.** This invokes the same web service only if there is no data on the current element.



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu set to "RunNoData" and a button with a function icon (fx). Below this is a "Text" field containing the expression: `(BusinessRules.DBServer InvokeWebService WEBSRV1 WSin WSout (BusinessRules.Utilities DisplayErrorByNumber 29001))`.

**Example 3.** This invokes the same web service at the end of each instance of the 2000B loop.



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu set to "InvokeWebService" and a button with a function icon (fx). Below this is a "Text" field containing the expression: `WEBSRV1 WSin WSout (BusinessRules.Utilities DisplayErrorByNumber 29001)`.



# Exit Business Rules

An exit causes a rule to run every time a certain event occurs. Example: whenever a certain element, composite, or segment is encountered, or whenever a certain loop ends.

If you have multiple Exit rules for a particular item, they run in reverse order. See Appendix I: Processing Order on page 291.

---

## Recommendation

When using these rules, place them on the ST segment, regardless of where they are to run:

SetCompositePreExit  
SetElementPostExit  
SetLoopPostExit  
SetLoopPostInstanceExit  
SetSegmentPreExit

---

## ClearExits

---

### All validation programs

---

Clears all currently-set exits.



A typical use is to place this business rule on the ST segment to clear all exits that may be lingering from previous transactions in the same file.

## KeepOrder

---

### HIPAA Validator Desktop, EDISIM Validator, Instream

---

Causes element, segment, and group/loop exit rules to process in the same order as specified in the Standards Editor Business Rules window.

Normally, **SetCompositePreExit**, **SetElementPostExit**, **SetLoopPostExit**, **SetLoopPostInstanceExit**, and **SetSegmentPreExit** rules execute in reverse order.

KeepOrder is mainly needed within business rule loops.

#### Format of Parameters:

KeepOrder has no parameters.

**Example 1.** Normal execution order for exit rules:

| List of exit rules at a one location:                    | Execution order |
|--|-----------------|
| BusinessRules.Exits.SetSegmentPreExit UNT <i>Rule #1</i> | Rule #5         |
| BusinessRules.Exits.SetSegmentPreExit UNT <i>Rule #2</i> | Rule #4         |
| BusinessRules.Exits.SetSegmentPreExit UNT <i>Rule #3</i> | Rule #3         |
| BusinessRules.Exits.SetSegmentPreExit UNT <i>Rule #4</i> | Rule #2         |
| BusinessRules.Exits.SetSegmentPreExit UNT <i>Rule #5</i> | Rule #1         |

**Example 2.** Effect of KeepOrder rule on execution order for exit rules:

| List of exit rules at a one location:                    | Execution order |
|--|-----------------|
| BusinessRules.Exits.SetSegmentPreExit UNT <i>Rule #1</i> | Rule #2         |
| BusinessRules.Exits.SetSegmentPreExit UNT <i>Rule #2</i> | Rule #1         |
| BusinessRules.Exits. KeepOrder                           | Rule #3         |
| BusinessRules.Exits.SetSegmentPreExit UNT Rule #3        | Rule #4         |
| BusinessRules.Exits.SetSegmentPreExit UNT Rule #4        | Rule #5         |
| BusinessRules.Exits.SetSegmentPreExit UNT <i>Rule #5</i> |                 |

## SetCompositePreExit

---

### All validation programs

---

Calls a function before each occurrence of a specified composite is processed.

#### Format of Parameters

*CompositeID* *ServerName* *FunctionName* *FunctionParms*

Where:

|                      |  |
|----------------------|--|
| <i>CompositeID</i>   | The 4-character composite ID where the function should run.                        |
| <i>ServerName</i>    | The server that should run whenever Validator encounters a composite with that ID. |
| <i>FunctionName</i>  | The function within that server, if the function has parameters.                   |
| <i>FunctionParms</i> | The parameters for that function, if the function has parameters.                  |

**Example.** This example displays a message whenever Validator encounters a composite with ID C024:



Message 32217 is a custom message in file CustomerFSBRERRS.TXT.

If the rule is to be used with Analyzer, the DisplayErrorByNumber format is slightly different.

### Execution Order of Multiple SetCompositePreExit Rules

---

When validating a composite, the rules directly on the composite execute first, and then the pertinent SetCompositePreExit rules execute. Unlike other business rules, these SetCompositePreExit rules execute in the reverse order from how they are listed. See Appendix I: Processing Order on page [291](#).

## SetElementPostExit

---

### All validation programs

---

Calls a function after each occurrence of a specified element is processed. This rule slows down validation significantly.

#### Format of Parameters

*ElementID ServerName FunctionName FunctionParms*

Where:

|                      |   |
|----------------------|---|
| <i>ElementID</i>     | The element ID where the function should run.                                     |
| <i>ServerName</i>    | The server that should run whenever Validator encounters an element with that ID. |
| <i>FunctionName</i>  | The function within that server.  |
| <i>FunctionParms</i> | The parameters for that function, if the function has parameters.                 |

**Example.** This example displays a message 32214 (which might say, for example, "Presence of SBR02 indicates a subscriber-as-patient scenario") whenever Validator encounters element 1069. The rule is placed on the element itself so that it executes only when the SBR02 contains data.



Message 32214 is a custom message in file CustomerFSBRERRS.txt.

If the rule is to be used with Analyzer, the DisplayErrorByNumber format is slightly different.

---

### Execution order of multiple SetElementPostExitRules

---

Rules directly on the element execute before the SetElementPostExit rules.

Unlike other business rules, SetElementPostExit executes in reverse order from how they are listed. See Appendix I: Processing Order on page [291](#).

## SetLoopPostExit

---

### All validation programs

---

Calls a function after completion of all repetitions of the specified loop.

Place this business rule on the ST segment.

#### Format of Parameters

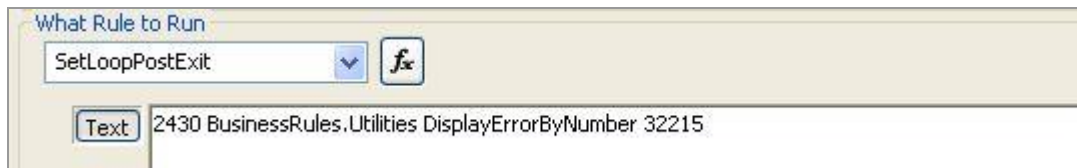
*LoopID ServerName FunctionName FunctionParms*

Where:

|                      |   |
|----------------------|---|
| <i>LoopID</i>        | The loop ID where the function should run.  |
| <i>ServerName</i>    | The server that should run after Validator completes processing all occurrences of the loop with that ID. |
| <i>FunctionName</i>  | The function within that server.  |
| <i>FunctionParms</i> | The parameters for that function, if the function has parameters.   |

**Example.** This example uses AddVar to total the quantities in CAS segments in each 837I claim loop. The variable used for totaling is SLAdjustTot.

Put this SetLoopPostExit rule on the ST segment.



What Rule to Run

SetLoopPostExit

Text 2430 BusinessRules.Utilities DisplayErrorByNumber 32215

At the end of ALL repetitions of loop 2430 for this service line, it displays custom message 32215 ("Total adjustment amount for this claim is <value of SLAdjustTot>"). 2430 is the ID for the Service Line Loop.

This is the error message in CustomerFSBRERRS.TXT:

32215 Total adjustment amount for the claim above is #SLAdjustTot#

If the rule is to be used with Analyzer, the DisplayErrorByNumber format is slightly different.

In this example, the rules can be applied in these locations:

Up to 25 adjudication loops per claim loop.

Place an AddVar rule on all quantities in the CAS segment to accumulate total in a variable called SLAdjustTot.

|             |                              |   |  |    |
|-------------|------------------------------|---|--|----|
| 2430 (Loop) |                              |   |  | 25 |
| 540 : SVD   | Service Line Adjudication    | O |  | 1  |
| 545 : CAS   | Claims Adjustment            | O |  | 99 |
| 01 : 1033   | Claim Adjustment Group Code  | M |  |    |
| 02 : 1034   | Claim Adjustment Reason Code | M |  |    |
| 03 : 782    | Monetary Amount              | M |  |    |
| 04 : 380    | Quantity                     | O |  |    |
| 05 : 1034   | Claim Adjustment Reason Code | X |  |    |
| 06 : 782    | Monetary Amount              | X |  |    |
| 07 : 380    | Quantity                     | X |  |    |
| 08 : 1034   | Claim Adjustment Reason Code | X |  |    |
| 09 : 782    | Monetary Amount              | X |  |    |
| 10 : 380    | Quantity                     | X |  |    |
| 11 : 1034   | Claim Adjustment Reason Code | X |  |    |
| 12 : 782    | Monetary Amount              | X |  |    |
| 13 : 380    | Quantity                     | X |  |    |
| 14 : 1034   | Claim Adjustment Reason Code | X |  |    |
| 15 : 782    | Monetary Amount              | X |  |    |
| 16 : 380    | Quantity                     | X |  |    |
| 17 : 1034   | Claim Adjustment Reason Code | X |  |    |
| 18 : 782    | Monetary Amount              | X |  |    |
| 19 : 380    | Quantity                     | X |  |    |

You will need to use SetVar to reset the variable SLAdjustTot to 0 at the top of the CLM loop to zero the calculation for the next repetition of the loop.

### Execution order of multiple SetLoopPostExecute

After rules execute on all repetitions of a loop, the pertinent SetLoopPostExecute rules execute.

Unlike other business rules, SetLoopPostExecute rules, by default, execute in the reverse order from how they are listed. See Appendix I: Processing Order on page 291.

## SetLoopPostInstanceExit

---

### All validation programs

---

Calls a function after completion of each repetition of the specified loop.

Place this business rule on the ST segment.

This function is exactly like SetLoopPostExit (page 103) except that it executes the function at the end of EACH REPETITION of the loop.

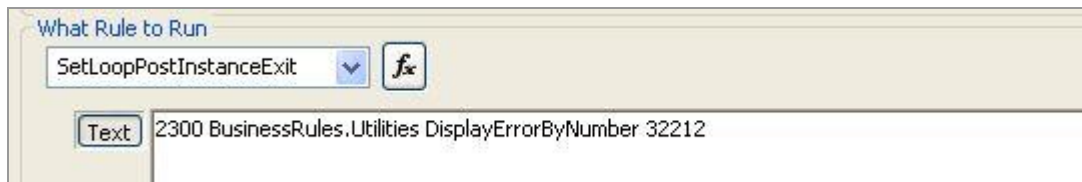
#### Format of Parameters

*LoopID ServerName FunctionName FunctionParms*

Where:

|                      |  |
|----------------------|--|
| <i>LoopID</i>        | The ID of the loop where the function should run.  |
| <i>ServerName</i>    | The server that should run after Validator completes processing of each repetition of the loop with that ID. |
| <i>FunctionName</i>  | The function within that server.   |
| <i>FunctionParms</i> | The parameters for that function, if any.  |

**Example.** This example displays a message at the end of each instance of loop 2300 (the claim loop):



Message 32212 appears in file CustomerFSBRERRS.TXT and might say, for example:

```
End of CLM loop #CLMcount#
```

In Validator, this would display messages like these:

```
End of CLM loop 1
End of CLM loop 2
End of CLM loop 3
```

CLMcount is a variable that counts the number of CLM segments. It was created by placing the following AddVar rule on the CLM segment:



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "AddVar" selected, a blue checkmark icon, and a function icon (fx). Below this, there is a "Text" tab and a text input field containing "CLMcount \"1\"".

Create a SetLoopPostExecute rule for the CLM loop that resets CLMcount to 0 at the end of all repetitions of the CLM loop.

## Execution Order of Multiple SetLoopPostExecute

---

After rules execute on a repetition of a loop, all pertinent SetLoopPostExecute rules execute.

Unlike other business rules, SetLoopPostExecute rules execute in the reverse order from how they are listed. See Appendix I: Processing Order on page 291.

## SetSegmentPreExit

---

### All validation programs

---

Calls a function before each occurrence of a specified segment is processed.

#### Format of Parameters

*SegmentID ServerName FunctionName FunctionParms*

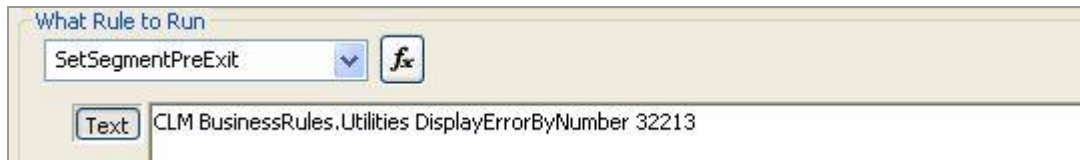
Where:

|                      |  |
|----------------------|--|
| <i>SegmentID</i>     | The 2 or 3-letter segment ID where the function should run.                      |
| <i>ServerName</i>    | The server that should run whenever Validator encounters a segment with that ID. |
| <i>FunctionName</i>  | The function within that server.   |
| <i>FunctionParms</i> | The parameters for that function, if the function has parameters.                |



**Example.** This example displays custom message 32213 ("Beginning of claim number *nnnn*") whenever Validator encounters a CLM segment.

The business rule on the ST segment is:



What Rule to Run

SetSegmentPreExit

Text CLM BusinessRules.Utilities DisplayErrorByNumber 32213

Message 32213 is a custom message in file CustomerFSBRERRS.txt that includes the variable assigned to the claim number:

```
32213 Beginning of claim number #S2300CLM01ClaimNum#
```

## Execution Order of Multiple SetSegmentPreExit

---

All rules directly on the segment execute first.

Then, all pertinent SetSegmentPreExit rules for that segment execute. Unlike other business rules, these execute in the reverse order from how they are listed. See Appendix I: Processing Order on page [291](#).

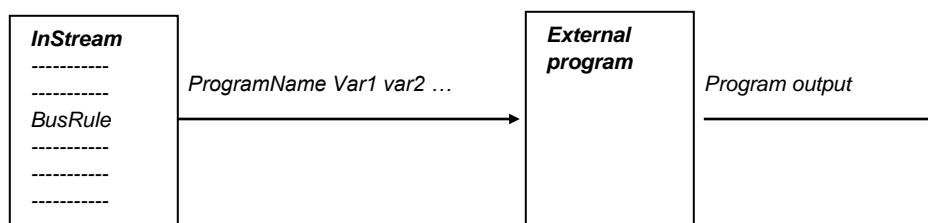
## UserExitWithoutWait

---

### Instream and HIPAA Validator Desktop

---

Starts an external program and immediately continues with validation. The outcome of the external program's activities has no effect on validation.



#### Format of Parameters

*ExecutableName Var1 Var2 ...*

Where:

|                       |   |
|-----------------------|---|
| <i>ExecutableName</i> | Name of executable to process.  |
| <i>Var</i>            | Optional. An input parameter to the executable being processed. This can be a BusinessRules.Variable, a literal in double quotes, or Current_Element. You can include any number of these, each separated by a space. |

---

### Identifying the location of the program called by a user exit

---

Set the environment variable FSUSEREXITS.

You can do this system-wide or within the batch file that runs validation. Do not put quotes around the path for FSUSEREXITS, even if it contains spaces, and don't add a trailing slash:

```
SET FSUSEREXITS=C:\Foresight\InStream\DemoData\UserExits

"C:\Foresight\InStream\Bin\HVINStream.exe"
-i"C:\Foresight\InStream\DemoData\Two837i.txt"
-o"C:\Foresight\InStream\Output\Two837iNW_Results.txt" -gREISSUE
```

---

### HIPAA example

---

This rule checks the BHT02 to see if it 18. If so, it runs an external program that logs the submitter's ID into a file.

**A working demo** of this example is installed with HIPAA Instream. Please see readme-UserExits.txt in Instream's DemoData\UserExits directory.

The rule we are trying to create is:

|                      |  |
|----------------------|--|
| If the BHT02 = 18    | Create a local variable on the BHT02                           |
| Then Run Reissue.bat | Run a UserExitWithoutWait and pass it the NM109 (submitter ID) |

To do this:

1. Create a local variable on the BHT02 (in this example, we will name it BHT02):
2. On the 1000A Submitter NM109-09, test the variable and create the rule to run the UserExitWithoutWait. This rule runs Reissue.bat and passes it the contents of the current element:

|      | Local Variable | Operator | Constant |
|------|----------------|----------|----------|
| when | BHT02          | EQ       | 18       |
| when |                | EQ       |          |

What Rule to Run  
UserExitWithoutWait

Text: Reissue.bat Current\_Element

Reissue.bat might contain:

```
set InStreamRoot= C:\Foresight\InStream
echo %1 >> "%InStreamRoot%\Output\ReissueLog.txt"
```

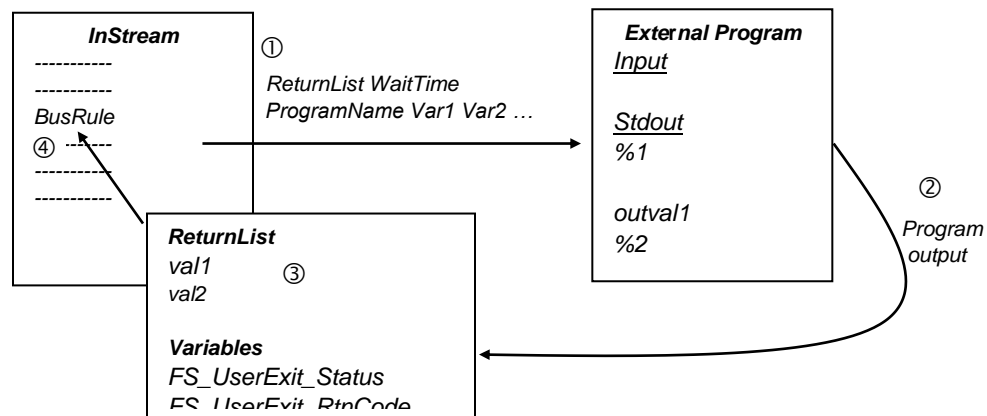
## UserExitWithWait

---

### Instream and HIPAA Validator Desktop

---

Runs an external program and waits up to a specified number of seconds for a response, which it puts into a specified list and then continues with validation.



- ① Business rule runs external program and passes it some values.  
It specifies the name of a list to hold returned values from the external program. It also specifies the number of seconds to wait before killing the external program.
- ② The external program runs and produces output that goes back to Instream via standard output.

- ③ Instream puts the returning values into the list.

It also captures the program's return code and a status into variables **FS\_UserExit\_RtnCode** and **FS\_UserExit\_Status** (see TIBCO Foresight-Defined Variables on page 242).

Statuses can be:

- 200 User Exit business rule has been encountered
- 201 User Exit business rule has been called
- 202 The User Exit business rule has completed
- 203 The User Exit business rule timed out
- 204 The User Exit business rule failed

After the exit rule, you can use a CompareString business rule to check the contents of FS\_UserExit\_Status and FS\_UserExit\_RtnCode and display a message, like this:

... where the error message (32004 in this example) would be something like this:

```
32004 UserExit to run Reissue2.bat return code is #FS_UserExit_RtnCode# and
status is #FS_UserExit_Status#
```

... and the output might be:

```
EMSG 6UserExit to run Reissue2.bat return code is 0 and status is 202
```

- ④ Validation continues, presumably executing additional rules that make use of the list and two variables.

### Format of Parameters

*ResultList WaitTimeInSeconds ExecutableName Var1 Var2 ...*

Where:

*ResultList* BusinessRules.List containing values returned by the UserExit. This list can contain duplicate values.

If the list exists, it is cleared before returned values are added.

If the list does not exist, it is created.

*"WaitTimeInSeconds"* Number of seconds before Instream should continue. This is an integer in double quotes or a variable containing an integer.

If the external program does not send a return code to Instream within the allotted seconds, it is killed and validation continues.

Examples:

"5"

WaitSeconds

*ExecutableName* Name of executable to process.

*Var*

Optional. An input parameter to the external program, which can be a BusinessRule.Variable, a literal in double quotes, or Current\_Element. Use any number of these, each separated by a space.

Please see Identifying the location of the program called by a user exit on page 107 for details about how to set an environment variable that is necessary when using a User Exit.

## Example

This example runs an external file called Reissue1.bat and passes it the contents of the current element. Any value returned from Reissue1.bat goes in list Reissue1. If there is no response from Reissue1.bat within 10 seconds, Reissue1.bat is killed and validation continues.

**A working demo** of this example is installed with Instream. Please see **readme-UserExits.txt** in Instream's **DemoData\UserExits** directory.

This rule is on the 837I 1000A NM1-09:

| When to Run Rule                               |  |
|--|--|
| <input type="radio"/> Always                   |  |
| <input checked="" type="radio"/> Conditionally |  |

|        | Local Variable | Operator | Constant |
|--------|----------------|----------|----------|
| Single | when BHT02     | EQ       | 18       |
|        | when           | EQ       |          |

What Rule to Run

UserExitWithWait

Text: Reissue1 "10" Reissue1.bat Current\_Element

Each time the rule executes, it checks local variable BHT02 to see if it contains 18. If so, this rule runs batch file Reissue1, which might contain, for example:

```
@if "%1%"=="123456789" @echo valid submitter
@if not "%1%"=="123456789" @echo invalid submitter
```

This example checks to see if the current element contains 123456789, the only provider who can submit claims over 10000. It sends “valid submitter” or “invalid submitter” to list Reissue1.

In a subsequent rule, you might run a ListCheck on the list Reissue1 and display an error message if it contains “invalid submitter.”

# ICD Business Rules

Using Instream validation and Dataswapper, you can convert and replace ICD-9 with ICD-10 codes and vice versa. You will need the TIBCO Foresight® ICD-10 Conversion Adapter, which is a separate product.

Business rules for ICD conversion include:

- ICDConvertOne
- ICDConvert
- ICDInsertToArrayWithType

These are described in **ICD\_at\_Foresight.pdf**.

## List Business Rules

This section describes how to accumulate lists of values from an EDI file and then act on them in various ways.

### ClearList

---

#### All validation programs

---

Removes one or all lists from the repository. You should clear a list specifically whenever you want it cleared. Do not count on it being automatically cleared at the end of a transaction set, group, or interchange.

The location of the **ClearList** is important. A typical place is on the first segment in a repeating loop or on the first required element of a repeating segment.

#### Caution

It is hazardous to use ClearList without specifying which list is being cleared. A ClearList without a list name results in clearing all lists, including those in the HIPAA guideline with which you will eventually merge your rules.

#### Format of Parameters

*ListName*


Where:

*ListName*

Optional but recommended. Name of the list to be removed. If *<ListName>* is omitted, all lists are removed. See caution above.

**Example.** This example removes list **2300CRCconditionInd**.

What Rule to Run

ClearList  

Text 2300CRCconditionInd

## InList

---

### All validation programs

---

Adds a value to a list. Takes an action if the value is already in the list.

#### Format of Parameters

*ListName ListValue ifAlreadyInListAction*

Where:



*ListName* Name of the list. If the list does not exist, it is created.

*ListValue* Optional. Value to be added to the list. This can be a variable, a literal in double quotes, or `Current_Element`. If omitted, `Current_Element` is assumed.

*(ifAlreadyInListAction)* Optional. Action to be taken if *ListValue* is already in *ListName*. If omitted, a default message displays if the value is already in the list.










**Example.** This example checks Condition Indicator values to be sure that they are not duplicated within the segment. As each Condition Indicator value is encountered in the EDI file, a rule checks to see if it is in a list called 2300CRCconditionInd. If so, a custom error message is issued. If not, it is added to the list.

What Rule to Run

InList  

Text 2300CRCconditionInd Current\_Element (BusinessRules.Utilities DisplayErrorByNumber 32216)

Message 32216 is a custom message in file CustomerFSBRERRS.txt. The format of `DisplayErrorByNumber` is different for rules used by Analyzer.

|   |   |   |                                   |
|---|---|---|-----------------------------------|
|  |  | 220 : CRC   | Ambulance Certification           |
|   |  |  01 : 1136 | Code Category                     |
|   |  |  02 : 1073 | Yes/No Condition or Response Code |
|   |  |  03 : 1321 | Condition Indicator               |
|   |  |  04 : 1321 | Condition Indicator               |
|   |  |  05 : 1321 | Condition Indicator               |
|   |  |  06 : 1321 | Condition Indicator               |
|   |  |  07 : 1321 | Condition Indicator               |

## ListCheck

---

### All validation programs

---

Checks for the presence of a certain value in a list and takes action based in whether it is found in the list.

#### Format of Parameters

*ListName ListValue Operand (IfTrueAction)*

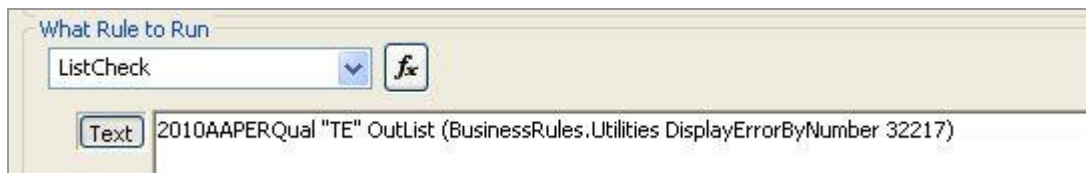
Where:

|                       |  |
|-----------------------|--|
| <i>ListName</i>       | Name of the list to be checked.  |
| <i>ListValue</i>      | The value that may or may not be in the list. This can be a variable, literal value in double quotes, or <code>Current_Element</code> .  |
| <i>Operand</i>        | Either <b>InList</b> (indicating the value is in the list) or <b>OutList</b> (indicating the value is not in the list). These are case-sensitive.  |
| <i>(IfTrueAction)</i> | Optional. Action to be taken if the test is true: If the operand is InList, this action will be taken only if the value is in the list. If the operand is OutList, this action will be taken only if the value is not in the list. If omitted, a general message is displayed. |

#### Example

*ListName:* 2010AAPERQual  
*ListValue:* TE  
*Operand* OutList  
*(IfTrueAction)* (BusinessRules.Utilities DisplayErrorByNumber 32217)

This example issues a message if the data in the PER segment does not include a telephone number. We used ListInsert to create a list containing the values in each Communication Number Qualifier in the segment, and then use ListCheck to issue an error message if the list does not contain the list value "TE".



Message 32217 is a custom message in file CustomerFSBRERRS.txt.












The format of DisplayErrorByNumber is different for rules used by Analyzer.

*ClearList of the 2010AAPERQual list.*

*Use ListInsert to record values of each Communication Number Qualifier in 2010AAPERQual list.*

*Use ListCheck to see if 2010AAPERQual contains "TE".*

|   |                |  |
|---|----------------|--|
|  | <b>045 PER</b> | <b>Submitter EDI Contact Information</b> |
|  | PER01          | 366 Contact Function Code                |
|  | PER02          | 93 Name                                  |
|  | PER03          | 365 Communication Number Qualifier       |
|  | PER04          | 364 Communication Number                 |
|  | PER05          | 365 Communication Number Qualifier       |
|  | PER06          | 364 Communication Number                 |
|  | PER07          | 365 Communication Number Qualifier       |
|  | PER08          | 364 Communication Number                 |

## ListContig

---

### All validation programs except Analyzer

---

Check whether a list contains a contiguous block of dates or integers. Before using ListContig, the list already has to be defined.

They do not have to be in any order, and representations of the same number are acceptable unless you use both the D and U parameters. Here are some examples:

| Contents of list  | Contiguous?                                   | Reason   |
|---|---|--|
| 1, 2, 3, 4, 5, 6, 7, 8  | Yes   |  |
|   | No  | Empty lists are not considered contiguous  |
| 1, 2, 4, 5, 7, 8, 10, 55  | No  |  |
| 2, 4, 6, 5, 3, 8, 7, 1  | Yes   | Order does not matter  |
| 1, 01, 2, 3, 4, 5, 6, 06, 7, 8  | Yes   | Representation of the same number are allowed in non-date lists  |
| 20071231, 20080101  | Yes, with "D" option<br>No without "D" option | See below for additional information on the D parameter  |
| 20071230, 20080101  | No  | Missing 20071231   |
| 20071201-20071231, 20080101-20080131  | Yes with "D" option                           |  |
| 01/01/2008-03/31/2008, 02/01/2008-04/15/2008, 04/15/2008, 04/16/2008-04/30/2008 | Yes, with D option but not U option           | The dates cover the contiguous period 01/01/2008 through 04/30/2008 and overlapping is OK                                  |
| 01/01/2008-03/31/2008, 02/01/2008-04/15/2008, 04/15/2008, 04/16/2008-04/30/2008 | No, with D and U option                       | The dates/ranges overlap:<br>01/01/2008-03/31/2008 overlaps 02/01/2008-04/15/2008<br>04/15/2008 is in the first two ranges |

### Format of Parameters

*ListName* *ResultVar* (D) (U)

Where:

*ListName*                      Name of the list to be checked.

*ResultVar*                      Name of the variable where the result is to be stored. It will be 1 if the list is contiguous, or 0 if not.

**D** Optional. D causes ListContig to consider the values in the list as dates, and also enables ListContig to recognize date ranges. Any hours, minutes, or seconds, in the dates are ignored. Without the D option, ListContig treats the values as integers. This parameter must be a constant within double-quotes.

**U** Optional. It requires the D option and must immediately follow the D with no space, like this:

Mydatelist DateResults DU

U (which stands for Unique) checks to see if the dates and date ranges are unique. Dates must be unique and contiguous to get a 1 in ResultVar.

This parameter must be a constant within double-quotes.

For example, if the list contains the following dates (in human readable form, for easy consumption):

01/01/2008-03/31/2008  
02/01/2008-04/15/2008  
04/15/2008  
04/16/2008-04/30/2008

ListContig without the U would pass, because the dates cover the contiguous period 01/01/2008 through 04/30/2008.

ListContig with the U would fail because the list contains overlapping dates/ranges: 01/01/2008-03/31/2008 overlaps 02/01/2008-04/15/2008 and 04/15/2008 is in the first two ranges.

**Example.** We have a list of Invoice Numbers called InvcList. We check to make sure that there are no gaps in the invoices received:

| Server                  | Function       | Parameter   |
|-------------------------|----------------|---|
| BusinessRules.Lists     | ListContig     | InvcList InvcContig   |
| BusinessRules.Utilities | CompareNumeric | InvcContig EQ 0<br>(BusinessRules.Utilities.DisplayError ByNumber 0 0<br>"Missing at least one Invoice Number") |

We first check list InvcList with the ListContig function to determine whether it contains a contiguous list of numbers. The result goes into variable InvcContig, which will be either a **1** if InvcList is contiguous, or **0** if it is not contiguous. We then check InvcContig for **0** and display an error message if true.

## ListCount

---

### All validation programs

---

Reports the number of entries in a list. Before using ListCount, the list already has to be defined and in use.

#### Format of Parameters

*ListName* *ResultVar*

Where:

*ListName*                      Name of the list, which already exists.

*ResultVar*                      Name of the variable that is to contain the count. If the variable does not exist, it is created.

**Example.** This example is from loop 2010AA REF01 in a HIPAA 837I. The REF01 can occur up to 8 times. We want to know how many were used, because we only allow 5 and we want to make sure that the Qualifier 1B was the first occurrence.

Rule 1 on REF01: Add the contents of the REF01 to the list.

What Rule to Run

ListInsert

Text 2010AAREFList Current\_Element

Rule 2 on REF01: Count the number of list entries and put them in variable ProvQualifier.

What Rule to Run

ListCount

Text 2010AAREFList ProvQualifier

Rule 3 on REF01: Display an error message if the count exceeds 5.

What Rule to Run

CompareNumeric

Text ProvQualifier GT "5" (BusinessRules.Utilities DisplayErrorByNumber 0 0 "We only process up to 5 different provider qualifiers.")

Rules 4 and 5 on REF01: Check the first entry in the list.

Put the first entry from the list into a variable.

What Rule to Run

ListGetVar

Text 2010AAREFList "1" FirstQualifier

Check the variable and issue a message if it is not 1B:

What Rule to Run

CompareString

Text FirstQualifier NE "1B" (BusinessRules.Utilities DisplayErrorByNumber 0 0 "We expect the first qualifier to be 1B for B5 provider number.")

## ListGetVar

---

### All validation programs

---

Places entry *n* from a list into a variable.

#### Format of Parameters

*ListName ListEntry ResultVar*

Where:

|                  |   |
|------------------|---|
| <i>ListName</i>  | Name of the list.   |
| <i>ListEntry</i> | Position of the entry in the list – an integer in double quotes or a variable that contains an integer. |
| <i>ResultVar</i> | A variable that is to contain the value from the list. If the variable does not exist, it is created.   |

**Example.** This example takes the first value in the list Reissue1 and places it in a variable called ReissueAnswer.

What Rule to Run

ListGetVar

Text Reissue1 "1" ReissueAnswer

## ListInsert

---

### All validation programs

---

Adds one or more values to a list. If the values are already in the list, they are not added again.

#### Format of Parameters

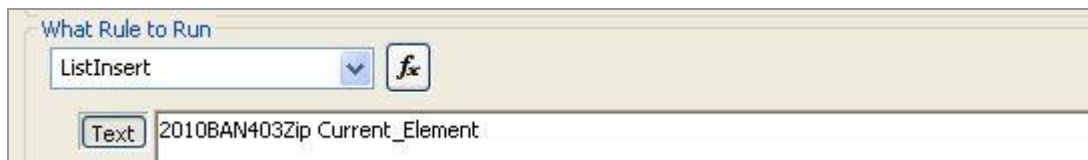
*ListName ListValue <ListValue ...>*

Where:

|                  |  |
|------------------|--|
| <i>ListName</i>  | Name of the list. If the list does not exist, it is created.   |
| <i>ListValue</i> | Optional. Value to be added to the list. Variable, literal in double quotes, or <code>Current_Element</code> . If omitted, <code>Current_Element</code> is used.<br><br>To add multiple values, separate each with a space.<br><br><b>Important:</b> The maximum total length of all values is 4000 characters. If you have more than that, do another <code>ListInsert</code> to the same list. |

#### Examples

This example inserts the value of the current element into list 2010BAN403Zip. Subsequent repetitions of the loop would add more zip codes to the list.

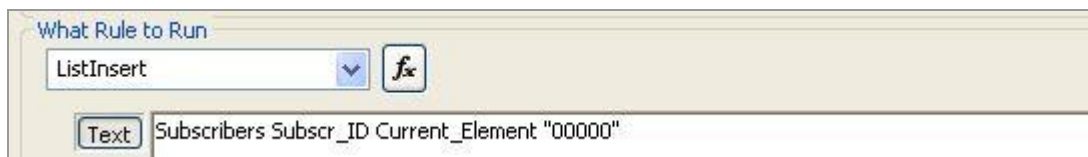


What Rule to Run

ListInsert

Text 2010BAN403Zip Current\_Element

This example inserts the value of the variable `Subscr_ID`, the value of the current element, and the literal value 00000 into a list called `Subscribers`.



What Rule to Run

ListInsert

Text Subscribers Subscr\_ID Current\_Element "00000"

## ListMinMax

---

### All validation products except Analyzer

---

Acquires this information about a list, which has been set up with other List rules:

- Minimum and maximum values in the list.

- Positions in the list of the minimum and maximum values.
- Count of the number of objects in the list, or number of days spanned by the lists members.

### Format of Parameters

*ListName MinResultVar MaxResultVar <CountResultVar MinPosResultVar  
MaxPosResultVar <Options>>*

Where:

|                        |   |
|------------------------|---|
| <i>ListName</i>        | Name of the list, which already exists.   |
| <i>MinResultVar</i>    | Variable where the list's minimum value is to be stored.  |
| <i>MaxResultVar</i>    | Variable where the list's maximum value is to be stored.  |
| <i>CountResultVar</i>  | Optional. Variable where the list's object count is to be stored.<br><br>For date lists (see Options below), this is the number of days spanned by the lists members.<br><br>For other lists, this is the number of members in the list.  |
| <i>MinPosResultVar</i> | Optional. Variable that contains the name of the variable where the position of the list's minimum value is to be stored. Requires use of <i>CountResultVar</i> .   |
| <i>MaxPosResultVar</i> | Optional. Variable that contains the name of the variable where the position of the list's maximum value is to be stored. Requires use of <i>CountResultVar</i> and <i>MinPosResultVar</i> .  |
| <i>Options</i>         | Optional. One of these:<br><br>"D" causes <b>ListMinMax</b> to consider the values in the list as dates, and lets ListMinMax recognize date ranges. Any hours, minutes, or seconds, in the dates are ignored. It will also change the way the object count is determines (see <i>CountResultVar</i> above). Include quotes around "D".<br><br>"S" causes <b>ListMinMax</b> to treat <i>ListName</i> as a list of strings.<br><br>Without any option, the values are considered integers. Include quotes around "S".<br><br>If you use options, include <i>CountResultVar</i> , <i>MinPosResultVar</i> , and <i>MaxPosResultVar</i> to maintain positions. |

### Example 1. List of integers.

With no Options (defaults to Integer):

What Rule to Run

ListInsert  ☐ Look-Ahead Rule

| Text | Parameter Name | Parameter Value          |
|------|----------------|--------------------------|
|      | ListName       | NList                    |
|      | Value          | "1" "4" "6" "10" "9" "2" |

What Rule to Run

ListMinMax  ☐ Look-Ahead Rule

| Text | Parameter Name  | Parameter Value |
|------|-----------------|-----------------|
|      | ListName        | NList           |
|      | MinResultVar    | VMin            |
|      | MaxResultVar    | VMax            |
|      | CountResultVar  | VCount          |
|      | MinPosResultVar | VMinPos         |
|      | MaxPosResultVar | VMaxPos         |
|      | Options         |                 |

Results:

|         |      |                       |
|---------|------|-----------------------|
| VMin    | = 1  |                       |
| VMax    | = 10 |                       |
| VCount  | = 6  | (members)             |
| VMinPos | = 1  | (first item in list)  |
| VMaxPos | = 4  | (fourth item in list) |

With Options equal to S:

What Rule to Run

ListMinMax  ☐ Look-Ahead Rule

| Text | Parameter Name  | Parameter Value |
|------|-----------------|-----------------|
|      | ListName        | NList           |
|      | MinResultVar    | VMin            |
|      | MaxResultVar    | VMax            |
|      | CountResultVar  | VCount          |
|      | MinPosResultVar | VMinPos         |
|      | MaxPosResultVar | VMaxPos         |
|      | Options         | "S"             |

Results:

|         |     |                                  |
|---------|-----|----------------------------------|
| VMin    | = 1 |                                  |
| VMax    | = 9 | (with strings, 9 comes after 10) |
| VCount  | = 6 | (members)                        |
| VMinPos | = 1 | (first item in list)             |
| VMaxPos | = 5 | (fifth item in list)             |

### Example 2. List of strings.

Assume that NameList = Greig, Beth, Cindy, Dorrie, Woody, Norman

ListMinMax NList VMin VMax VCount VMinPos VMaxPos



|          |            |  |
|----------|------------|--|
| Results: | VMin       | = Beth                                 |
|          | VMax       | = Woody                                |
|          | VCount     | = 6 (members)                          |
|          | VMinPos    | = 2 (second item in list)              |
|          | VMaxPos    | = 5 (fifth item in list)               |
|          | PeriodList | = 20071201, 20080101, 2007010120070331 |

**Example 3.** List of dates, including ranges with and without hyphens.

Assume that PeriodList = 20071201-20071208, 20080101, 2007010120070331

```
ListMinMax PeriodList VMin VMax VCount VMinPos VMaxPos "D"
```

|          |         |                           |
|----------|---------|---------------------------|
| Results: | VMin    | = 20070101                |
|          | VMax    | = 20080101                |
|          | VCount  | = 99 (Days)               |
|          | VMinPos | = 3 (third item in list)  |
|          | VMaxPos | = 2 (second item in list) |

```
ListMinMax PeriodList VMin VMax VCount VMinPos VMaxPos "S"
```

|          |         |                           |
|----------|---------|---------------------------|
| Results: | VMin    | = 2007010120070331        |
|          | VMax    | = 20080101                |
|          | VCount  | = 3 (members)             |
|          | VMinPos | = 3 (third item in list)  |
|          | VMaxPos | = 2 (second item in list) |

```
ListMinMax PeriodList Placeholder1 Placeholder2 Placeholder3 VMinPos "S"
```

We don't care about the min, max, and count but we need them in the parameters as placeholders. We actually name them Placeholder1, etc., to clarify that we aren't using them. We omit the last parameter (the maximum value) since it is at the end and isn't needed as a placeholder.

|          |              |                          |
|----------|--------------|--------------------------|
| Results: | Placeholder1 | = 2007010120070331       |
|          | Placeholder2 | = 20080101               |
|          | Placeholder3 | = 3 (Members)            |
|          | VMinPos      | = 3 (Third item in list) |

# Lookahead Business Rules

---

## All validation programs except Analyzer

---

Lookahead is a way to pre-scan a defined section of the data, execute only Lookahead business rules, and then return to the beginning of the range to start validation. The purpose is usually to grab information that is farther down in the transaction.

The main steps for Lookahead are:

1. Mark the Lookahead range(s)
2. Create Lookahead business rules
3. Create one or more regular business rules to use the Lookahead information

---

## Rules will execute in this order

---

1. Rules before the Lookahead range will execute as usual.
2. When the Lookahead range is reached, the lookahead rules will execute until the end of the range.
3. Regular business rules will execute, starting at the top of the range and continuing as usual.
4. If a second Lookahead range is encountered, its lookahead rules will execute until the end of that range.
5. Regular business rules will execute, starting at the top of the second range.

### Example 1. Simple Lookahead scenario (range is enclosed in brackets)

|                   |                                   |
|-------------------|-----------------------------------|
| ST                | Rule A                            |
| Seg1              | Rule B                            |
| Seg2              |                                   |
| Seg3              |                                   |
| Lookahead range { | LoopA – max repeat is 1           |
|                   | Seg4 Rule C – lookahead<br>Rule D |
|                   | Seg5 Rule E – lookahead<br>Rule F |
|                   | End LoopA                         |
|                   | LoopB – max repeat is 1           |
|                   | Seg6 Rule G                       |
|                   | Seg7                              |
|                   | End LoopB                         |
|                   | Seg 8 Rule H                      |

Rules will execute in this order: A , B , C , E , D , F , G , H

### Example 2. Lookahead range is a repeating loop

|                 |                         |                              |
|-----------------|-------------------------|------------------------------|
|                 | ST                      | Rule A                       |
|                 | Seg1                    | Rule B                       |
|                 | Seg2                    |                              |
|                 | Seg3                    |                              |
| Lookahead range | LoopA – max repeat is 2 |                              |
|                 | Seg4                    | Rule C – lookahead<br>Rule D |
|                 | Seg5                    | Rule E – lookahead<br>Rule F |
|                 | End LoopA               |                              |
|                 | LoopB – max repeat is 1 |                              |
|                 | Seg6                    | Rule G                       |
|                 | Seg7                    |                              |
|                 | End LoopB               |                              |
|                 | Seg 8                   | Rule H                       |

Rules will execute in this order: A , B , C , E , C , E , D , F , D , F , G , H

### Example 3. Nested loops in Lookahead range

|                 |                         |                              |
|-----------------|-------------------------|------------------------------|
|                 | ST                      | Rule A                       |
|                 | Seg1                    | Rule B                       |
|                 | Seg2                    |                              |
|                 | Seg3                    |                              |
| Lookahead range | LoopA – max repeat is 2 |                              |
|                 | Seg4                    | Rule C – lookahead<br>Rule D |
|                 | Seg5                    | Rule E – lookahead<br>Rule F |
|                 | LoopB - max repeat is 2 |                              |
|                 | Seg 6                   | Rule G – lookahead           |
|                 | Seg 7                   | Rule H                       |
|                 | End LoopB               |                              |
|                 | Seg 9                   | Rule I - lookahead           |
|                 | Seg 10                  | Rule J                       |
|                 | End LoopA               |                              |
|                 | Seg 11                  | Rule K                       |

Rules will execute in this order:

A , B , C , E , G , G , I , C , E , G , G , I , D , F , H , H , J , D , F , H , H , J , K

#### Example 4. Two Lookahead ranges

|                     |                         |                              |
|---------------------|-------------------------|------------------------------|
| Lookahead<br>ranges | ST                      | Rule A                       |
|                     | Seg1                    | Rule B                       |
|                     | Seg2                    |                              |
|                     | Seg3                    |                              |
|                     | LoopA – max repeat is 2 |                              |
|                     | Seg4                    | Rule C – lookahead<br>Rule D |
|                     | Seg5                    | Rule E – lookahead<br>Rule F |
|                     | End LoopA               |                              |
|                     | LoopB – max repeat is 2 |                              |
|                     | Seg6                    | Rule G – lookahead           |
|                     | Seg7                    | Rule H                       |
|                     | End LoopB               |                              |
|                     | Seg 8                   | Rule I                       |

Rules will execute in this order:

A, B, C, E, C, E, D, F, D, F, G, G, H, H, I

#### Example 5. End of loop rules

|                    |                         |  |
|--------------------|-------------------------|--|
| Lookahead<br>range | ST                      | Rule A – SetLoopPostInstanceExit <b>lookahead</b> rule<br>Rule B – SetLoopPostExecute rule |
|                    | Seg1                    | Rule C   |
|                    | Seg2                    |  |
|                    | Seg3                    |  |
|                    | LoopA – max repeat is 2 |  |
|                    | Seg4                    | Rule D – lookahead<br>Rule E   |
|                    | Seg5                    | Rule F – lookahead<br>Rule G   |
|                    | End loop                |  |
|                    | Seg 6                   | Rule H   |

Rules will execute in this order:

C, D, F, A, D, F, A, E, G, E, G, B, H

### Example 6. End of loop rules with nested loops and two Lookahead ranges

|                         |   |
|-------------------------|---|
| ST                      | Rule A – SetLoopPostInstanceExit <b>lookahead</b> rule on Loop A (Ignored – outside of range)<br>Rule B – SetLoopPostInstanceExit <b>lookahead</b> rule on Loop B |
| Seg1                    | Rule C  |
| Seg2                    |   |
| Seg3                    |   |
| LoopA – max repeat is 2 |   |
| Seg4                    | Rule D – lookahead<br>Rule E  |
| Seg5                    | Rule F – lookahead<br>Rule G  |
| LoopB - max repeat is 2 |   |
| Seg 6                   | Rule H – lookahead  |
| Seg 7                   | Rule I  |
| End loop B              |   |
| Seg 9                   | Rule J – lookahead (ignored – outside of range)   |
| Seg 10                  | Rule K  |
| End loopA               |   |
| Seg 11                  | Rule L  |

LoopB is nested within LoopA, but new Lookahead range starts here, ending the one on LoopA

Rules will execute in this order:

C, D, F, E, G, H, B, I, H, B, I, K, D, F, E, G, H, B, I, H, B, I, K, L

### Typical example

See if a subscriber was covered on the claim service dates. If not, display an error message at the subscriber ID location (which is earlier in the transaction than the dates).

The pertinent parts of validation:

1. After reaching the Lookahead start range on the 2000B, Instream scans through the range and executes two Lookahead business rules on the service date. These capture the oldest and newest service dates.
2. Instream then resumes normal validation at the 2000B. A rule captures the subscriber ID in the NM109.
3. Also on the NM109, an InvokeWebService business rule checks the subscriber ID, oldest service date, and newest service date against a database that records when this subscriber was covered. It sends back a Y if they were covered and an N if not.
4. A rule then displays an error message on the NM109 if the returned value was N.

### Demo

Please see Appendix J: LookAhead and Array Extended Example on page 295 for a complete example.

## Marking a Lookahead Range

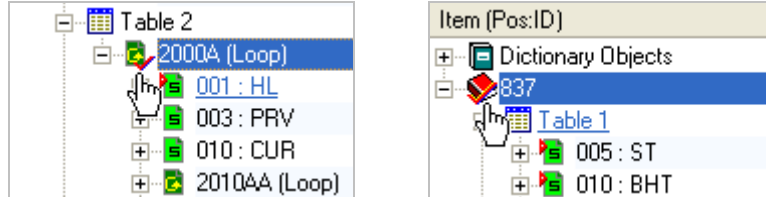
Things to know:

- There can be more than one Lookahead ranges in a guideline.
- When Instream detects the start of the Lookahead range, it goes into a mode where it scans through the range and executes only the Lookahead rules. When the range ends, it then returns to the start of the range and continues normal execution.
- For speediest validation, make the range only as large as necessary.
- To monitor Lookahead ranges by seeing where they stop and start, edit your .apf file (default file `/bin/$fsdefault.apf`). Turn on messages 17037 and 17038 (i.e., set them to a number greater than 0). Message 17037 indicates where the Lookahead range begins and 17038 indicates where it ends.

### Setting a Starting Point

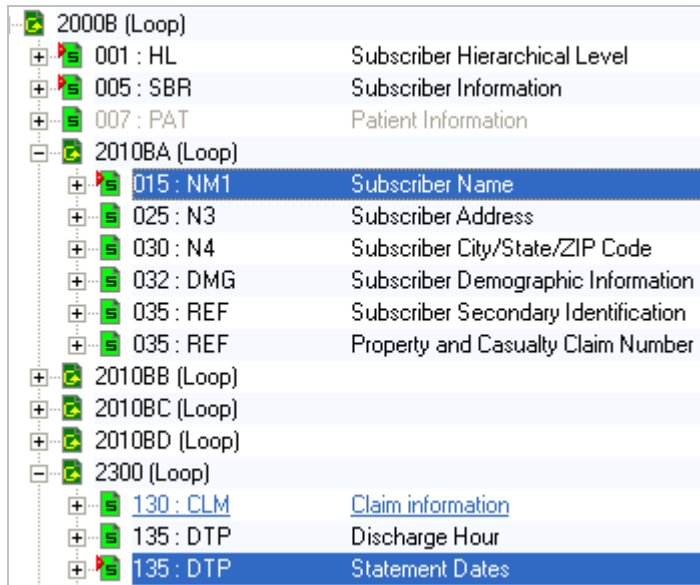
1. Open the guideline in the EDISIM Standards Editor.
2. Decide where the Lookahead range starts.

The lookahead range starts on a loop header, or on the transaction line at the top of the guideline. This must be on the parent loop of everything that is involved, including the location of the Lookahead rule and the location where the data is found.



## Example

If they were not a subscriber on the Statement Dates, you want to display a message (earlier) on the Subscriber Name NM1.



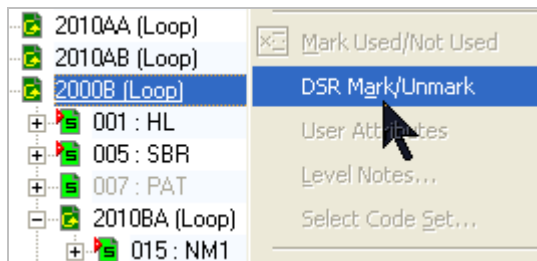
|               |                                     |
|---------------|-------------------------------------|
| 2000B (Loop)  |                                     |
| 001 : HL      | Subscriber Hierarchical Level       |
| 005 : SBR     | Subscriber Information              |
| 007 : PAT     | Patient Information                 |
| 2010BA (Loop) |                                     |
| 015 : NM1     | Subscriber Name                     |
| 025 : N3      | Subscriber Address                  |
| 030 : N4      | Subscriber City/State/ZIP Code      |
| 032 : DMG     | Subscriber Demographic Information  |
| 035 : REF     | Subscriber Secondary Identification |
| 035 : REF     | Property and Casualty Claim Number  |
| 2010BB (Loop) |                                     |
| 2010BC (Loop) |                                     |
| 2010BD (Loop) |                                     |
| 2300 (Loop)   |                                     |
| 130 : CLM     | Claim information                   |
| 135 : DTP     | Discharge Hour                      |
| 135 : DTP     | Statement Dates                     |

The starting point will be the 2000B loop, which is the parent loop of both the Subscriber Name NM1 and the Statement Dates DPT.

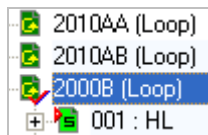
(The starting point cannot be the 2010BA loop since it is not the parent to the 2300 loop – even though it appears ahead of it in the guideline.)

3. Put a DSR mark at the top of the range.

Right click on the loop and select **DSR/Unmark**.



A red check now marks the range's start:



|               |
|---------------|
| 20104A (Loop) |
| 20104B (Loop) |
| 2000B (Loop)  |
| 001 : HL      |


## Ending a Lookahead Range

Lookahead ranges end in these ways:



- The range started on a loop, and the loop ends.
- Another Lookahead range starts. This automatically ends any previous range.
- An ExitLookahead business rule is encountered.

## Example



To stop Lookahead after it gets the value from the Statement Date DTP in the 2300 loop, add this to the DTP03:

| What Rule to Run |                |   |   |
|------------------|----------------|---|---|
| ExitLookahead    |                |  | <input checked="" type="checkbox"/> Look-Ahead Rule |
| Text             | Parameter Name | Parameter Value   | LA  |
|                  |                |   |   |

If the ending item might not be in the data, use one of these methods:

| What Rule to Run        |                |   |   |
|-------------------------|----------------|---|---|
| SetLoopPostInstanceExit |                |  | <input checked="" type="checkbox"/> Look-Ahead Rule   |
| Text                    | Parameter Name | Parameter Value   | LA  |
|                         | loopID         | 2010BA  |   |
|                         | Rule           | ExitLookahead   |  <input checked="" type="checkbox"/> |

or

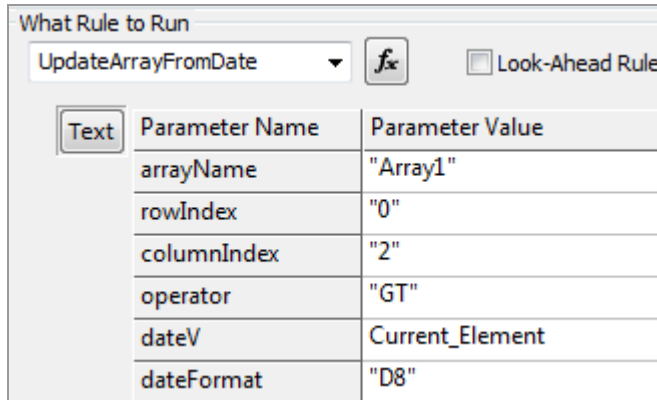
| What Rule to Run |                |   |   |
|------------------|----------------|---|---|
| RunNoData        |                |  | <input checked="" type="checkbox"/> Look-Ahead Rule   |
| Text             | Parameter Name | Parameter Value   | LA  |
|                  | Rule           | ExitLookahead   |  <input checked="" type="checkbox"/> |



## Creating Lookahead Business Rules

Lookahead business rules are different than all other business rules in that they can actually execute any other rule's functions.

This example shows the Array server updating an array from the current location:



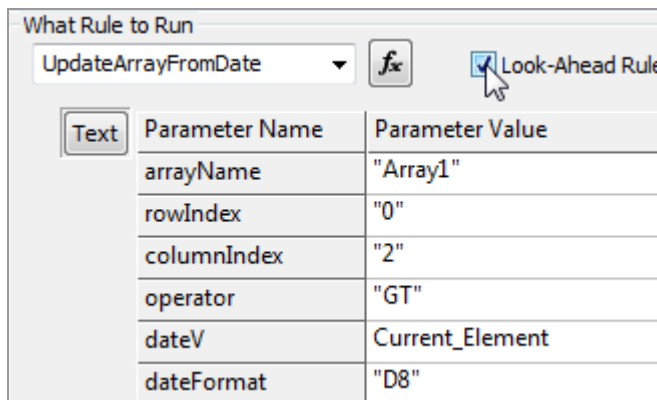
What Rule to Run

UpdateArrayFromDate  ☐ Look-Ahead Rule

Text

| Parameter Name | Parameter Value |
|----------------|-----------------|
| arrayName      | "Array1"        |
| rowIndex       | "0"             |
| columnIndex    | "2"             |
| operator       | "GT"            |
| dateV          | Current_Element |
| dateFormat     | "D8"            |

By simply checking the Look-Ahead Rule box, this same rule now executes when the Lookahead range is first encountered.



What Rule to Run

UpdateArrayFromDate  ☒ Look-Ahead Rule

Text

| Parameter Name | Parameter Value |
|----------------|-----------------|
| arrayName      | "Array1"        |
| rowIndex       | "0"             |
| columnIndex    | "2"             |
| operator       | "GT"            |
| dateV          | Current_Element |
| dateFormat     | "D8"            |

This should be attached to an item in the Lookahead range (see page 127). You can type the desired function into the Function field if it does not appear in the drop-down list.

The difference between these two rules is when they execute:




- The first example executes when it is encountered during normal validation.
- The Lookahead rule executes earlier. When validation encounters a Lookahead starting point, it scans through the whole range to find and execute any Lookahead business rules. It then returns to the range start and validates normally.

## Lookahead in Exit rules


---

Check the Lookahead box in the parameters area.

Correct:

|   |                |   |   |                                     |
|---|----------------|---|---|-------------------------------------|
| What Rule to Run  |                |   |   |                                     |
| SetLoopPostInstanceExit   |                |  | <input checked="" type="checkbox"/> Look-Ahead Rule                                 |                                     |
|  | Parameter Name | Parameter Value   |   | LA                                  |
|   | loopID         | 2000B   |   |                                     |
|   | Rule           | DumpArray   |  | <input checked="" type="checkbox"/> |
|   | > arrayName    | SubscriberArray   |   |                                     |

Incorrect:



|   |   |
|---|---|
| What Rule to Run                                    |   |
| * Call External Routine                             |  |
| <input checked="" type="checkbox"/> Look-Ahead Rule |   |
| Server:   | BusinessRules.Lookahead   |
| Function:   | DumpArray   |
| Parameters:   | 2000B BusinessRules.Array DumpArray "SubscriberArray"                             |

## Lookahead example



---

Put the Lookahead start point on an 837I 2000B loop.

On the ST, create an array called Array1.

|   |   |                   |
|---|---|-------------------|
| What Rule to Run  |   |                   |
| CreateArray   |  |                   |
| <input type="checkbox"/> Look-Ahead Rule  |   |                   |
|  | Parameter Name  | Parameter Value   |
|   | arrayName   | "SubscriberArray" |

On the Subscriber NM109, add the subscriber identification code to the first cell in Array1.

|   |   |                   |
|---|---|-------------------|
| What Rule to Run  |   |                   |
| SetArrayFromVar   |  |                   |
| <input type="checkbox"/> Look-Ahead Rule  |   |                   |
|  | Parameter Name  | Parameter Value   |
|   | arrayName   | "SubscriberArray" |
|   | rowIndex  | "0"               |
|   | columnIndex   | "0"               |
|   | resultVar   | Current_Element   |

On the date element (2400 DTP03 Service Line Date), add the service line date to Array1 with a Lookahead rule. If there are multiple service lines, this will store only the oldest date in cell 0,1. If cell 0,1 is empty, the rule simply inserts the element's date into it.

What Rule to Run

UpdateArrayFromDate  ☒ Look-Ahead Rule

| Text | Parameter Name | Parameter Value   |
|------|----------------|-------------------|
|      | arrayName      | "SubscriberArray" |
|      | rowIndex       | "0"               |
|      | columnIndex    | "1"               |
|      | operator       | "LT"              |
|      | dateV          | Current_Element   |
|      | dateFormat     | "D8"              |

Likewise, put the most recent service line date into cell 0,2.

What Rule to Run

UpdateArrayFromDate  ☒ Look-Ahead Rule

| Text | Parameter Name | Parameter Value   |
|------|----------------|-------------------|
|      | arrayName      | "SubscriberArray" |
|      | rowIndex       | "0"               |
|      | columnIndex    | "2"               |
|      | operator       | "GT"              |
|      | dateV          | Current_Element   |
|      | dateFormat     | "D8"              |

Now, back at the Subscriber NM109, use InvokeWebService to check the ID and dates against a database. Assume that the web service has been preconfigured to send back a Y or N in cell 0,4 in an array called Array1BACK:.


What Rule to Run

InvokeWebService  ☐ Look-Ahead Rule

| Text | Parameter Name  | Parameter Value       |
|------|-----------------|-----------------------|
|      | WSName          | CheckSub              |
|      | inputArrayName  | "SubscriberArray"     |
|      | outputArrayName | "SubscriberArrayBack" |
|      | failRule        |                       |

The final task is to display an error message if cell 0,4 does not contain Y:

What Rule to Run

CheckVarFromArray  ☐ Look-Ahead Rule

**Text**

| Parameter Name       | Parameter Value   |
|----------------------|---|
| arrayName            | "SubscriberArrayBack"                                     |
| rowIndex             | "0"   |
| columnIndex          | "4"   |
| valueA               | "Y"   |
| elseRule             | DisplayErrorByNumber                                      |
| > ErrorNumber        | 0   |
| > Severity           | 0   |
| > <b>MessageText</b> | "Subscriber was not insured on one or more service dates" |

Business rules for the NM109 must be in this order:

Business Rules

1. Always call External Routine BusinessRules.Array.SetArrayFromVar "SubscriberArray" "0" "0" Current\_Element
2. Always call External Routine BusinessRules.DBServer.InvokeWebService CheckSubscriber "SubscriberArray" "SubscriberArrayBack"
3. Always call External Routine BusinessRules.Array.CheckVarFromArray "SubscriberArrayBack" "0" "4" "Y" (BusinessRules.Utilities.DisplayError

# Looping Business Rules

---

## All validation programs except Analyzer

---

The Looping rules let you repeatedly execute a group of rules:

- The **ForEach** function indicates the start of a loop.
- The **Next** function identifies the end of a loop, with the rules between ForEach and Next repeatedly executing. When all members of the list are processed, execution continues with the function following the Next function.
- **ExitLoop** causes the ForEach loop to immediately exit, with execution resuming with the rule after the Next function.

Loops can be nested, but cannot span objects. For example, you can't start a loop on one element and end it on another.

If the list is empty, none of the looping rules execute.

## ForEach

---

### All validation programs except Analyzer

---

Marks the top of a set of rules that execute once for each member of a list.

#### Format of Parameters

*MemberVar* IN *ListName*

Where:

*MemberVar*                      Name of the variable to contain each member of the list.

IN                                  Literal text.

*ListName*                        Name of the list to be processed.

#### Example

Let's say we have a list of Supplier IDs called SuppList. These rules will display each of them:

| Server                  | Function             | Parameter                |
|-------------------------|----------------------|--------------------------|
| BusinessRules.Looping   | ForEach              | SID IN SuppList          |
| BusinessRules.Utilities | DisplayErrorByNumber | 0 0 "Supplier ID: %SID%" |
| BusinessRules.Looping   | Next                 |                          |

One at a time, **ForEach** puts each member of SuppList into the SID variable, and then begin executing rules until it hits the **Next** function. Note that we are using %SID% in the error message to include the current contents of the SID variable (i.e., the current SuppList member) into the error message. (See Preprocessor Variables on page [245](#)).

The **Next** function causes **ForEach** to load the next SuppList member into SID and repeat execution of the loop.

Once the last member of SuppList is processed, execution will continue with the function following the **Next** function. If SuppList contains these members: A1001, B2002, Z99, then the output would be:

Supplier ID: A1001

Supplier ID: B2002

Supplier ID: Z99

## Next

---

### All validation programs except Analyzer

---

Marks the end of a **ForEach** loop. See **ForEach** on page [135](#) for further explanation.

#### Format of Parameters

Next has no parameters.

## ExitLoop

---

### All validation programs except Analyzer

---

Causes the **ForEach** loop to immediately exit, with execution resuming with the rule after the Next function. See **ForEach** on page [135](#) for further explanation of the looping concept.

#### Format of Parameters

ExitLoop has no parameters.

#### Example

See Extended Looping Example on page [137](#).

## Extended Looping Example

We have a list of Supplier IDs called SuppList.

We have accumulated the total amount invoiced by Supplier ID into a variable array called InvcTotals.

We want to make sure that each Supplier invoiced something greater than zero. If this is not the case, we want to display an error message, just once (not one per supplier), saying that at least one supplier didn't have invoices.

| Server                  | Function       | Parameter  |
|-------------------------|----------------|--|
| BusinessRules. Variable | SetVar         | VLCCount "0"<br><i>(Set up variable VLCCount to contain a count of the suppliers that do have an invoice total greater than zero.)</i>   |
| BusinessRules. Looping  | ForEach        | SID IN SuppList<br><i>(Go through each entry in SuppList.)</i>   |
| BusinessRules. Variable | CompareNumeric | InvcTotals(SID) LE "0" (BusinessRules.Looping.ExitLoop)<br><i>(If the InvcTotal entry is less than or equal to zero, exit the loop with the ExitLoop command. Execution continues with the ListCount rule after the Next rule.)</i>  |
| BusinessRules. Variable | AddVar         | VLCCount "1"<br><i>(If the InvcTotal entry is greater than zero, we increment the VLCCount counter.)</i>   |
| BusinessRules. Looping  | Next           | <i>(Repeat the loop for the next SuppList member.)</i>   |
| BusinessRules. Lists    | ListCount      | SuppList VSuppListCount<br><i>(After the looping completes, we put the number of members in SuppList into a variable called VSuppListCount.)</i>   |
| BusinessRules. Variable | CompareNumeric | VLCCount NE VSuppListCount<br>(BusinessRules.Utilities. DisplayErrorByNumber 0 0 "At least one Supplier had no Invoice Total"<br><i>(We then compare VSuppListCount against VLCCount; if all suppliers had invoice totals greater than zero, they should match. If not, then we display an error message.)</i> |



# ODBC Business Rules

---

## Windows Instream and HIPAA Validator Desktop

---

You can validate data against your own databases using ODBC. The capabilities include:

- Testing for the existence of a particular record in a database table.
- Retrieval of one or more fields from a record in a database table.
- Execution of a stored procedure in the database.

### ODBC lookup or Customer Code Tables?

Use ODBC to access existing databases. If you are creating a new table just for Instream, then set up a code table (see page [261](#)) – which will give you faster lookup.

For example, if you already have a SQL ODBC-accessible database containing information about 500,000 health care policies that you wish to validate against, then an ODBC query makes sense.

The alternative would be to export data from this database into a text table and distribute that on some schedule, a process that comes with its own coordination and distribution pitfalls.

ODBC rules are handled by the **BusinessRules.ODBC** server, and include:

| This command ... | Does this ...  | See page ...        |
|------------------|--|---------------------|
| DBOpen           | Opens a database.  | <a href="#">140</a> |
| DBCclose         | Closes one or more databases.  | <a href="#">142</a> |
| DBQuery          | Executes an SQL query against an open database, returning the number of records selected and, optionally, values of fields from the first selected record. | <a href="#">143</a> |
| DBExecute        | Executes a database command.   | <a href="#">145</a> |

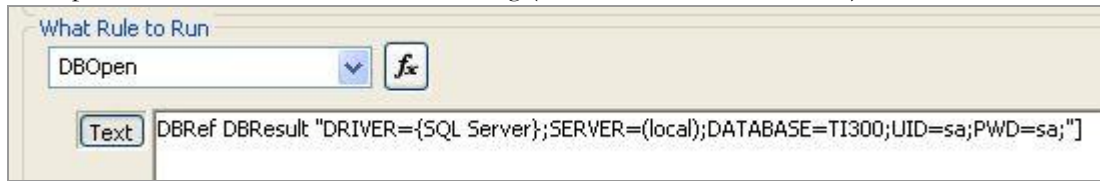
See Appendix E on page [271](#) for two extended ODBC examples.

## Setting up your ODBC Connection String

If you use ODBC business rules, you need to supply your database connection string in the DBOpen rule. You can put it in either or both of these places:

- **Within each DBOpen business rule**  
This method lets you globally change the connection string without having to edit a guideline and change business rules. This can save you significant effort when changing databases from test to production, for example.

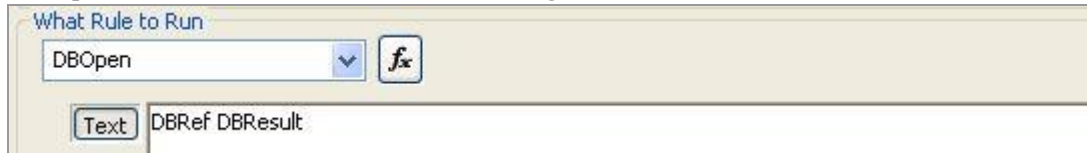
Example business rule with connection string (detailed in the next sections):



The screenshot shows a configuration window titled "What Rule to Run". It has a dropdown menu with "DBOpen" selected and a button with a function symbol (fx). Below this is a "Text" field containing the connection string: `DBRef DBResult "DRIVER={SQL Server};SERVER=(local);DATABASE=TI300;UID=sa;PWD=sa;"`.

- In the \$Dir.ini file for Instream and HIPAA Validator Desktop

Example business rule without connection string:



The screenshot shows a configuration window titled "What Rule to Run". It has a dropdown menu with "DBOpen" selected and a button with a function symbol (fx). Below this is a "Text" field containing the text: `DBRef DBResult`.

During validation, if Instream or HIPAA Validator HIPAA Validator Desktop find a connection string within a DBOpen business rule, they will use it. If not, they look in \$Dir.ini in their Bin directory for a connection string.

### Putting connection strings in \$Dir.ini

---

Use a text editor to edit \$Dir.ini in the Bin directory of Instream or HIPAA Validator Desktop.

Add a **Database** section at end of the file, like this:

```
[Database]
DBRef="DRIVER={SQL Server};SERVER=(local);DATABASE=TI300;UID=sa;PWD=sa;"
DBRef1="DRIVER={SQL Server};SERVER=FCSUPP10;DATABASE=TI301;UID=sa;PWD=sa;"
```

You can have as many lines in this section as you need.

DBRef and DBRef1 are logical names of your choice and are used as the first parameter in the DBOpen business rule. Notice that the example rules above use DBRef. According to the first \$Dir.ini database entry, this is the TI300 database.

## DBOpen

---

### Windows Instream and HIPAA Validator Desktop

---

Opens an ODBC connection to a particular database.

#### Format of Parameters

*DBRef ResultVar Connection*

Where:

|                   |   |
|-------------------|---|
| <i>DBRef</i>      | Identifier you are giving to this ODBC connection. It is used in subsequent ODBC business rules to tell Validator which database to use. <i>DBRef</i> must be alphanumeric with no spaces.  |
| <i>ResultVar</i>  | Variable that is to be used to contain the results of the DBOpen command. Result will be: <ul style="list-style-type: none"><li><b>0</b> = Database opened successfully</li><li><b>-1</b> = Bad parameter or argument</li><li><b>-2</b> = Could not allocate ODBC environment handle</li><li><b>-3</b> = Could not allocate ODBC connection handle</li><li><b>-4</b> = Could not connect to database</li><li><b>-5</b> = Could not allocate ODBC statement handle</li></ul> |
| <i>Connection</i> | The connection string used to establish the connection to the database. The connection string can be supplied in the \$Dir.ini file (see page 138) rather than here. If you use <code>local</code> in the connection string, be sure that it will be correct on all machines where this guideline will be used.   |

**Example 1:** SQL database.

What Rule to Run

DBOpen

Text PatientDB ErrVar "Driver={SQL Server};;Server=dbase;Database=PatDB;Uid=rwood;Pwd=my password"

**Example 2:** Access database.

What Rule to Run

DBOpen

Text PatientDB ErrVar "Driver={Microsoft Access Driver (\*.mdb)};DBQ=C:\\Program Files\\HIPAA Validator\\Pat.mdb"

**Example 3:** Named Data Source (via Control Panel>Administrative Tools>Data Sources (ODBC)).

The screenshot shows a configuration window titled "What Rule to Run". It features a dropdown menu with "DBOpen" selected and a function icon (fx). Below this, a "Text" tab is active, displaying the following text: "PatientDB ErrVar \"DSN=Human Resources;UID=Smith;PWD=Sesame\"".

or

The screenshot shows a configuration window titled "What Rule to Run". It features a dropdown menu with "DBOpen" selected and a function icon (fx). Below this, a "Text" tab is active, displaying the following text: "PatientDB ErrVar \"DSN=Human Resources;Trusted\_Connection=yes\"".

## DBCclose

---

### Windows Instream and HIPAA Validator Desktop

---

Closes one or more ODBC connections.

#### Format of Parameters

*DBRef* {*DBRef* ...}

or:

*DBRef* **ALL**

Where:

*DBRef*

The name of an ODBC connection to close. This name is the same one that was specified with `DBOpen`. Specifying **ALL** will cause all open ODBC connections to close.

#### Example



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu for "Rule" with "DBCclose" selected. To the right of the dropdown is a button with a function symbol (fx). Below the dropdown is a "Text" field containing the value "ProvDB".

## DBQuery

---

### Windows Instream and HIPAA Validator Desktop

---

Performs an SQL query on a specified database.

#### Format of the Parameters

*DBRef ResultVar SQL {Var1=n1 {Var2=n2 ...} }*

Where:

|                  |  |
|------------------|--|
| <i>DBRef</i>     | Name of an ODBC connection specified in the DBOpen command.  |
| <i>ResultVar</i> | Name of a variable that contains the results of the DBQuery command. If a result is less than zero, it is an error code; otherwise, the result is the number of records returned.<br><br>-1 = Bad Parameter or Argument<br>-2 = Database DBRef not opened, or had problem with open<br>-3 = Invalid Variable Assignment parameter            |
| <i>SQL</i>       | SQL command to execute. The SQL command must return a recordset. The SQL command can contain business rule variables in the form % <i>var</i> % where <i>var</i> is the name of a business rule variable. Before the SQL command is processed, any variables in the SQL string will be replaced with the contents of the specified variable. |
| <i>Var=n</i>     | A variable and the column number from the returned record where it is to get a value. The variable can then be used in other business rules. If zero records are selected, <i>Var</i> is cleared. If the DBQuery command returns an error, then <i>Var</i> remains unchanged.  |

#### Example 1

This example selects all records from the **PatTable** table in the **PatientDB** database whose **SSN** field matches the contents of the variable **2000AN102**.

It returns a recordset containing the following:

- ID (Field #1)
- Name (Field #2)
- SSN (Field #3)

Variables returned include:

- ResVar - The number of records found
- SSNVar - The first record's SSN (Field #3)
- NameVar - The first record's Name (Field #2)

What Rule to Run

DBQuery(ODBC)  ☐ Look-Ahead Rule

PatientDB ResVar "SELECT ID,Name,SSN FROM PatTable WHERE (SSN='%2000AN102%');"; SSNVar=3 NameVar=2

## Example 2

This uses a stored procedure.

What Rule to Run

DBQuery  ☐ Look-Ahead Rule

MYF5demo retVar "EXEC checkList '%BillProvIDQual%', '%Current\_Element%' " retName=1 retStatus=2

## DBExecute

---

### Windows Instream and HIPAA Validator Desktop

---

Executes an SQL command against an open ODBC database connection.

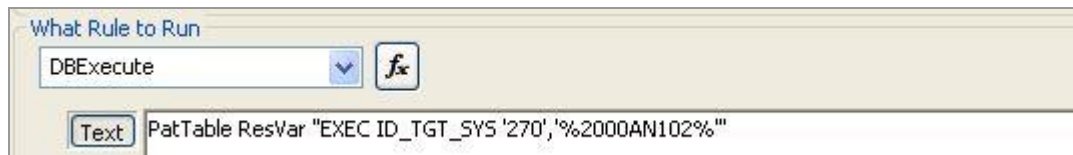
#### Format of Parameters

*DBRef ResultVar Command*

Where:

|                  |   |       |       |       |       |       |       |       |  |  |
|------------------|---|-------|-------|-------|-------|-------|-------|-------|--|--|
| <i>DBRef</i>     | Name of an ODBC connection to use. This name was specified with DBOpen.   |       |       |       |       |       |       |       |  |  |
| <i>ResultVar</i> | <p>The name of a variable to contain the results of the query. If the result is less than zero, it is an error code; otherwise, it is the number of records returned:</p> <p><b>-1 =</b> Bad parameter or Argument</p> <p><b>-2 =</b> Database <i>DBRef</i> not opened, or had problem with open</p> <p>These return codes show up in an EMSG in the DTL file.</p> <p>ODBC error numbers (see <b>APF.pdf</b>):</p> <table><tr><td>31029</td><td>31023</td><td>31024</td></tr><tr><td>31025</td><td>31026</td><td>31027</td></tr><tr><td>31028</td><td></td><td></td></tr></table> | 31029 | 31023 | 31024 | 31025 | 31026 | 31027 | 31028 |  |  |
| 31029            | 31023   | 31024 |       |       |       |       |       |       |  |  |
| 31025            | 31026   | 31027 |       |       |       |       |       |       |  |  |
| 31028            |   |       |       |       |       |       |       |       |  |  |
| <i>Command</i>   | The database command to execute. It can contain business rule variables in the form % <i>var</i> % where <i>var</i> is the name of a business rule variable. Before the command is processed, <i>var</i> will be replaced with the contents of the specified variable.  |       |       |       |       |       |       |       |  |  |

#### Example



What Rule to Run

DBExecute [v] [fx]

[Text] PatTable ResVar "EXEC ID\_TGT\_SY5 '270','%2000AN102%"



# Run Business Rules

## RunAlways

Specifies that a business rule is to be run on an optional segment, regardless of whether the segment is actually present in the data.

This only works on a segment, not an element.

### Format of Parameters

*BusinessRule*

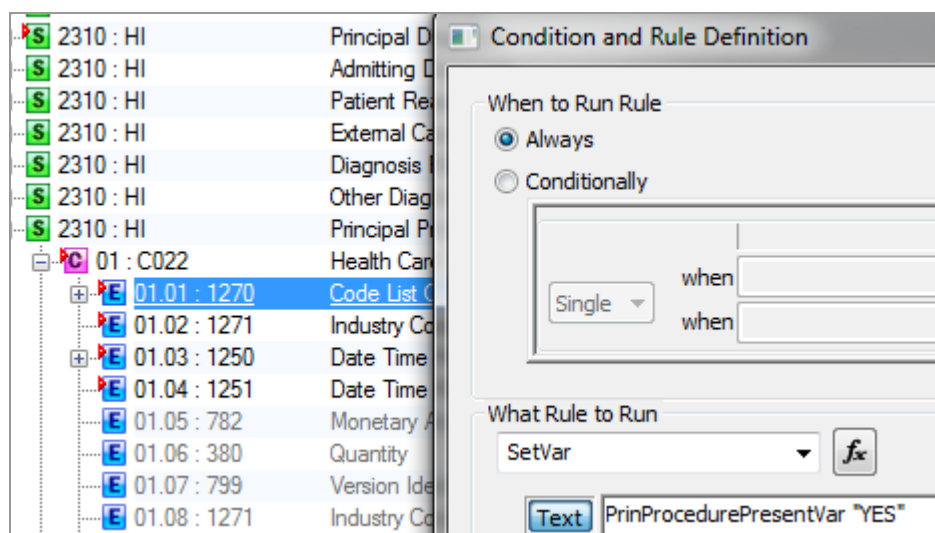
Where:

*BusinessRule*                      Business rule to run, whether data is present or not.

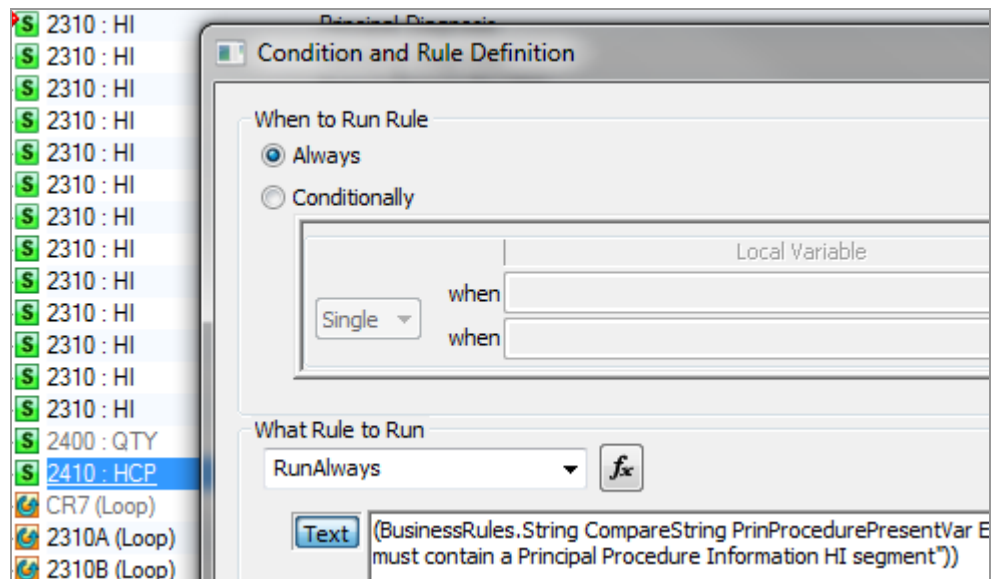
### Example

We want to see if the Principal Procedure Information segment is present. The HI segments can come in any order and most are optional.

1. Set a variable on an element within the segment:



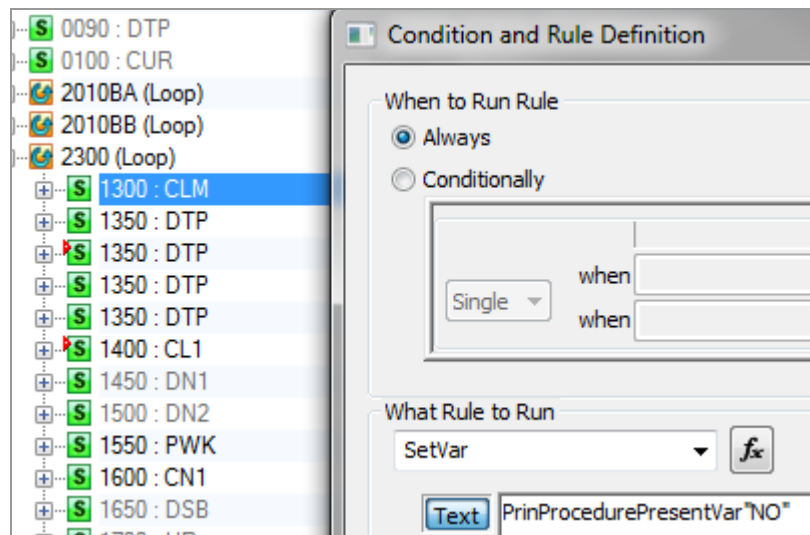
2. Check the variable on an optional segment below the last HI:



The rule parameter is:

```
(BusinessRules.String CompareString PrinProcedurePresentVar EQ "NO"
(BusinessRules.Utilities DisplayErrorByNumber 0 0 "The data must contain a
Principal Procedure Information HI segment"))
```

3. Initialize the variable on the mandatory CLM segment, above the HI segments:



When the Principal Diagnosis Information HI segment is missing, the message appears:

```
Loaded Transaction Set 837 from Standard RunAlways (Date=2012/03/07
Start of Transaction Set, Ctl. No. 0370
The data must contain a Principal Procedure Information HI segment
End of Transaction Set, Ctl. No. 0370, contains 69 Segments
End of Functional Group, Ctl. No. 370, contains 1 Transaction Set(s)
```

## RunNoData

---

### Instream and HIPAA Validator Desktop

---

Specifies a business rule to be run on this segment if and only if the segment is not present in the EDI. This only works on a segment, not an element, and the segment cannot be within an unused loop.

**Warning.** This business rule significantly slows down validation.

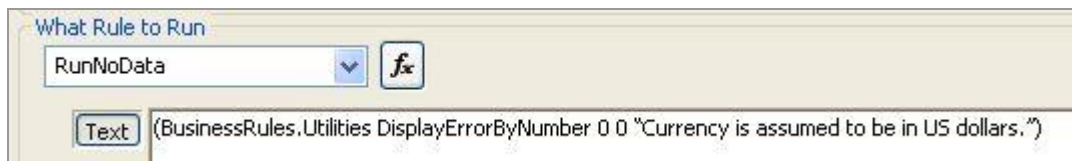
#### Format of Parameters

*BusinessRule*

Where:

*BusinessRule*                      Business rule to run if no data is present.

**Example.** This example attached to the CUR segment displays a message if the segment is not present.



**HIPAA Validator Desktop Demo.** The demo guideline NO\_DATA has this rule on the CUR segment. Use it to validate 837Idate.txt or 837I\_4010\_H\_FutureDateBHT.txt in HIPAA Validator Desktop's DemoData directory. Since there is no CUR segment in the data, you will see the message "Currency is assumed to be in US dollars."

**Instream Demo.** Execute **V\_837I\_4010\_noData** in Instream's Scripts directory to see this rule in action. Since there is no CUR segment in the data, the detail results file will see the message "Currency is assumed to be in US dollars."

# Substitute Business Rules

---

## Instream

---

The Substitute, SubstituteFind, SubstituteReplace and MakeKey business rules let Instream users replaces the value in the current element or sub-element with a new value.

They are used with the Instream's Dataswapper program.

## DeleteSegment

---

### Instream

---

**Details and examples** are in the Dataswapper Business Rules section of **TIB\_fsp\_instream\_<n.n>\_dataswapper.pdf**.

Deletes the current segment from the EDI.

#### Format of Parameters

*MetaData*

Where:

*MetaData*

Optional. "Literal" or variable containing text for your own use. It appears in the SBSTA record in the Dataswapper audit file.

## InsertSegment

---

### Instream

---

**Details and examples** are in Dataswapper Business Rules section of **TIB\_fsp\_instream\_<n.n>\_dataswapper.pdf**.

Inserts a segment into the EDI above or below the current location.

#### Format of Parameters

*SegmentID (Elements) MetaData*

Where:

|                   |   |
|-------------------|---|
| <i>SegmentID</i>  | A “Literal” or variable containing the segment ID.  |
| <i>(Elements)</i> | A series of FindKeys holding the values to replace. The series is surrounded by parentheses. To include sub-elements, surround them in a separate set of parentheses. See Example 2 below. The FindKeys are set with SubstituteReplace rules, which can come before or after the InsertSegment. |
| <i>MetaData</i>   | Optional. “Literal” or variable containing text for your own use. It appears in the SBSTA record in the Dataswapper audit file.   |

## MakeKey

---

### Instream

---

**Details and examples** are in Dataswapper Business Rules section of **TIB\_fsp\_instream\_<n.n>\_dataswapper.pdf**.

Creates unique key for SubstituteFind/SubstituteReplace pairs by incrementing a counter at the end of a string of characters.

#### Format of Parameters

*Prefix KeyVar*

Where:

|               |  |
|---------------|--|
| <i>Prefix</i> | “Literal” or variable holding the base part of the key. MakeKey will automatically add an incrementing counter to this base. |
| <i>KeyVar</i> | Variable containing the key.   |

## Substitute

---

### Instream

---

**Details and examples** are in Dataswapper Business Rules section of **TIB\_fsp\_instream\_<n.n>\_dataswapper.pdf**.

Replaces the value in the current element or sub-element with a new value.

#### Format of Parameters

*ReplaceValue Metadata*

Where:

*ReplaceValue*                      “Literal” or variable containing a value that is to replace the value in the current value.

*Metadata*                              Optional. For your own use. “Literal” or variable containing the text of your choice that is to appear at the end of the SBST record in the detail results file.

## SubstituteFind

---

### Instream

---

**Details and examples** are in Dataswapper Business Rules section of **TIB\_fsp\_instream\_<n.n>\_dataswapper.pdf**.

SubstituteFind identifies a value that is to be replaced. It specifies that the value in the current element or subelement is to be replaced.

#### Format of Parameters

*Key Metadata*

Where:

*Key*                                      “Literal” or variable containing a key that will identify this element as the one where the value is to be replaced.

*Metadata*                              Optional. For your own use. “Literal” or variable containing the text of your choice that is to appear at the end of the SBSTF record in the detail results file.

# SubstituteReplace

---

## Instream

---

**Details and examples** are in Dataswapper Business Rules section of **TIB\_fsp\_instream\_<n.n>\_dataswapper.pdf**.

SubstituteReplace identifies the value that will replace a value identified with a SubstituteFind that has the same key.

### Format of Parameters

*Key ReplaceValue*

Where:

|                     |  |
|---------------------|--|
| <i>Key</i>          | “Literal” value or a variable containing the same key as the one that identifies the element to be replaced. This same key appears in the SubstituteFind rule. |
| <i>ReplaceValue</i> | “Literal” or variable containing a value to replace the one in the SubstituteFind element.   |

# Utilities Business Rules

## AppendString

---

### All validation programs

---

Appends one string or constant to the end of another.

#### Format of Parameters

*DestStr SourceStr*

Where:

|                  |  |
|------------------|--|
| <i>DestStr</i>   | Variable to which another value should be appended. If DestStr does not exist, it will be created.   |
| <i>SourceStr</i> | The value to be appended to the end of DestStr. This can be a variable, a literal in double quotes, Current_Element, or Current_Date. If it does not exist as a variable, it will be treated as a literal value. |

**Example.** This example set of rules issues an error message if it finds that the same procedure was given to the patient more than once in a given day.

This rule appends the date to the industry code. It is applied to the Industry Code and the Date Time Period in the first composite in the HI segment shown below:

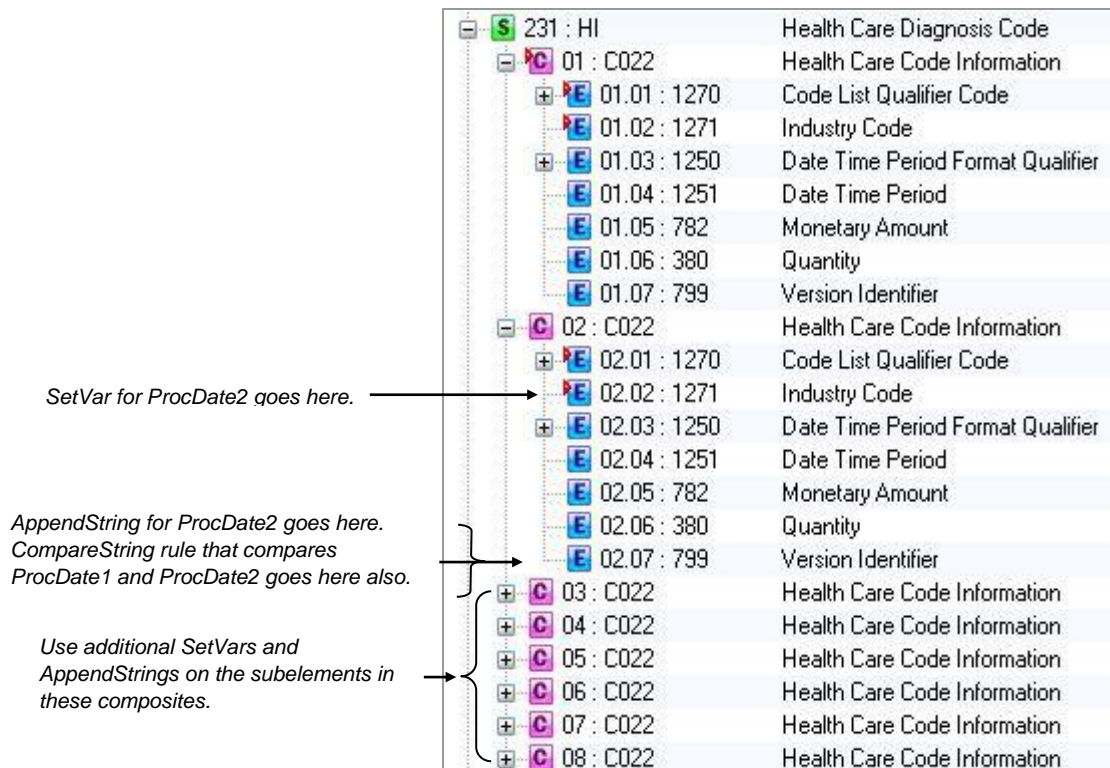
What Rule to Run

AppendString

Text ProcDate1 Current\_Element

Similar AppendStrings are used in the other composites, but with different names for the variables.





A rule could be applied to the second Date Time Period element that did a CompareString on ProcDate1 and ProcDate2 and issued a custom error message if they matched:



The format of DisplayErrorByNumber is different for Analyzer.

Similar CompareString rules could be applied to the other Date Time Periods in the subsequent composites.

## BuildString

---

### All validation programs

---

Builds a string from a list of variables, constants, or reserved words.

#### Format of Parameters

*DestStr Separator SourceStr1 SourceStr2 ...*

Where:

*DestStr* Variable to contain the values from the *SourceStrs*. If this variable exists, its contents will be overwritten. If it does not exist, it will be created.

*Separator* The character to be used to separate the values. This can be a keyboard character, a space (surrounded by double quotes), or nothing (two consecutive double quotes)

| Separator wanted  | Separator in business rule | Example output     |
|-------------------|----------------------------|--------------------|
| one slash         | /                          | ANDREWS/ALAN/A     |
| space             | " "                        | ANDREWS ALAN A     |
| no separator      | ""                         | ANDREWSALANA       |
| space-slash-space | " / "                      | ANDREWS / ALAN / A |

*SourceStr* The values to be combined into *DestStr*. These can be a combination of variables, literals in double quotes, *Current\_Element*, or *Current\_Date*. Ones that do not exist as variables will be treated as literals.

#### Example

This rule places this information into a variable Subname:

- The literal text "Name is "
- The contents of variables **SubLastName** and **SubFirstName**
- The contents of the current element.

Each is separated with a slash.

What Rule to Run

BuildString

Text Subname "/" "Name is " SubLastName SubFirstName Current\_Element

Output might look like this: Name is /ANDREWS/ALAN/A

# ChangeCase

---

## All validation programs

---

Converts all letters in a string to upper or lower case.

### Format of Parameters

*SourceString ResultVar CaseOption*


Where:

|                     |  |
|---------------------|--|
| <i>SourceString</i> | The string to be converted. This can be a literal in double quotes, a system variable like <code>Current_Element</code> , or a variable.   |
| <i>ResultVar</i>    | The variable to contain the result.  |
| <i>CaseOption</i>   | An optional parameter that specifies whether to convert to upper or lower case. This can be a variable name, or U or L surrounded by optional double quotes:<br><br>U        (default) Convert all lowercase letters to their uppercase equivalent<br><br>L        Convert all uppercase letters to their lowercase equivalent |


### Example 1

Assume that the current element contains `Westerville Medical Clinic`.

Either of these rules put `WESTERVILLE MEDICAL CLINIC` into variable `SubmitterVar`:

| What Rule to Run                         |                |   |
|--|----------------|---|
| ChangeCase                               |                | <input checked="" type="checkbox"/>  |
| <input type="checkbox"/> Look-Ahead Rule |                |   |
| <input checked="" type="checkbox"/> Text | Parameter Name | Parameter Value   |
|  | SourceStr      | Current_Element   |
|  | DestVar        | SubmitterVar  |
|  | Option         | U   |

Or, using default behavior for Option:

| What Rule to Run                         |                |   |
|--|----------------|---|
| ChangeCase                               |                | <input checked="" type="checkbox"/>  |
| <input type="checkbox"/> Look-Ahead Rule |                |   |
| <input checked="" type="checkbox"/> Text | Parameter Name | Parameter Value   |
|  | SourceStr      | Current_Element   |
|  | DestVar        | SubmitterVar  |
|  | Option         |   |

### Example 2

Assume that:

- Variable `PatNameVar` contains `Fred Flintstone`
- Variable `CaseVar` contains `U`

This puts FRED FLINTSTONE into PatNameCapsVar.

| Parameter Name | Parameter Value |
|----------------|-----------------|
| SourceStr      | PatNameVar      |
| DestVar        | PatNameCapsVar  |
| Option         | CaseVar         |

## ChangeElmAttribute

---

### All validation programs

---

Changes the element's type, minimum length, maximum length, or user attributes.

This rule executes before the length or user attribute is validated.

#### Format of Parameters

*Attribute Value*

Where:

*Attribute* "UA", "Type", "Min", or "Max", or a variable containing one of these.

*Value* If Attribute is UA (for User Attributes), value can be one of these or a variable containing one of these:

|      |                         |
|------|-------------------------|
| "O"  | Used (Optional)         |
| "N"  | Not Used                |
| "MU" | Must be Used (Required) |
| "R"  | Recommended (Advised)   |
| "NR" | Not Recommended         |
| "D"  | Dependent               |

If Attribute is Type, value can be one of these or a variable containing one of these:

"AN"  
 "ID"  
 "N"  
 "R"

See **DataTypes.pdf** in EDISIM's Documentation directory for details.

If Attribute is Min or Max, value can be an integer in quotes or a variable containing an integer.

## Examples:

This example checks the contents of local variable 2010AANM108. If it contains XX, then the current element's type is changed to R.

When to Run Rule

☐ Always

☒ Conditionally

|        | Local Variable   | Operator | Constant |
|--------|------------------|----------|----------|
| Single | when 2010AANM108 | EQ       | XX       |
|        | when             | EQ       |          |

What Rule to Run

ChangeElmAttribute

Text "Type" "R"

This example checks the value of 2010AANM108. If it contains 24, the maximum length of the value in the element is 12:

When to Run Rule

☐ Always

☒ Conditionally

|        | Local Variable   | Operator | Constant |
|--------|------------------|----------|----------|
| Single | when 2010AANM108 | EQ       | 24       |
|        | when             | EQ       |          |

What Rule to Run

ChangeElmAttribute

Text "Max" "12"

This example uses two variables:

- NewAttr contains **Type**.
- NewAttrValue contains R.

What Rule to Run

ChangeElmAttribute

Text NewAttr NewAttrValue

# CheckFormat

---

## All validators except Analyzer

---

For a rule that will work in Analyzer, see Other CheckDigit Options on page [232](#).

Checks the format of the value. Formats include:

### Format of Parameters

*CheckType Value (IfFalseAction) (IfTrueAction)*

Where:

*CheckType*

One of these:

|                     |  |
|---------------------|--|
| SocialSecurity      | See below  |
| NationalProviderID  | See below  |
| CHARSET( <b>x</b> ) | Data must conform to the characters in the specified character set. Sets available are X12B, X12E, UNOA and UNOB.<br>See below.  |
| EAN8                | Data must be exactly 8 characters with no leading zeros.   |
| EAN13               | Data must be exactly 13 characters long with no leading zeros.   |
| EAN14               | Data must be exactly 14 characters long with no leading zeros.   |
| HIN                 | Data must be 9 characters long, with position 7 acting as a check digit for verifying the first six positions.<br>Positions 8 and 9 are a suffix that acts as a unique identifier.<br>Example:<br>9C8341600<br>(9C83416 is the base HIN, with 6 being the check digit. 00 is the suffix identifier.) |
| MOD11               | Calculates the Mod11 check digit for a value. The format of this rule is slightly different than other CheckFormat rules. Please see <b>Example 11</b> on page 166.  |
| POA                 | Positions 1-3 must contain the literal string POA<br>Position 4 must contain a Y<br>Starting at position 5 (up to 25 positions) - any combination of N, U, W, 1, Y<br>The last position must contain X or Z.   |

|                            |   |
|----------------------------|---|
| POAX                       | Like POA, but will allow: <ul style="list-style-type: none"> <li>- a regular POA code</li> <li>- a POA code whose next-to-the-last character is a Z or X, and whose last character is a Y, N, U, W, or 1 (an e-code). This is only used in the 4010 837I when the H103.01=BN.</li> </ul>  |
| SSCC                       | Data must conform to the Serial Shipping Container Code - be exactly 18 digits.   |
| UPC12                      | Data must be exactly 12 characters long and can include leading zeros. The last digit is a check-digit.   |
| USER( <i>min-maxZn</i> )   | Specify the minimum and maximum length and the number of leading zeros included within that length. The last digit is a check digit. There is no space before the parenthesis.<br><br>For details about the format within the parentheses, see <a href="#">USER check digit</a> and examples <a href="#">7</a> and <a href="#">8</a> below. |
| USERNC( <i>min-maxZn</i> ) | Same as USER except the last digit is not a check digit. There is no space before the parenthesis.<br><br>See example <a href="#">9</a> below.  |
| ALL_BLANKS                 | Value is all blanks   |
| ALL_?                      | ? is a capital or lower case letter or a digit.<br><br>Examples:<br><br>ALL_0<br>(Value is all zeros)<br><br>ALL_9<br>(Value is all nines)<br><br>ALL_A<br>(Value is all capital A's)   |

|                 |   |
|-----------------|---|
| <i>Value</i>    | The value being checked. This can be a variable, Current_Element, or a literal in double quotes. If omitted, the value of Current_Element is assumed.                     |
| (IfFalseAction) | Action to take if the format does not match. Required if you are specifying an IfTrueAction. To do nothing if false, use this:<br><br>(BusinessRules.Utilities DoNothing) |
| (IfTrueAction)  | Action to take if the format matches.   |



## SocialSecurity

---

- Length can be 9 digits, plus optional dashes if it is X12-4010 data: *nnnnnnnnnn* or *nnn-nn-nnnn*. For X12-5010 and later, it cannot contain dashes.
- Area code (first three digits) must not be 000 or 666. HIPAA validation products look up the area code in the SSNArea table. Analyzer does not do this lookup.
- Group (4<sup>th</sup> and 5<sup>th</sup> digits) must not be 00.
- Serial (last four digits) cannot be 0000.
- For HIPAA guidelines, the first five digits are automatically checked according to the SSA guidelines.

## NationalProviderID

---

- Length is 9 digits followed by one numeric check digit.
- The check digit uses the Luhn formula for the modulus 10 “double-add-double” check digit and includes the prefix 80840, even though that is not included in the EDI value.
- NationalProviderID

## CHARSET

---

The CHARSET CheckFormat type code that allows you to verify that a value is comprised only of characters in the specified character set.

The format is

CHARSET(*x*)

There should be no spaces between CHARSET and (*x*) or within the parentheses; spaces will result in an error.

Where *x* is one of the following Character Set ID codes:

**X12B** = X12 Basic character set

- Uppercase letters: A–Z
- Decimal digits: 0–9
- Punctuation Characters: ! " & ' ( ) \* + , - . / : ; ? = space

**X12E** = X12 Extended character set

- Uppercase letters: A–Z
- Lowercase letters: a–z
- Decimal digits: 0–9
- Punctuation Characters: ! " & ' ( ) \* + , - . / : ; ? = space  
% @ [ ] \_ { } \ | < > ~ # \$

**UNOA** = EDIFACT UNOA character set

- Uppercase letters: A–Z
- Decimal digits: 0–9
- Punctuation Characters: . , - ( ) / = space

**UNOB** = EDIFACT UNOB character set

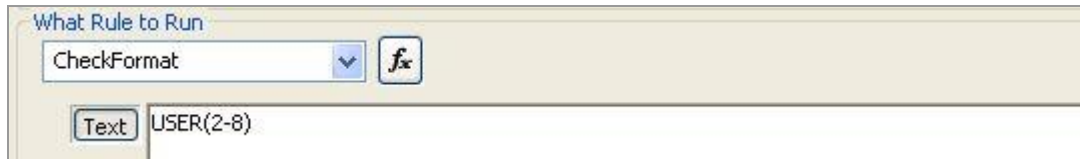
- Uppercase letters: A–Z
- Lowercase letters: a–z
- Decimal digits: 0–9
- Punctuation Characters: . , - ( ) / = space ' + : ? ! " % & \* ; < >

## USER check digit

---

The USER check digit is a Modulo 10 calculation.

Consider this rule as an example of USER:



What Rule to Run

CheckFormat

Text USER(2-8)

And consider this value in the data:

```
ISA*00*          *00*
GS*HC*901234572000*90888
ST*837*0386~
BHT*0019*00*1234565*2003
REF*87*004010X096A1~
```

The last digit (5) is the check digit. Is it correct or not?

Here is how Foresight validators will calculate the check digit:

1. Add up the digits in the “odd” positions, starting FROM THE RIGHT with the digit just BEFORE the check digit: **1234565**  
 $6+4+2=12$   
Multiply this sum by 3:  
 $12*3=36$
2. Add up the digits in the “even” positions: **1234565**  
 $5+3+1=9$
3. Add the odd and even results:  
 $36+9=45$
4. The check digit is the number would you have to add to this to get to a multiple of 10. In our example:  
 $45+5=50$   
Our check digit should be 5, which makes the value 1234565 correct.

## CheckFormat examples

### Example 1

This example displays default error message (31000 Comparison Failed!) if the current element's value does not match the format of a Social Security Number.

The screenshot shows the 'What Rule to Run' dialog. The 'Rule' dropdown is set to 'CheckFormat'. The 'Text' field contains 'SocialSecurity'.

### Example 2

This example displays custom error message 32001 (Social Security number format is wrong) if the current element's value does not match the format of a Social Security Number.

The screenshot shows the 'What Rule to Run' dialog. The 'Rule' dropdown is set to 'CheckFormat'. The 'Text' field contains 'SocialSecurity Current\_Element (BusinessRules.Utilities DisplayErrorByNumber 32001)'.

The format of DisplayErrorByNumber is different for Analyzer.

### Example 3

This example displays custom error message 32001 (Social Security number format is wrong) if variable 2010BAREF01 contains SY and the current element's value does not match the format of a Social Security Number.

The screenshot shows the 'What Rule to Run' dialog. The 'Rule' dropdown is set to 'CompareString'. The 'Text' field contains '2010BAREF01 EQ "SY" (BusinessRules.Utilities CheckFormat SocialSecurity Current\_Element (BusinessRules.Utilities DisplayErrorByNumber 32001))'.

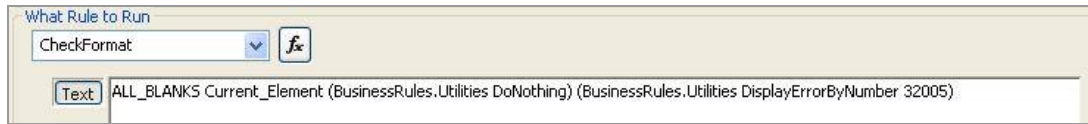
### Example 4

This example has the same result as the previous example, but the “if” part uses a local variable (see page 239) instead of a SetVar variable.

The screenshot shows the 'When to Run Rule' dialog. The 'When to Run Rule' section has 'Conditionally' selected. The 'Local Variable' table has two rows: '2010BAREF01' with operator 'EQ' and constant 'SY', and an empty row with operator 'EQ' and an empty constant. The 'What Rule to Run' section has 'Rule' set to 'CheckFormat' and 'Text' set to 'SocialSecurity Current\_Element (BusinessRules.Utilities DisplayErrorByNumber 32001)'.

### Example 5

This example displays custom error message 32005 if the current element contains all blanks.



What Rule to Run

CheckFormat

Text: ALL\_BLANKS Current\_Element (BusinessRules.Utilities DoNothing) (BusinessRules.Utilities DisplayErrorByNumber 32005)

### Example 6

This example displays custom error message 32006 if the current element does not contain all blanks, and custom error message 32005 if it does contain all blanks.



What Rule to Run

CheckFormat

Text: ALL\_BLANKS Current\_Element (BusinessRules.Utilities DisplayErrorByNumber 32006) (BusinessRules.Utilities DisplayErrorByNumber 32005)

### Example 7

This example displays a default error message if the current element does not contain exactly 10 digits including up to one leading zero. The last digit is a check digit.



What Rule to Run

CheckFormat

Text: USER(10Z1)

### Example 8

This example displays a default error message if the current element does not contain exactly 10 to 12 digits including up to one leading zero. The last digit is a check digit.



What Rule to Run

CheckFormat

Text: USER(10-12Z1)

### Example 9

This example displays a default error message if the current element does not contain exactly 2 to 8 digits. This can contain up to 6 leading zeros. The list digit is not a check digit.



What Rule to Run

CheckFormat

Text: USERNC(2-8Z6)

### Example 10

This example verifies that the value (“Value to be Checked”) is comprised only of characters in the X12B character set, then causes the FalseAction to be taken because the string contains lower case letters not included in the X12B character set.

|                  |  |
|------------------|--|
| What Rule to Run |  |
| CheckFormat      | <input type="checkbox"/> Look-Ahead Rule |
| Text             | CHARSET(X12B) "Value to be Checked"      |

### Example 11

This example uses MOD11 to calculate a value’s check digit, and then see if the data includes the correct check digit in this HL7 ORU R01 Ambulance guideline:

|                                  |                                 |
|----------------------------------|---------------------------------|
| ORU_R01                          | Unsolicited Observation Message |
| + S 0010 : MSH                   | Message Header Segment          |
| + S 0020 : SFT                   | Software Segment                |
| + S 0030 : UAC                   | User Authentication Credential  |
| - G 0040 : PATIENT_RESULT (Loop) |                                 |
| - S 0050 : PID                   | Patient Identification Segment  |
| - E 01 : 104                     | Set ID - PID                    |
| + C 02 : 105                     | Patient ID                      |
| - C 03 : 106                     | Patient Identifier List         |
| - E 03.01 : 99079                | ID Number                       |
| - E 03.02 : 99080                | Identifier Check Digit          |
| + E 03.03 : 99081                | Check Digit Scheme              |

We capture the ID number with this rule on the PID-03-01:

|                  |  |                 |
|------------------|--|-----------------|
| What Rule to Run |  |                 |
| SetVar           | <input type="checkbox"/> Look-Ahead Rule |                 |
| Text             | Parameter Name                           | Parameter Value |
|                  | VarToAssign                              | PatIDNumVar     |
|                  | Value                                    | Current_Element |

We capture its Check Digit from the data with this rule on the PID-03-02:

|                  |  |                       |
|------------------|--|-----------------------|
| What Rule to Run |  |                       |
| SetVar           | <input type="checkbox"/> Look-Ahead Rule |                       |
| Text             | Parameter Name                           | Parameter Value       |
|                  | VarToAssign                              | PatIDNumCheckDigitVar |
|                  | Value                                    | Current_Element       |

If the value in the PID-03-03 is M11, that means we should use the Mod 11 algorithm. If so, we calculate the check digit for the value in the PID-03-01:

What Rule to Run

CompareString  ☐ Look-Ahead Rule

Text

| Parameter Name | Parameter Value   |
|----------------|-------------------|
| Value1         | Current_Element   |
| Operator       | EQ                |
| Value2         | "M11"             |
| ThenRule       | CheckFormat       |
| > CheckType    | PatIDNumVar       |
| > Value        | CheckDigitCalcVar |

If the check digit value in the PID-03-02 is different from the one calculated by CheckFormat, we issue an error message.

What Rule to Run

CompareString  ☐ Look-Ahead Rule

Text

| Parameter Name | Parameter Value       |
|----------------|-----------------------|
| Value1         | PadIDNumCheckDigitVar |
| Operator       | NE                    |
| Value2         | CheckDigitCalcVar     |
| ThenRule       | DisplayErrorByNumber  |
| > ErrorNumber  | 32001                 |

## CreateFSUID

---

### Instream

---

CreateFSUID creates a unique identifier (TIBCO Foresight Unique ID or FSUID) and places it in a variable. The Identify (see page 175) or Match (see page 178) rules then use this variable to output an IDENT record.

#### Warnings

It is extremely important that you never output the same FSUID twice. The safest way to ensure this is to use a CreateFSUID rule before each Identify or Match rule.

#### Format of Parameters

*FSUID CompressedID*

Where:

*FSUID* Variable to hold a unique 36-character TIBCO Foresight User ID (FSUID).

*CompressedFSUID* Optional variable to hold a 27-character version of the FSUID.

**Example 1.** This example, placed on the subscriber CLM segment, causes Instream to insert an IDENT record each time a CLM segment is encountered.

First, create an FSUID in a variable called FSUniqueID:

|                                 |                       |
|---------------------------------|-----------------------|
| External Routine                |                       |
| Server: BusinessRules.Utilities | Function: CreateFSUID |
| Parameters: FSUniqueID          |                       |



Next, write the IDENT record using the FSUID value in the variable:

|                                 |                    |
|---------------------------------|--------------------|
| External Routine                |                    |
| Server: BusinessRules.Utilities | Function: Identify |
| Parameters: FSUniqueID          |                    |

Each time a subscriber CLM is encountered in the EDI, the detail results file will contain an IDENT record similar to this:

```
STRUS      31|2300|0|1|1159
SVALU      31|S009|464|CLM*2235057*460.00***25:B:1*N*A*N*I*P*OA*****1
IDENT      31|I|7ee91081-f49f-11de-a384-f131e23d4046|1|
DTL        31  2300 CLM1332C023          28 5 2          1          10618 3  4641 ...
```

**Example 2.** These two rules put an IDENT record in the detail results file. This would be suitable for marking the document level for generic X12 or EDIFACT documents that will be sent to TI.

|   |   |
|---|---|
| Business Rules  |   |
|  | 1. Always call External Routine BusinessRules.Utilities.CreateFSUID FSUniqueID CompressedFSFSUniqueID |
|  | 2. Always call External Routine BusinessRules.Utilities.Identify FSUniqueID                           |

## DisplayErrorByNumber

---

### All validation programs

---

In Analyzer, DisplayErrorByNumber only supports explicit text as shown in the example for Analyzer.

Looks up the error number and then displays the corresponding error text. For more details about error messages, see Appendix B on page [255](#).

---

### Format of parameters for EDISIM Validator, Instream and HIPAA Validator Desktop

---

*ErrorNumber Severity OverridingErrorMessage*

Where:

*ErrorNumber*                      Number of the error in FSBRErrs.txt or CustomerFSBRERRS.TXT files (in the Bin directory). The number can be from 32000 to 32999. Please see **ErrorMessageNumbers.pdf** for a complete list of error number ranges.

*Severity*                              Optional. Specifies the severity, one of these:

- |    |                    |
|----|--------------------|
| -1 | for Un-Initialized |
| 0  | for Message        |
| 1  | for Non-Critical   |
| 2  | for Warning        |
| 3  | for Error          |
| 4  | for Fatal          |
| 5  | for User1          |
| 6  | for User2          |

**Examples.** This example displays message 32001. Because the severity is listed as 2, the message displays as a warning.



Also see the example for ListCheck on page [113](#).



## Format of Parameters for any Foresight validation program

---

To display a message, use two zeros, separated by a space, and then the text:

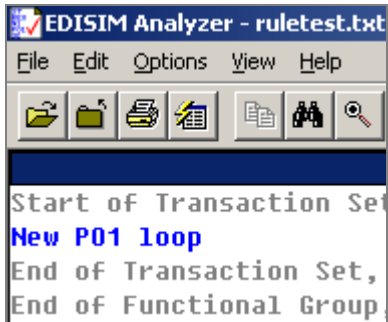


What Rule to Run

DisplayErrorByNumber  

Text 0 0 "New PO1 loop"

Analyzer will display the text:



See also FSVBExit.DisplayMessage on page [233](#).

## FindString

---

### All validation programs

---

Allows you to determine if String A can be found in String B, and if so, return its starting position.

#### Format of Parameters

*StringA StringB ResultVar (StartPos) (Opt)*

Where:

|                   |  |
|-------------------|--|
| <i>StringA</i>    | The string (variable or constant) to search.   |
| <i>StringB</i>    | The substring (variable or constant) to search for.<br><br>If StringB is not found in StringA, then the result will be zero.<br>Otherwise, it will be the position in StringA that StringB starts (1-based; i.e. the first character position is 1).   |
| <i>ResultVar</i>  | The variable to contain the result. Note that if either StringA or StringB is an empty string, the value of -1 will be returned in ResultVar.  |
| <i>(StartPos)</i> | Optional. A variable or constant that tells FindString where to start its search within StringA. If StartPos is omitted, then the search begins in position 1 (i.e. the first character) of StringA. If StartPos is less than 1 or greater than the length of StringA, then a zero will be returned. If StartPos is not numeric, it will be ignored. |
| <i>(Opt)</i>      | Optional. A variable or constant that modifies aspects of the function. Currently, the only option supported is <b>I</b> , which means Ignore Case.  |

#### Examples (Simple)

The following examples use these assumptions:

- SetVar INVCONST “Office Box”
- Current element = “Post office box 1234”

This example causes 0 to be put into VARPOPOS because the exact string wasn’t found.

What Rule to Run

FindString  ☐ Look-Ahead Rule

Current\_Element INVCONST VARPOPOS

This example causes 6 to be put into VARPOPOS because, once we ignore the case of the letters, the string is found.

|                  |  |
|------------------|--|
| What Rule to Run |  |
| FindString       | <input type="checkbox"/> Look-Ahead Rule |
| Text             | Current_Element INVCONST VARPOPOS "I"    |

### Examples (Complex)

This more complex example shows how to find the second occurrence of a delimiter:

The following examples use this assumption:

- Current element = "123-ER-456T" I"

This rule causes 4 to be put into DELIMPOS1.

|                  |  |
|------------------|--|
| What Rule to Run |  |
| FindString       | <input type="checkbox"/> Look-Ahead Rule |
| Text             | Current_Element "-" DELIMPOS1            |

This AddVar rule increments DELIMPOS1 to 5.

|                  |  |
|------------------|--|
| What Rule to Run |  |
| AddVar           | <input type="checkbox"/> Look-Ahead Rule |
| Text             | DELIMPOS1 1                              |

This rule causes 7 to be put into DELIMPOS2.

|                  |  |
|------------------|--|
| What Rule to Run |  |
| FindString       | <input type="checkbox"/> Look-Ahead Rule |
| Text             | Current_Element "-" DELIMPOS2 DELIMPOS1  |

## GenerateFSUID

---

### Instream

---

**NOTE** This rule has been deprecated. Please see [FSUID\\_and\\_AppDocs.pdf](#) for alternatives.

## GetToken

---

### Instream, HIPAA Validator Desktop, and Analyzer

---

Finds a specific value in a series of delimited values and places it in another variable.

#### Format of Parameters

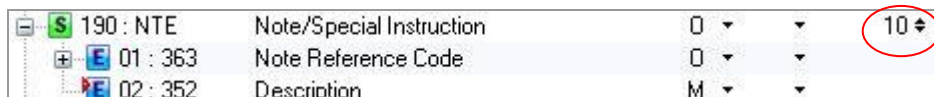
*DestVar SourceVar Index Delimiter*

Where:

|                  |  |
|------------------|--|
| <i>DestVar</i>   | The variable that is to hold the extracted value.  |
| <i>SourceVar</i> | A variable, literal in double quotes, or <code>Current_Element</code> that holds the series of values before one of them is extracted.       |
| <i>Index</i>     | The position of the value that is to be extracted. This can be a literal (no quotation marks), a variable, or <code>Current_Element</code> . |
| <i>Delimiter</i> | Optional. The delimiter that separates values in <i>SourceVar</i> . If omitted, a space character is assumed.                                |

**Example.** This example requires that the first NTE segment starts with the code MED and the second repetition of the same NTE segment starts with the code NTR.

AddVar is used on the NTE segment as a counter. It increments by 1 with each NTE segment. This segment can repeat up to 10 times, and has many code values available to the NTE01.



|           |                          |   |    |
|-----------|--------------------------|---|----|
| 190 : NTE | Note/Special Instruction | 0 | 10 |
| 01 : 363  | Note Reference Code      | 0 |    |
| 02 : 352  | Description              | M |    |

The AddVar NTE segment counter might look like this:



What Rule to Run

AddVar

Text CLMnote 1

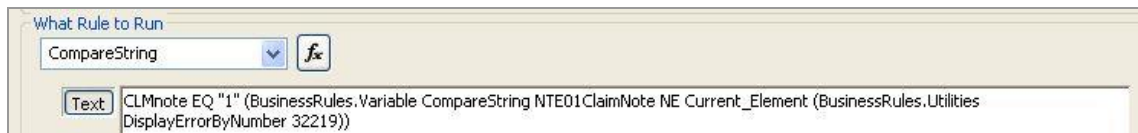
And the GetToken on the NTE01ClaimNote might look like this:



The GetToken places MED in a variable called NTE01ClaimNote for the first instance of the NTE segment (when CLMnote will equal 1).

It places NTR in a variable called NTE01ClaimNote for the second instance of the NTE segment (when CLMnote will equal 2).

You could then put a CompareString on the NTE01ClaimNote, after the GetToken rule, to see if CLMnote equals 1, and, if so, Current\_Element should equal the contents of NTE01ClaimNote - which will be MED.



Another rule could check to see if CLMnote equals 2. If true, then the contents of Current\_Element should equal NTR.

## Identify

This follows a CreateFSUID rule to generate an IDENT record in Instream's detail file. Please see **TIB\_fsp\_instream\_<n.n>\_usersguide.pdf** for the layout of the IDENT record.

The rule never generates the same ID twice.

### Activating the rule in your validation profile

To activate the business rule, edit your Instream validation profile (by default, \$fsdeflt.apf in Instream's Bin directory), and set IDENT to 1:

IDENT=1

### Where to place the rule

Place the rule on a mandatory or must use segment where you want the IDENT record to appear. Be sure a previous CreateFSUID business rule has executed to load the variable(s) needed. Precede each Identify rule with its own CreateFSUID record so that the number is never repeated. **Example.** If you want each claim to have its own unique number, place the CreateFSUID and Identify business rules on the CLM segment.

### Format of Parameters

*FSUID CompressedFSUID SystemID*

Where:

|                        |   |
|------------------------|---|
| <i>FSUID</i>           | Variable containing a unique 36-character TIBCO Foresight User ID (FSUID) that was loaded by a CreateFSUID rule.  |
| <i>CompressedFSUID</i> | Optional variable to hold a unique 27-character ID that was loaded by a CreateFSUID rule. Required as a placeholder if you include a SystemID. This value is not used by TIBCO Foresight programs.  |
| <i>SystemID</i>        | Variable or literal in double quotes identifying the system where Instream resides. If omitted, the IDENT record will contain a 1 for SystemID. Transaction Insight® expects this to be 1 for the initial Instream validation. If it is not 1, be sure the value is defined under <b>Settings   External System Setting</b> before attempting to import it into TI. |

**Results:** An IDENT record is placed in the validation detail results file:

IDENT      *line\_num | RuleID | FSUID | SystemID | CompressedFSUID*

**Example 1.** This rule creates a 36-character FSUID in variable FSuniqueID and a 27-character unique ID in variable ShortID.

|                  |  |
|------------------|--|
| External Routine |  |
| Server:          | BusinessRules.Utilities      Function: CreateFSUID |
| Parameters:      | FSuniqueID ShortID                                 |

This Identify rule then creates an IDENT record that contains the FSUID that was created above:

|                  |   |
|------------------|---|
| External Routine |   |
| Server:          | BusinessRules.Utilities    Function: Identify |
| Parameters:      | FSUniqueID                                    |

The record might look like this. Note the “I” after the line number, indicating that this was created with an Identify rule rather than a Match rule.

```
SVALU      31|S009|464|CLM*2235057*460.00***25:B:1*N*A*N*I*P*OA*****1
IDENT      31|I|97c0a5aa-f4a0-11de-a384-f131e23d4046|1|
DTL        31  2300  CLM1332C023          28 5 2          1          10618 3...
```

**Example 2.** Instead of the Identify rule above, you could have used this one, which contains the FSUID, the compressed FSUID and an ID for the external system.

|                  |   |
|------------------|---|
| External Routine |   |
| Server:          | BusinessRules.Utilities    Function: Identify |
| Parameters:      | FSUniqueID ShortID "2"                        |

The record might look like this:

```
SVALU      65|P009|108|CLM*2235057*680.00***25:B:1*Y*A*N*Y*P*OA*****2
IDENT      65|I|97c56a60-f4a0-11de-a384-f131e23d4046|2|IV2MKO7KK08TT8S4U4OU4FA086
```

**Example 3.** These two rules put an IDENT record in the detail results file. This would be suitable for marking the document level for generic X12 or EDIFACT documents that will be sent to TI.

|                |   |
|----------------|---|
| Business Rules |   |
|                | 1. Always call External Routine BusinessRules.Utilities.CreateFSUID FSUniqueID CompressedFSFSUniqueID |
|                | 2. Always call External Routine BusinessRules.Utilities.Identify FSUniqueID                           |

# IdentifierLookup

---

## Instream and HIPAA Validator Desktop

---

Checks a lookup file for the value in `Current_Element` and optionally for the value in a variable assigned with `SetIdentifier`. If the lookup file contains the value(s), validation uses the profile and/or guideline given in the lookup file.

Content-based trading partner automation is described in detail in **TIB\_fsp\_instream\_<n.n>\_tpa.pdf**.

Any `SetIdentifier` value used in this rule must have already been set before this rule executes.

### Format of Parameters

*LookupFile* *SetIDvariable* **Current\_Element**

Where:

|                        |  |
|------------------------|--|
| <i>LookupFile</i>      | Name of the lookup file, including file extension CSV.   |
| <i>SetIDvariable</i>   | A variable assigned with <code>SetIdentifier</code> . It holds one of the values that determine whether the guideline and/or profile will be changed.<br><br>Only include this parameter if two values are involved in the lookup. |
| <b>Current_Element</b> | Literal text (typically); in complex scenarios, this can be another <i>SetIDvariable</i> .   |

**Example 1.** This example uses **two** values. It checks lookup file `MyCBpartnerAutomation.csv` for the values in variable `PayeeN103` and `Current_Element`. If found, it validates using the guideline and/or profile listed in the lookup file.

The screenshot shows a window titled "What Rule to Run". Inside, there is a dropdown menu with "IdentifierLookup" selected, followed by a small icon with "fx". Below this, there is a "Text" input field containing the string "MyCBpartnerAutomation.csv PayeeN103 Current\_Element".

**Example 2.** This example uses **one** value. It checks lookup file `MyCBpartnerAutomation.csv` for the value in `Current_Element`. If found, it validates using the guideline and/or profile listed in the lookup file.

The screenshot shows a window titled "What Rule to Run". Inside, there is a dropdown menu with "IdentifierLookup" selected, followed by a small icon with "fx". Below this, there is a "Text" input field containing the string "MyCBpartnerAutomation.csv Current\_Element".



## InsertIdentifier

---

### Instream

---

Flags an element used in the Java API callback, which is described in **TIB\_fsp\_instream\_<n.n>\_api.pdf**.

#### Format of Parameters

*Variable*

Where:

*Variable*                      A variable to hold this element's value.

**Example.** This example saves the value in the current element in a variable called T1055Ref0406:



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "InsertIdentifier" selected, a small icon to its right, and a "Text" input field containing "T1055Ref0406".

## Match

For internal TIBCO Foresight use.

## MatchAppList

---

### All validators except Analyzer

---

Checks to see if a value is found in an Application Value List, and then take the appropriate action.

#### Format of Parameters

*Value AppList (IfFalseAction) (IfTrueAction)*

Where:

|                      |   |
|----------------------|---|
| <i>Value</i>         | Value to be searched for in the list. This can be a constant in double quotes, a system variable (Current_Element, Current_Date, etc.), or an external variable name.   |
| <i>AppList</i>       | The name of the Application Value List to be searched for <i>Value</i> . This list must already exist in the guideline. Note that <b>MatchAppList</b> works with both explicit values and with regular expression list members. For information about regular expressions, see <b>TIB_fsp_edisim_&lt;n.n&gt;_fseditor.pdf</b> , the section Customizing your Guidelines, Regular Expressions. |
| <i>IfTrueAction</i>  | Specifies a rule to be run if <i>Value</i> is found in, or matches a regular expression in, <i>AppList</i> . Required if you are specifying an IfFalseAction. To do nothing if false, use this:<br><br>(BusinessRules.Utilities DoNothing)  |
| <i>IfFalseAction</i> | Optional. Specifies a rule to be run if <i>Value</i> is not found in <i>AppList</i> , or if <i>AppList</i> does not exist.  |

#### Examples

The examples below assume these two lists:

- **TestList** is an Application Value List that contains:  
VAL1  
VAL2  
VAL3
- **MyPattList** is an Application Value List that contains:  
^REF[A-Z][0-9]\$  
^REF[A-Z][A-Z][0-9]\$  
^REF[A-Z][0-9][0-9]\$  
REFANY
- and these two external variables:  
**ListToUse** is an external variable that contains the string 'TestList'  
**ValToCheck** is an external variable that contains the string 'REF01'

### Example 1

This example causes the TrueAction to be taken because 'VAL1' is found in list 'TestList'.

|                  |  |
|------------------|--|
| What Rule to Run |  |
| MatchApplList    | <input type="checkbox"/> Look-Ahead Rule |
| Text             | "VAL1" "TestList"                        |

### Example 2

This example causes the FalseAction to be taken because ValToCheck contains the value REF01, which is not in the application value list whose name is stored in ListToUse (i.e., TestList).

|                  |  |
|------------------|--|
| What Rule to Run |  |
| MatchApplList    | <input type="checkbox"/> Look-Ahead Rule |
| Text             | ValToCheck ListToUse                     |

### Example 3

This example causes the TrueAction to be taken because 'REFZ9' matches one of the regular expressions in list MyPattList.

|                  |  |
|------------------|--|
| What Rule to Run |  |
| MatchApplList    | <input type="checkbox"/> Look-Ahead Rule |
| Text             | "REFZ9" "MyPattList"                     |

### Example 4

This example causes the the TrueAction to be taken because the value in variable ValToCheck (REF01) matches one of the patterns in MyPattList.

|                  |  |
|------------------|--|
| What Rule to Run |  |
| MatchApplList    | <input type="checkbox"/> Look-Ahead Rule |
| Text             | ValToCheck "MyPattList"                  |

### Example 5

This example causes the FalseAction to be taken because list PattList2 does not exist.

|                  |  |
|------------------|--|
| What Rule to Run |  |
| MatchApplList    | <input type="checkbox"/> Look-Ahead Rule |
| Text             | "REFANY" "MyPattList"                    |

## Normalize

Converts a string into a normalized form, which includes any or all of these:

- Converting the entire string to upper or lower case
- Removing extra spacing
- Removing any non-alphanumeric characters, etc.
- Removing any titles, such as 'Mr.', 'Mrs.', 'Dr.', etc.

This rule is handy for CAQH as well as other types of normalization as defined in the Phase II CORE 258 rule.

### Format of Parameters

*SourceString ResultVar CommandString*

Where:

|                      |  |
|----------------------|--|
| <i>SourceString</i>  | The string to be converted. This can be a literal in double quotes, a system variable like <code>Current_Element</code> or <code>Current_Date</code> , or a variable name.   |
| <i>ResultVar</i>     | The variable to contain the result.  |
| <i>CommandString</i> | <p>A string (variable or literal) containing one or more of the normalization operations, each separated by a comma, as described in <b>CommandString Details</b> below.</p> <p>These will be performed on <i>SourceString</i>, in the order you specify, so the results may differ if the options are in a different order.</p> |

## CommandString Details

This can be one of the following:

| Option             | Result  |
|--------------------|---|
| LC                 | Convert all upper case letters to lower case.   |
| UC                 | Convert all lower case letters to upper case  |
| TRIM <i>limits</i> | <p>Trim all leading and trailing spaces, and replacing any embedded sequences of two or more spaces with a single space.</p> <p>To limit the range of TRIM, append one or more of the following <i>limits</i>:</p> <ul style="list-style-type: none"><li>L Remove leading spaces</li><li>T Remove trailing spaces</li><li>M Replace embedded sequences of two or more spaces in the middle of the string with a single space</li></ul> <p><b>Examples</b></p> <p>TRIM or TRIMLTM Removes all leading and trailing spaces, and all embedded sequences of two or more spaces.</p> <p>TRIMLT Removes all leading and trailing spaces.</p> <p>The order of the suffix codes do not matter. TRIMLT is the same as TRIMTL. Finally, TRIM with no suffix codes is the same as TRIMLTM.</p> |
| RC:NonX12B         | <p>Remove all characters not in the X12 basic character set. This character set includes:</p> <ul style="list-style-type: none"><li>Uppercase letters</li><li>Decimal digits 0–9</li><li>Punctuation Characters</li></ul>   |
| RC:NonX12E         | <p>Remove all characters not in the X12 extended character set. This character set includes:</p> <ul style="list-style-type: none"><li>Uppercase letters</li><li>Lowercase letters</li><li>Decimal digits 0–9</li><li>Punctuation Characters</li></ul> <p>: ; ? = % @ [ ] _ { } \   &lt; &gt; ~ # \$ <i>space</i></p>   |


| Option                      | Result  |
|-----------------------------|---|
| RC:NonUNOA                  | Remove all characters not in the EDIFACT UNOA character set. This character set includes:<br><br>Uppercase letters<br><br>Decimal digits 0–9<br><br>Punctuation Characters  |
| RC:NonUNOB                  | Remove all characters not in the EDIFACT UNOB character set. This character set includes:<br><br>Uppercase letters<br><br>Lowercase letters<br><br>Decimal digits 0–9<br><br>Punctuation Characters   |
| RC:NonAN                    | Remove all characters that are not alphanumeric (not an uppercase or lowercase letter or a digit)   |
| RC:LoCC                     | Remove all control characters that have an ASCII value of 1 through 31  |
| RC:HiCC                     | Remove all control characters that have an ASCII value of 128 through 255   |
| RC:AllCC                    | Remove all control characters that have an ASCII value of 1 through 31 or 128 through 255   |
| RC:List'ccc'                | Remove all characters listed in <i>ccc</i> .<br><br><b>Example</b><br>This removes all colons, commas, and periods: RC:List':,.''<br><br>To remove a single quote character, use two consecutive single quotes in the 'ccc' string.<br><br><b>Example</b><br>This removes all double quote and single quote characters:<br>RC: <b>List'</b> ''' |
| RW:CAQH                     | Removes all occurrences of the following titles from the front and/or end of SourceString, as specified in section 4.2.2 of the CAQH CORE document:<br><br>JR SR I II III IV V RN MD MR MS DR MRS PHD REV<br>ESQ  |
| RW:List' <i>w1 w2 ...</i> ' | Removes all occurrences of the words specified by <i>w1, w2 ....</i><br><br><i>w1 w2 ...</i> is a list of words, with each word separated by a space. Letter case is not significant.<br><br>Words will be removed if they are found at the beginning or end of   |

| Option | Result  |
|--------|---|
|        | <p><i>SourceString</i>, and separated from the rest of the string by a space, comma, or forward slash character. If any word is immediately followed by a period, the period will also be removed.</p> <p>To include a single quote character in a word, use two consecutive single quotes in the 'w1 w2 ...' string.</p> |

## Examples

The character '·' in these examples represents a space

### Example 1


| What Rule to Run                    |   |  |
|-------------------------------------|---|--|
| Normalize                           |  | <input type="checkbox"/> Look-Ahead Rule |
| <input type="button" value="Text"/> | Parameter Name  | Parameter Value                          |
|                                     | SourceStr   | " Cat {feline}! "                        |
|                                     | DestVar   | SpeciesVar                               |
|                                     | <b>CommandString</b>  | "UC,RC:NONX12B"                          |

This puts ··CAT··FELINE!·· (with leading and trailing spaces remaining) into the variable SpeciesVar because:

- UC converts to upper case.
- RC:NONX12B remove all characters not in the Basic X12 character set... in this case, the curly brackets. The spaces and exclamation point remain.

### Example 2

Assume that the current element contains Dr. Fred Schultz .

| What Rule to Run                    |   |  |
|-------------------------------------|---|--|
| Normalize                           |  | <input type="checkbox"/> Look-Ahead Rule |
| <input type="button" value="Text"/> | Parameter Name  | Parameter Value                          |
|                                     | SourceStr   | Current_Element                          |
|                                     | DestVar   | NormNamevar                              |
|                                     | <b>CommandString</b>  | "UC,RW:CAQH"                             |

This puts FRED SHULTZ into variable NormNamevar because:

- UC converts to upper case.
- RW:CAQH removes all CAQH titles.

### Example 3

This shows how the sequence of operations can affect the result.

Assume that variable VarDat contains This·is·a·Test!··· .

```

Normalize VarDat NormResult1var "UC,RC:NONX12B"
Normalize VarDat NormResult2var "RC:NONX12B,UC"

```

The first rule causes `THIS ·IS ·A ·TEST! ··` to be put into variable `NormResult1var` because:


- UC converts to upper case.
- RC:NONX12B removes all characters not in the Basic X12 character set.

However, the second rule causes `T ·· ·T! ··` to be put into variable `NormResult2var` because:

- RC:NONX12B remove all characters not in the Basic X12 character set.
- UC converts to upper case.
- Since lower case letters are not in the X12 Basic Character Set, they all get removed.

#### Example 4


Assume that the current element contains `Rev. Raymond A. Ratchet, Esq, PhD`

| What Rule to Run |   |  |
|------------------|---|--|
| Normalize        |  | <input type="checkbox"/> Look-Ahead Rule |
| <b>Text</b>      | Parameter Name  | Parameter Value                          |
|                  | SourceStr   | Current_Element                          |
|                  | <b>DestVar</b>  | NormNameVar                              |
|                  | CommandString   | "UC,RW:CAQH,RC:NONAN"                    |

This causes `RAYMOND A RATCHET` to be put into variable `NormNameVar` because:

- UC converts to upper case.
- RW:CAQH removes all CAQH titles like Rev, Esq, and PhD
- RC:NONAN remove all non-alphanumeric characters like the punctuation

#### Example 5

| What Rule to Run |   |  |
|------------------|---|--|
| Normalize        |  | <input type="checkbox"/> Look-Ahead Rule |
| <b>Text</b>      | Parameter Name  | Parameter Value                          |
|                  | SourceStr   | "Dr.NoSpace"                             |
|                  | DestVar   | DrNameVar                                |
|                  | <b>CommandString</b>  | "UC,RW:CAQH"                             |

This causes `DR.NOSPACE` to be put into `DrNameVar` because:

- UC converts to upper case.
- RW:CAQH removes all prefixes and suffixes in the CAQH title list. However, there is no space, comma, or forward slash between the DR. and the rest of the string so it isn't removed.



## Numbers

Adds, subtracts, multiplies, and divides numbers and put the output into a variable. Maximum precision is 8 decimal places.

### Format of Parameters

*VarA Operand VarB VarOut*

Where:

|                      |  |            |
|----------------------|--|------------|
| <i>VarA and VarB</i> | A variable, <code>Current_Element</code> , or a literal surrounded with double quotes. If <code>Current_Element</code> is used and is empty, processing stops on the rule. If the variable does not represent a numeric, an error message is issued. |            |
| <i>Operand</i>       | +  | (plus)     |
|                      | -  | (minus)    |
|                      | *  | (multiply) |
|                      | /  | (divide)   |
| <i>VarOut</i>        | The variable to hold the result. If <i>VarOut</i> has not yet been defined, it is created. If it exists, its contents are overwritten.   |            |

**Example.** This example displays a message if the line item value exceeds \$1,000,000 in a purchase order.

On the PO102 (quantity), capture the quantity:

What Rule to Run  
SetVar [v] [fx]  
Text LineItemQty

On the PO103 (unit price), multiply the unit price by the quantity:

What Rule to Run  
Numbers [v] [fx]  
Text Current\_Element \* LineItemQty LineItemTot

Also on the PO103 (after the previous rule), display a message if the total is more than \$1,000,000:

What Rule to Run  
CompareNumeric [v] [fx]  
Text LineItemTot GT "1000000" (BusinessRules.Utilities DisplayErrorByNumber 0 0 "Line item total is greater than 1,000,000")

## OracleLookup and OracleLookupWithDate

---

### AIX Instream

---

These two business rules are available by request. Please contact TIBCO Foresight Technical Support.

Performs a lookup from an Oracle database and executes a business rule if it is false.

**OracleLookupWithDate** Use if the SQL statement or stored procedure includes a date calculation.

**OracleLookup** Use if the SQL statement or stored procedure does not include a date calculation.

#### Format of Parameters

*"SQLstatement" (IfFalseAction)*

*or*

*"StoredProcedure(procedure\_name)" Parameter\_for\_Procedure (IfFalseAction)*

Where:

*SQLstatement* A SQL statement.

Business rule variables within the statement must be set already with a SetVar or similar business rule.

Enclose business rule variables and Current\_Element in single quotes:

``%ProviderIdNumber%``

``%Current_Element%``

*IfFalseAction* A business rule to execute if the SQL statement is false.

**StoredProcedure** Literal text.


*procedure\_name* Name of the Oracle procedure.

*Parameter\_for\_Procedure*

One parameter to pass to the Oracle procedure.

**Example 1.** This example uses OracleLookup to check for a provider ID. It includes the SQL statement. If it is not found by the lookup, then an error message is issued.

What Rule to Run

OracleLookup 

Text "select 1 from Table1 where ProviderID='%ProviderIDNumber%' (BusinessRules.Utilities.DisplayErrorByNumber 0 0 Provider ID Number is not valid)"

**Example 2.** This example uses OracleLookupWithDate to check for a provider ID. If it is not found by the lookup, then an error message is issued.

What Rule to Run

OracleLookupWithDate

Text

"select 1 from Table1 where ProviderID='%ClaimProvNumber%' AND TO\_DATE (%ClaimServiceDate%, 'YYMMDD') >=StartDate"  
(BusinessRules.Utilities DisplayErrorByNumber 0 0 Provider ID Number is not valid)

**Example 3.** This example uses an Oracle lookup to execute a stored procedure that does not contain any date calculations.

What Rule to Run

OracleLookup

Text

"StoredProcedure(npi\_VerifyNPI\_sp)" MemberIDNumber (BusinessRules.Utilities DisplayErrorByNumber 0 0 Member ID Number is not valid)]

**Example 4.** This example uses an Oracle lookup at the end of a loop if two conditions are true.

What Rule to Run

SetLoopPostInstanceExit

Text

2010AB BusinessRules.Variable CompareString Have2010ABREFG21B EQ "OFF" (BusinessRules.Variable CompareString Need2010ABREFG21B EQ "ON" (BusinessRules.Utilities OracleLookup "select 1 from db1.vw\_instream\_verify\_npi a where a.npi='%2010ABNM109NPIN%' (BusinessRules.Utilities DisplayErrorByNumber 32054)

## OutputCTX

---

### Instream

---

Creates a CTX record in the detail file. This record is used by Response Generator to create a CTX segment in the 999.

During Instream validation, TIBCO Foresight 5010 837 guidelines generate CTX segments under conditions specified in the HIPAA Implementation Guides.

#### Format of Parameters

*CTXvar*

Where:

|               |   |
|---------------|---|
| <i>CTXvar</i> | Variable containing the contents of the CTX record. This is usually created from a SaveCurrentSegment rule, a GetValueFromSegment rule, and a BuildString rule. |
|---------------|---|

**Example.** This rule creates a CTX record from the contents of the CTXOUTSTRING variable.

```
BusinessRules.Utilities.OutputCTX:CTXOUTSTRING
```

#### Rules required to create your own CTX record

A number of rules are required to create your own CTX record. Please see

**TIB\_fsp\_instream\_<n.n>\_respgen.pdf** for details.

## ReplaceChars

---

### All validation programs

---

Performs any or all of these and stores the result in a variable:

- Replaces characters that are not in the:
  - X12 basic character set
  - X12 extended character set
  - EDIFACT UNOA character set
  - EDIFACT UNOB character set
- Replaces characters that are:
  - not alphanumeric
  - control characters
- Replaces characters that you specify.

## Format of Parameters

*SourceString ResultVar CharsToReplace ReplacementChar*

Where:

|                        |  |                    |  |      |                                |
|------------------------|--|--------------------|--|------|--------------------------------|
| <i>SourceStr</i>       | The string to be changed. This can be a string constant in double quotes, a variable, or a system variable like <code>Current_Element</code> or <code>Current_Date</code> .  |                    |  |      |                                |
| <i>ResultVar</i>       | The variable to contain the result. This can be the same variable name as specified in <b>SourceString</b> , if desired.   |                    |  |      |                                |
| <i>CharsToReplace</i>  | A string describing which characters to replace.<br><br>See CharstoReplace on page 190.  |                    |  |      |                                |
| <i>ReplacementChar</i> | A string identifying the character that is to be used to replace all matched characters in <i>CharsToReplace</i> .<br><br>This can be: <table><tr><td>A single character</td><td>Replace each matched character with this character. Examples: "X" or " "</td></tr><tr><td>NONE</td><td>Remove each matched character.</td></tr></table> | A single character | Replace each matched character with this character. Examples: "X" or " " | NONE | Remove each matched character. |
| A single character     | Replace each matched character with this character. Examples: "X" or " "   |                    |  |      |                                |
| NONE                   | Remove each matched character.   |                    |  |      |                                |

## CharstoReplace

---

This can be one of the following:

| Option  | Result  |
|---------|---|
| NonX12B | Replace all characters not in the X12 basic character set. This character set includes:<br><br>Uppercase letters A–Z<br><br>Decimal digits 0–9<br><br>Punctuation Characters<br><br>! " & ' ( ) * + , - . / : ;   |
| NonX12E | Replace all characters not in the X12 extended character set. This character set includes:<br><br>Uppercase letters A–Z<br><br>Lowercase letters a–z<br><br>Decimal digits 0–9<br><br>Punctuation Characters<br><br>; ? = % @ [ ] _ { } \   <<br>> ~ # \$ space |

| Option    | Result   |
|-----------|--|
| NonUNOA   | <p>Replace all characters not in the EDIFACT UNOA character set. This character set includes:</p> <p>Uppercase letters A–Z</p> <p>Decimal digits 0–9</p> <p>Punctuation Characters</p> <p><code>. , - ( ) / = space</code></p>   |
| NonUNOB   | <p>Replace all characters not in the EDIFACT UNOB character set. This character set includes:</p> <p>Uppercase letters A–Z</p> <p>Lowercase letters a–z</p> <p>Decimal digits 0–9</p> <p>Punctuation Characters</p> <p><code>! " % &amp; * space</code></p>  |
| NonAN     | Replace all characters that are not alphanumeric (not an uppercase or lowercase letter or a digit)   |
| LoCC      | Replace all control characters that have an ASCII value of 1 through 31  |
| HiCC      | Replace all control characters that have an ASCII value of 128 through 255   |
| AllCC     | Replace all control characters that have an ASCII value of 1 through 31 or 128 through 255   |
| List'ccc' | <p>Replace all characters listed in <i>ccc</i>.</p> <p><b>Example</b></p> <p>This removes all colons, commas, and periods: List':,.'</p> <p>To remove a single quote character, use two consecutive single quotes in the 'ccc' string.</p> <p><b>Example</b></p> <p>This removes all double quote and single quote characters: List'""''</p> |

### Example 1

This example replaces all lowercase “c” characters with a capital C.

Assume that the current element contains `col. John Crocker`

SubmitterVar would then contain `Col. John CroCker`.

What Rule to Run

ReplaceChars  ☐ Look-Ahead Rule

| Parameter Name         | Parameter Value |
|------------------------|-----------------|
| SourceStr              | Current_Element |
| DestVar                | SubmitterVar    |
| CharsToReplace         | "c"             |
| <b>ReplacementChar</b> | <b>"C"</b>      |

### Example 2

This example replaces any characters that are not in the X12 basic character set with uppercase X so that SpeciesVar contains TXXX XX X XXX .

What Rule to Run

ReplaceChars  ☐ Look-Ahead Rule

| Parameter Name         | Parameter Value |
|------------------------|-----------------|
| SourceStr              | "This is a cat" |
| DestVar                | SpeciesVar      |
| CharsToReplace         | "NONX12B"       |
| <b>ReplacementChar</b> | <b>"X"</b>      |

### Example 3

Assume that the current element contains P . O . Box #1234 .

This example removes (not replaces) non-alphanumeric characters so that variable AddressVar contains P O Box 1234 .

What Rule to Run

ReplaceChars  ☐ Look-Ahead Rule

| Parameter Name         | Parameter Value |
|------------------------|-----------------|
| SourceStr              | Current_Element |
| DestVar                | AddressVar      |
| CharsToReplace         | "NONAN"         |
| <b>ReplacementChar</b> | <b>"NONE"</b>   |

### Example 4

Assume that:

- Variable PatPhoneVar contains (614) 431-2345
- Variable ReplCharsVar contains "List'() - '"  
Note the space between ) and - '
- Variable ReplWithVar contains none

What Rule to Run

ReplaceChars  ☐ Look-Ahead Rule

| Parameter Name         | Parameter Value |
|------------------------|-----------------|
| SourceStr              | PatPhoneVar     |
| DestVar                | ReplCharsVar    |
| CharsToReplace         | "List'() -"     |
| <b>ReplacementChar</b> | ReplWithVar     |

NewPatPhoneVar will contain 6144312345 .

### Example 5

This causes KAVER Corporation# to be put into variable CompanyNameVar.

What Rule to Run

ReplaceChars  ☐ Look-Ahead Rule

| Parameter Name         | Parameter Value      |
|------------------------|----------------------|
| SourceStr              | "KAVER Corporation®" |
| DestVar                | CompanyNameVar       |
| CharsToReplace         | "HiCC"               |
| <b>ReplacementChar</b> | "#"                  |



## ReplaceString

---

### All validation programs

---

Replaces one value with another and places the result in a variable.

#### Format of Parameters

*SourceStr* {**ALL**} *OldString* *NewString* {*DestVar*}

Where:

|                  |  |
|------------------|--|
| <i>SourceStr</i> | The location of the string to be changed. This can be a string constant in double quotes, a variable, <i>Current_Element</i> , or <i>Current_Date</i> .<br><br>If this parameter is not a variable, then <i>DestVar</i> is required.   |
| <i>OldString</i> | The substring to be replaced. The first occurrence (or all occurrences within <i>SourceStr</i> , if the <b>ALL</b> parameter is included before this one) will be replaced with <i>NewString</i> . This parameter can be a variable, <i>Current_Element</i> , <i>Current_Date</i> , or a string constant in double quotes. |
| <i>NewString</i> | The substring to replace <i>OldString</i> in the <i>SourceStr</i> . This can be a variable, <i>Current_Element</i> , <i>Current_Date</i> , or a string constant in double quotes.  |
| <i>DestVar</i>   | Required to hold the result if <i>SourceStr</i> is not a variable. If omitted, the result is stored back in the <i>SourceStr</i> variable.   |

**Example 1.** This example removes a single quote by replacing ' with nothing. It places the result in the variable *Patient\_name*. This removes quotes from values like O'Neill.



What Rule to Run

ReplaceString [v] [fx]

Text Current\_Element'''Patient\_name'

The parameters are:

*Current\_Element* " ' " " " *Patient\_name*.

Value before: O'Neill

Value after: O'Neill

Once the quote-less value is in *Patient\_name*, you can use it in other business rules such as ODBC rules.

**Example 2.** This example removes all hyphens within `Current_Element` by replacing hyphens with nothing. It places the result in the variable `Phone_num`.

What Rule to Run

ReplaceString

Text: Current\_Element ALL "-" "" Phone\_num

The parameters are:

`Current_Element ALL "-" "" Phone_num`

Value before: 614-555-1212

Value after: 6145551212

**Example 3.** This example replaces a single quote with a double quote.

What Rule to Run

ReplaceString

Text: PatientLastName "'" "" Patient\_name

The parameters are:

`PatientLastName "'" "" Patient_name`

Value before: O'Neill

Value after: O"Neill

## SetCheckCTT and SetCheckCTTCount

---

### Instream and HIPAA Validator Desktop, X12 only

---

For Analyzer checking, see [CheckCTT](#) on page [227](#).

SetCheckCTT checks the value in the CTT-01 (number of line items) and the CTT-02 (hash total).

SetCheckCTTCount checks the CTT-02.

For best results, place the rule on the ST segment. It has to appear before anything that it counts.

See the explanation under [CheckCTT](#) on page [227](#).

#### Format of Parameters

*These rules have no parameters*

#### Examples:

The image shows two examples of the 'What Rule to Run' dialog box. Each dialog has a title bar 'What Rule to Run', a dropdown menu, a formula icon (fx), a 'Text' button, and a text input field.

Example 1: The dropdown menu is set to 'SetCheckCTT'.

Example 2: The dropdown menu is set to 'SetCheckCTTCount'.

## SetIdentifier

---

### Instream and HIPAA Validator Desktop

---

Flags an element used in content-based trading partner automation, which is described in detail in **TIB\_fsp\_instream\_<n.n>\_tpa.pdf**.

This rule goes with the IdentifierLookup rule and must precede it.

This rule is only needed if *two* elements are used with content-based trading partner automation.

#### Format of Parameters

*SetIDvariable*

Where:

|                      |   |
|----------------------|---|
| <i>SetIDvariable</i> | A variable to hold this element's value for use in an IdentifierLookup rule for content-based trading partner automation. |
|----------------------|---|

**Example.** This example puts the value of the current element into variable PayeeN103 for use in an IdentifierLookup rule:



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "SetIdentifier" selected. To the right of the dropdown is a button with a function symbol (fx). Below the dropdown is a "Text" tab, which is active. The text field next to the "Text" tab contains the variable "PayeeN103".

## SubString

---

### All validation programs

---

Extracts a portion of one string into another.

#### Format of Parameters

*DestStr SourceStr StartIndex EndIndex*

Where:

|                   |  |
|-------------------|--|
| <i>DestStr</i>    | The variable that is to hold the extracted portion of the <i>SourceStr</i> .   |
| <i>SourceStr</i>  | The variable that holds the characters to be extracted. This can be a variable or <i>Current_Element</i> . If <i>Current_Element</i> is used and is empty, processing of the rule stops.   |
| <i>StartIndex</i> | The position of the first character to be extracted from the <i>SourceStr</i> . This can be a number or a variable containing a number. If the <i>StartIndex</i> is less than 1, it is set to 1.   |
| <i>EndIndex</i>   | The position of the last character to be extracted from the <i>SourceStr</i> . This can be a number or a variable containing a number. If the <i>StartIndex</i> is greater than the <i>EndIndex</i> , an error message is displayed. If the <i>EndIndex</i> is greater than the length of <i>SourceStr</i> , the <i>EndIndex</i> will be set to the number of characters in <i>SourceStr</i> . |

**Example 1.** This example extracts the first 8 characters of the value in the current element into a variable called *ShipDate* that can be used in a subsequent rule.

What Rule to Run

SubString

Text ShipDate Current\_Element 1 8

**Example 2.** This example extracts the first characters of the value in the current element into a variable called *ShipDate* that can be used in a subsequent rule. Since the *EndIndex* is not an integer, it is treated as a variable and should contain an integer that will serve as the ending point of the substring.

What Rule to Run

SubString

Text ShipDate Current\_Element 1 DateLength

# Trim

---

## All validation programs

---

Removes specified characters from the right or left of a value and places the result into a variable.

### Format of Parameters

*TrimLocation Characters Value ResultVar*

Where:

|                     |   |
|---------------------|---|
| <i>TrimLocation</i> | Where to trim: the literal LEFT, RIGHT, or BOTH.  |
| <i>Characters</i>   | One or more characters to trim, surrounded by double quotes.<br><br>If you supply one character, all of that character will be trimmed from the location you chose. Example: "0" removes all leading or trailing zeros.<br><br>If you supply multiple characters, all sets of them will be trimmed from the location you chose. Example: "12" removes all leading or trailing sets of 12. |
| <i>Value</i>        | The value to be trimmed. This can be CURRENT_ELEMENT or a variable.   |
| <i>ResultVar</i>    | A variable to hold the trimmed result.  |

**Example.** This rule is placed on the ISA08 to trim trailing spaces. It puts the trimmed result in variable GLOBAL\_ISA08.

What Rule to Run

Trim

Text: RIGHT "" Current\_Element GLOBAL\_ISA08

This rule uses GLOBAL\_ISA08 in a CodeLookup to see if the ISA08 is valid:

What Rule to Run

FindUserCode

Text: USERISA08Receiver227U GLOBAL\_ISA08 (BusinessRules.Utilities DisplayErrorByNumber 32636)

# TrimWhitespace

---

## All validation programs

---

Removes leading and trailing whitespace characters (spaces and tabs) from a value and replaces any sequences of two or more whitespace characters within the value with a single occurrence.

### Format of Parameters

*InputValue* *OutputVar* *Options*

Where:

|                   |  |
|-------------------|--|
| <i>InputValue</i> | The value to be trimmed. This can be a constant in double quotes, an internal variable name (Current_Element, Current_Date, etc.), or an external variable name.   |
| <i>OutputVar</i>  | An external variable name where the result is to be stored.  |
| <i>Option</i>     | The character string containing one or more of the following characters (these options can be combined):<br><br><b>L</b> Remove any leading whitespace characters<br><b>T</b> Remove any trailing whitespace characters<br><b>M</b> Replace any strings of two or more whitespace characters within <i>InputValue</i> with a single space<br><br>If Options is not specified then <b>LTM</b> is assumed. |

### Examples

In the following examples, ‘ ’ represents a space and ‘>’ represents a tab.

#### Example 1

This example causes the string ‘Test>Value’ to be stored into variable RESULTVAR. Because no Option was specified, ‘LTM’ is assumed so leading and trailing whitespace characters are removed and duplicate whitespace sequences within the string are replaced by a single space.

What Rule to Run

TrimWhitespace ☐ Look-Ahead Rule

**Text** \"..Test..Value..\" RESULTVAR «Option»

## Example 2

This example causes the string 'Post•Office•Box•1234' to be stored into variable TRIMMEDADDR1. The 'space tab space' sequence is handled as a string of three whitespace characters and is replaced by a single space. (ADDR1 = "»Post•»•Office••Box••1234»")

|                  |  |
|------------------|--|
| What Rule to Run |  |
| TrimWhitespace ▼ | <input type="checkbox"/> Look-Ahead Rule |
| Text             | ADDR1 TRIMMEDADDR1                       |

## Examples 3

This example causes the string 'Post•»•Office••Box••1234' to be stored into variable TRIMMEDADDR1. Because *Option LT* was used, leading and trailing whitespace characters are removed. (ADDR1 = "»Post•»•Office••Box••1234»")

|                  |  |
|------------------|--|
| What Rule to Run |  |
| TrimWhitespace ▼ | <input type="checkbox"/> Look-Ahead Rule |
| Text             | ADDR1 TRIMMEDADDR1 "LT"                  |



# Variable Business Rules

## SetLocalVariable

---

### All validation programs

---

Used to explicitly set the contents of a local variable.

#### Format of Parameters

*LocalVariableName* *Value*

Where:

*LocalVariableName*    The variable name in double quotes or an external variable name containing the local variable name.

*Value*                    The value to be stored in *LocalVariableName* in double quotes or an external variable name containing the value.

#### Examples

This example causes the string ‘True’ to be stored into local variable UseListA

The screenshot shows a configuration window titled "What Rule to Run". It contains a dropdown menu with "SetLocalVar" selected, a function icon (fx), and an unchecked checkbox labeled "Look-Ahead Rule". Below this is a text input field with a "Text" button on the left and the content '"UseListA" "True"'.

This example causes the string ‘SMITH’ to be stored into local variable LVar2 . (Where Current element = “SMITH”)

The screenshot shows a configuration window titled "What Rule to Run". It contains a dropdown menu with "SetLocalVar" selected, a function icon (fx), and an unchecked checkbox labeled "Look-Ahead Rule". Below this is a text input field with a "Text" button on the left and the content '"LVar2" Current\_Element'.

This example looks up external variable LVARNAME and uses its contents as the name of the local variable to set to “1”.

The screenshot shows a configuration window titled "What Rule to Run". It contains a dropdown menu with "SetLocalVar" selected, a function icon (fx), and an unchecked checkbox labeled "Look-Ahead Rule". Below this is a text input field with a "Text" button on the left and the content 'LVARNAME "1"'.

# SetVar

---

## All validation programs

---

Sets a BusinessRules.Variable to the contents of the current element or to a passed value. See Appendix A: Variables on page 239 for an overview of variables.

### Format of Parameters

*VarName VarValue*

Where:

|                 |   |
|-----------------|---|
| <i>VarName</i>  | Name you are assigning to that variable. If the variable exists, the current contents are overwritten with <i>VarValue</i> . The name can be any length, with no special characters or spaces. It is case-sensitive.                |
| <i>VarValue</i> | Optional. Value being assigned to that variable. This can be another variable, Current_Element, a literal in double quotes, or Current_Date if appropriate. If <i>VarValue</i> is omitted, the value in Current_Element is assumed. |

**Example.** This example assigns the variable name 2010AAN402State to the contents of the current element.

What Rule to Run

SetVar

Text 2010AAN402State Current\_Element

The variable 2010AAN402State can then be used in the parameter for a rule like the one shown below, which checks the zip code in the current element to see if it is valid for the state that is in the variable 2010AAN402State.

What Rule to Run

ValidateZipState

Text Current\_Element 2010AAN402State

## AddVar

---

### All validation programs

---

Adds a value to the current value of a variable. This can keep a running total for use in other rules. Maximum precision is 8 decimal places.

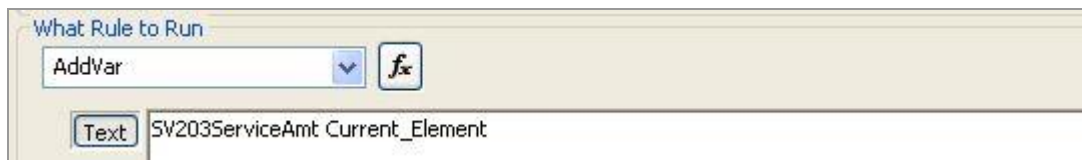
#### Format of Parameters

*VarName* *VarValue*

Where:

|                 |   |
|-----------------|---|
| <i>VarName</i>  | The variable to hold the accumulated values. If <i>VarName</i> has not yet been defined, it is created.   |
| <i>VarValue</i> | Optional. The amount being added to the variable. This can be a variable, <code>Current_Element</code> , or a literal in double quotes. If omitted, the value of <code>Current_Element</code> is assumed. |

**Example.** This rule keeps a running total of the service line amounts in each repetition of the SV2 and enclosing CLM loops. It is applied to each service line amount element.



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "AddVar" selected, a small icon to its right, and a "Text" button below it. The text "SV203ServiceAmt Current\_Element" is entered in the input field.

For examples of how to use the results of an AddVar, see these examples:

- CompareString on page [209](#)
- Example 2: Using Rules in Loops on page [268](#)
- Example 3: Adding and Comparing Numeric Values on page [269](#).

# Divide

---

## All validation programs

---

Divides one value by another and puts the result in a variable.

### Format of Parameters


*Dividend Divisor NumOfdec outVar*

Where:

|                 |   |
|-----------------|---|
| <i>Dividend</i> | The value to be divided. It can be a literal, variable, or Current_Element.                         |
| <i>Divisor</i>  | The value that the dividend is being divided by. If this is a literal, enclose it in double quotes. |
| <i>NumOfdec</i> | Number of decimal places to use in the result. Do not put quotes around this integer.               |
| <i>outVar</i>   | Variable to hold the result.  |

**Example.** This rule divides the value in the current element by 100 and puts the result in variable DollarVar. The result has two decimal places.

What Rule to Run

Divide 

**Text** Current\_Element / 100' 2 DollarVar

## DumpVars

---

### All validation programs except Analyzer

---

**You must be set up for debugging** before you can use DumpVars. See page [287](#).

Shows External Routine variables and their current values. It does not show local variables.

Place the rule on the segment or element where you would like Validator to display the variables and values. During validation, you can view these messages or suppress them.

#### Format of Parameters

*Variable Variable Variable ...*

Where:

*Variable*

A variable that is to have its current value displayed. Each additional variable can be separated by a space.

If no variable is specified, all variables are displayed.

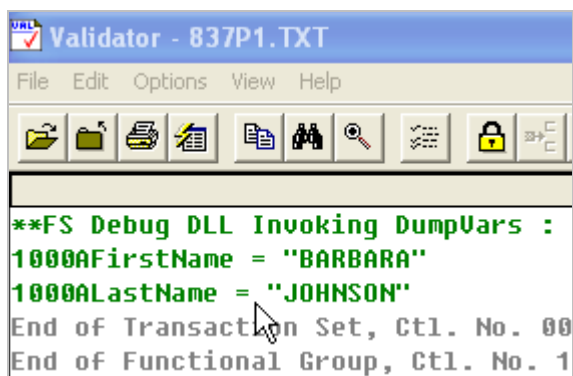
Specific array entries may not be dumped, though the entire array can be. For example, `ListTotals("1")` is invalid, but `ListTotals` is OK, and causes each member to be dumped.

To view the dumped variables, see page [287](#).

**Example.** This rule displays the current contents of two variables:



The output in EDISIM Validator:



# Balance

---

## All validation programs

---

This will validate mathematical operations on variables.

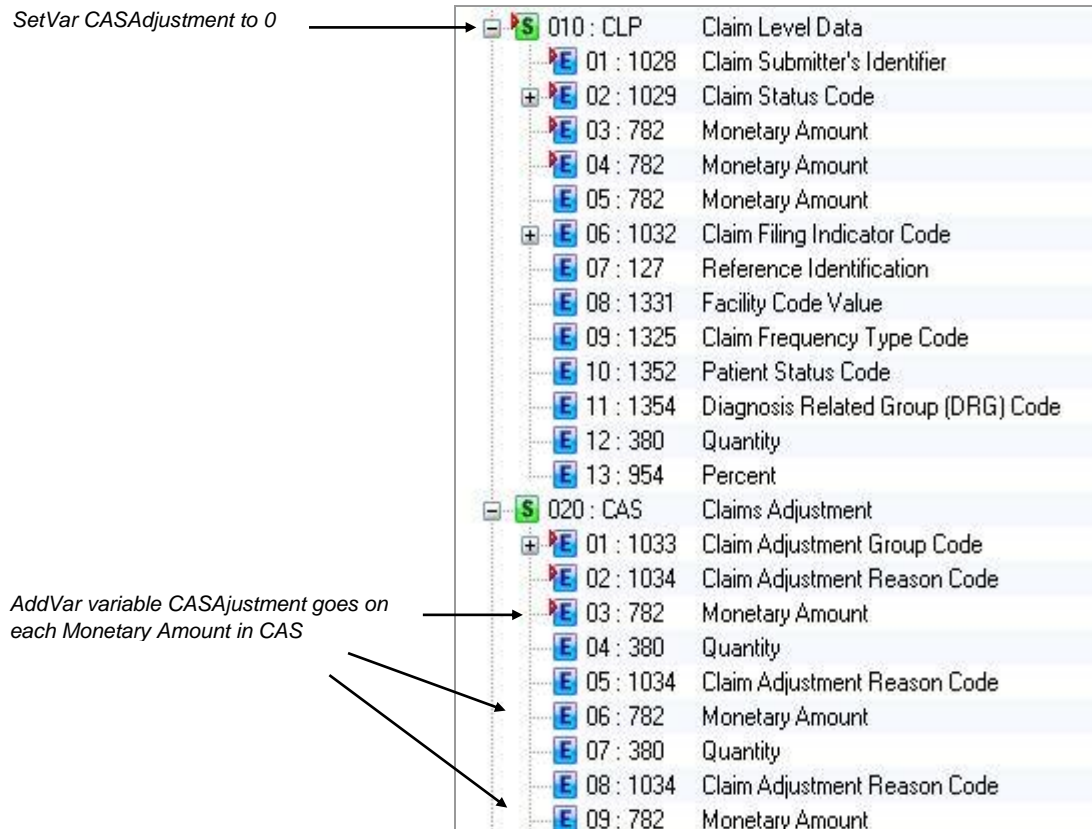
### Format of Parameters

*VarA Operand VarB = VarC IfFalseAction*

### Where:

|                        |  |
|------------------------|--|
| <i>VarA</i>            | A variable, <code>Current_Element</code> , or a literal in double quotes. If <code>Current_Element</code> is used and is empty, processing stops for the rule. |
| <i>Operand</i>         | One of these: - + * /  |
| <i>VarB</i>            | A variable, <code>Current_Element</code> , or a literal in double quotes. If <code>Current_Element</code> is used and is empty, processing stops for the rule. |
| =                      | Equal sign. Put one space before and one space after the equal sign.   |
| <i>VarC</i>            | A variable, <code>Current_Element</code> , or a literal in double quotes. If <code>Current_Element</code> is used and is empty, processing stops for the rule. |
| <i>(IfFalseAction)</i> | Optional. Executed if the mathematical operation is FALSE. If omitted, a generic message is displayed.   |

**Example.** This rule adds up adjustments and total paid amount to ensure that they equal submitted charges for each repetition of the CLP loop in an 835. If not, an error message displays.



1. Use SetVar to set the variables shown above.
2. Use the same AddVar name **CASAdjustment** on the CAS03, CAS06, CAS09, CAS12, CAS15, and CAS18. This sums all of the adjustments for a repetition of the loop.
3. On the CLP Segment (which begins the loop), set the **CASAdjustment** to 0. This starts the calculation at zero for each repetition of the loop:

What Rule to Run

SetVar

Text CASAdjustment "0"

4. To balance at the end of each repetition of the loop, go to the ST segment and use SetLoopPostInstanceExit and Balance functions:

What Rule to Run

SetLoopPostInstanceExit

Text 2100 BusinessRules.Variable Balance CLP03SubmittedCharges - CASAdjustment = CLP04TotalPaidAmount (BusinessRules.Utilities.DisplayErrorByNumber 32215)

This gives error message 32215 if the calculation is not true at the end of loop 2100, the CLP loop.

## CompareString and CompareStringNoCase

---

### All validation programs

---

Compares two values as strings based on the operand, and executes an action if the comparison is true.

CompareString requires the value to match exactly in order to be true; CompareStringNoCase does not consider capitalization when comparing the values.

#### Format of Parameters

*VarA Operand VarB (IfTrueAction)*

Where:

|                       |   |
|-----------------------|---|
| <i>VarA</i>           | A BusinessRules.Variable, Current_Element, or a literal surrounded with double quotes. If Current_Element is used and is empty, processing of the rule stops.               |
| <i>Operand</i>        | EQ, NE, GT, GE, LT, or LE.  |
| <i>VarB</i>           | A BusinessRules.Variable, Current_Element, Current_Date, or a literal surrounded with double quotes. If Current_Element is used and is empty, processing of the rule stops. |
| <i>(IfTrueAction)</i> | Optional. The action to be executed if the comparison is true. If omitted, a generic message is displayed if the comparison is true.  |

**Example.** This example checks the PER segment and issues an error message if a telephone number starts with 1. This involved these rules:

- On the PER03 (the qualifier) - Use SetVar to set up variable PER03Submitter.
- On the PER04 (the number itself):

Set up a rule that checks the qualifier to see if it is “TE” and, if so, places the first character of the PER04 into a variable that we call TEFIRSTDigit.

Issue an error message if the variable contains a 1.

Message 32211 is a custom message in file CustomerFSBRERRS.txt.



## CompareNstring

Compares parts of two values as strings based on the operand, and executes an action if the comparison is true.

CompareNstring requires the specified parts of the strings to match exactly, including their case, in order to be true.

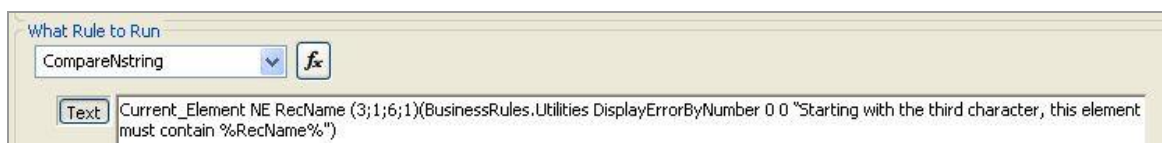
### Format of Parameters

*VarA Operand VarB (startVarA;startVarB;length;case) (IfTrueAction)*

Where:

|                       |  |
|-----------------------|--|
| <i>VarA</i>           | String to compare. A BusinessRules.Variable, Current_Element, or a literal surrounded with double quotes. If Current_Element is used and is empty, processing of the rule stops.               |
| <i>Operand</i>        | EQ, NE, GT, GE, LT, or LE.   |
| <i>VarB</i>           | String to compare. A BusinessRules.Variable, Current_Element, Current_Date, or a literal surrounded with double quotes. If Current_Element is used and is empty, processing of the rule stops. |
| <i>startVarA</i>      | Position where comparison starts for VarA.   |
| <i>startVar</i>       | Position where comparison starts for VarB  |
| <i>length</i>         | Number of characters to compare.   |
| <i>case</i>           | Whether to consider the case when comparing:<br><br>0            (default) Comparison is case sensitive<br><br>1            Comparison ignores the case  |
| <i>(IfTrueAction)</i> | Optional. The action to be executed if the comparison is true. If omitted, a generic message is displayed if the comparison is true.   |

**Example.** This example compares 6 characters of the current element, starting with position 3, to the first 6 characters of the variable RecName. The comparison is case-sensitive. If they do not match, an error message is displayed.



Assume:

Current\_Element = A B C 1 2 3 4 5 6 7 8 9

RecName = 9 9 9 1 2 3

The underlined characters will be compared. Since they do not match, this error message will display:

"Starting with the third character, this element must contain 999123"

## CompareNumeric

---

### All validation programs

---

Compares two values as numeric and executes an action if the comparison is true.

#### Format of Parameters

*VarA Operand VarB (IfTrueAction)*

Where:

|                       |  |
|-----------------------|--|
| <i>Var A</i>          | A variable, <code>Current_Element</code> , or a literal surrounded with double quotes. If <code>Current_Element</code> is used and is empty, processing stops on the rule. If the variable does not represent a numeric, an error message is issued. |
| <i>Operand</i>        | EQ, NE, GT, GE, LT, or LE.   |
| <i>VarB</i>           | A variable, <code>Current_Element</code> , or a literal surrounded with double quotes. If <code>Current_Element</code> is used and is empty, processing stops on the rule. If the variable does not represent a numeric, an error message is issued. |
| <i>(IfTrueAction)</i> | Business rule to be taken if the comparison is true.   |

**Extended Example.** Assume your company does not make adjustments in excess of 10000. You want to enforce this in a guideline based on 835-W120.

To accomplish this:

1. Assign an AddVar called PLBAdjustmentAmt to each Monetary Amount element in the PLB segment in Table 3.



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "AddVar" selected and a button with a function symbol (fx). Below this, there is a "Text" field containing the text "PLBAdjustmentAmt".

2. Create a CompareNumeric rule on the SE segment that would check the total in PLBAdjustmentAmt to see if it exceeds 10000. If so, display an error message.



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "CompareNumeric" selected and a button with a function symbol (fx). Below this, there is a "Text" field containing the text "PLBAdjustmentAmt GT "10000"(BusinessRules.Utilities.DisplayErrorByNumber 32214)".

Message 32214 is a custom message in file CustomerFSBRERRS.txt.

# Clear

---

## All validation programs

---

Clears the values from business rule variables. These are variables defined by a **BusinessRules.Variable** function like **SetVar** or **AddVar**.

Variables automatically clear out with each ISA, regardless of the presence of CLEAR rules, unless they are GLOBAL\_variables.

The location of the Clear is important. A typical place is on the first segment in a repeating loop or on the first required element of a repeating segment.

### Caution

It is hazardous to use Clear without specifying which variables are being cleared. A Clear without a variable name results in clearing all variables, including those in the HIPAA guideline with which you will eventually merge your rules.

A variable with a name that starts with **GLOBAL\_** will not be cleared unless it is specifically named in the Clear rule.

See page [214](#) for information on clearing local variables (those set by clicking the Variable button in the Business Rules dialog box).

### Format of Parameters

*"VarName" "VarName" "VarName" ...*

Where:

*"VarName"*                      Name of a variable, surrounded by double quotes. If omitted, all variables are cleared except those starting with GLOBAL\_.

*"VarName"*                      Optional names of additional variables, surrounded by double quotes and separated by spaces.

**Example.** This rule clears the BusinessRules.Variables HI0102IndustryCode and CLAIMcount.

The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "Clear" selected and a button with a function symbol (fx). Below this is a "Text" field containing the string: "HI0102IndustryCode" "CLAIMcount".

## ClearLocalVariable

---

### All validation programs

---

Removes one or more local variables from the repository. The variable had been defined by clicking the Variable button in the Business Rules dialog box as described on page [239](#).

Variables automatically clear out with each ISA, regardless of the presence of CLEAR rules, unless they are GLOBAL\_variables.

For information on clearing BusinessRules.Variable, see Clear on page [213](#).

### Format of Parameters

*"VarName" "VarName" "VarName" ...*

Where:

*"VarName"*                      Name of a local variable, surrounded by double quotes. You cannot clear all local variables by omitting a variable name in this rule. You must explicitly tell which ones are to be cleared.

*"VarName"*                      Optional. Names of additional local variables, surrounded by double quotes and separated by spaces.

**Example.** This rule clears two local variables.

What Rule to Run

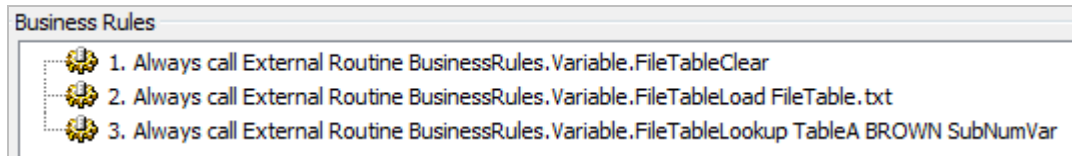
ClearLocalVariable ▼

Text "P2300D TP03AdmissDate" "P2300HI0201 Diag"

## FileTable Rules

The FileTable rules let you check an external text file for a value. If it is found, a related value is returned in a variable.

This set of rules clears the variables used by FileTableLookup rules, identifies the file containing the tables as FileTable.txt, and looks up BROWN in the file:



### The Table

The file must be in Instream's Bin directory. It contains the table name preceded with ^ and then one or more lines in this format:

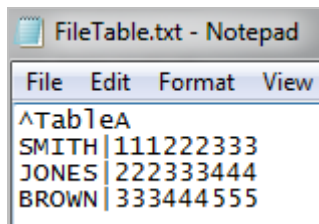
*key | value*

The business rule inquires if the key is in the file. If so, value is returned in a variable. If not, the variable is empty.

You can have one or more tables in this file.

### Example

This table lets you inquire if SMITH is in the table. If so, the value 111222333 is returned in a variable. Likewise, you could check for JONES or BROWN and get their related values.



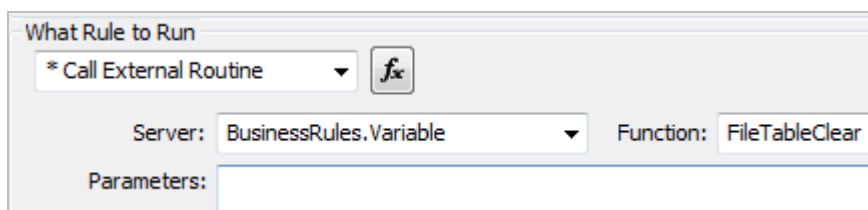
### FileTableClear

Clears the variable Retval, which is used in FileTableLookup.

### Format of Parameters

*None*

**Example.** This rule clears variable Retval.



## FileTableLoad

Identifies the file that contains the table that you will be using for a lookup. This file must be in Instream's Bin directory.

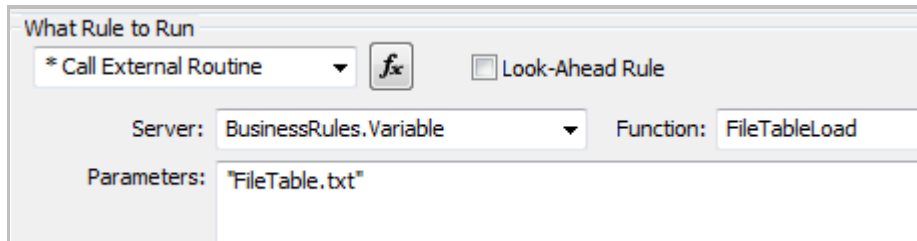
### Format of Parameters

*"filename"*


Where:

*"filename"* File name and extension.

**Example.** This rule identifies FileTable.txt in Instream's Bin directory.



What Rule to Run

\* Call External Routine  ☐ Look-Ahead Rule

Server: BusinessRules.Variable Function: FileTableLoad

Parameters: FileTable.txt

## FileTableLookup

Checks a table, in the file identified with FileTableLoad, for a value. If the value is in the table, a related value is returned.

### Format of Parameters

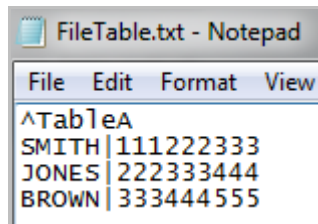
*"tableName" key ReturnVar*

Where:

*"tableName"* Table name within the file.

### Example

*tableName* is **TableA** in this file:



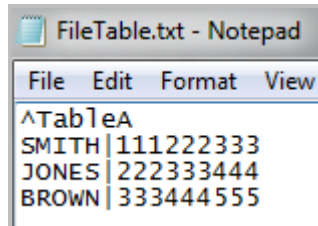
| FileTable.txt - Notepad |                  |
|-------------------------|------------------|
| File                    | Edit Format View |
| ^TableA                 |                  |
| SMITH                   | 111222333        |
| JONES                   | 222333444        |
| BROWN                   | 333444555        |

*key* Value to search for in first column.

*ReturnVar* Variable in which to return the corresponding value in the second column.

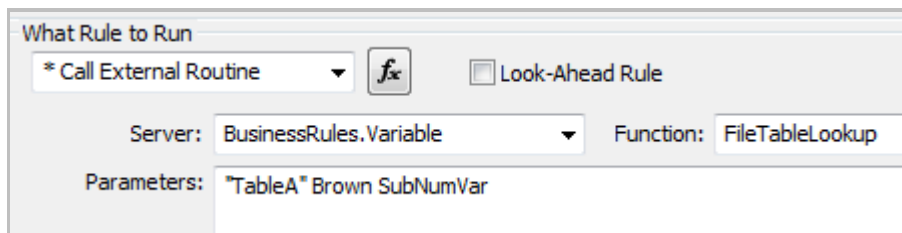
### Example

If *ReturnVar* is **SubNumVar** and *key* is **Brown**, then SubNumVar will contain **333444555** after the rule executes.




| ^TableA |           |
|---------|-----------|
| SMITH   | 111222333 |
| JONES   | 222333444 |
| BROWN   | 333444555 |

**Example.** If TableA contains BROWN, the corresponding value is returned in variable SubNumVar.



What Rule to Run

\* Call External Routine  ☐ Look-Ahead Rule

Server: BusinessRules.Variable Function: FileTableLookup

Parameters: "TableA" Brown SubNumVar

## GetInfo

Populates a variable with one of these:

- The iteration of the current loop. This is a digit.
- The loop ID, an underscore, and the iteration of the current loop. Example: 2000C\_3
- The value in an envelope element that is later in the segment than the business rule that uses it.



### Format of Parameters (3 variations):

**Current\_LoopCounter** *var*

**Current\_LoopKey** *var*

**ENV**(*elementIndex*) *var*

Where:

**Current\_LoopCounter**

Literal text that represents the iteration of the current loop. This reserved word should only be used with GetInfo.

**Current\_LoopKey**

Literal text that represents the loop ID, an underscore, and the iteration of the current loop. This reserved word should only be used with GetInfo.

**Env**

Literal text meaning the value is in the ISA or GS.

*elementIndex*

Element position within the current segment (the ISA or GS).

*var*

Variable to hold the loop count (for Current\_LoopCounter or Current\_LoopKey) or the value in the envelope element (for ENV).

### Example 1

This puts the loop count into a variable called CLMcount.



The screenshot shows a window titled "What Rule to Run". It has a dropdown menu with "GetInfo" selected and a button with a function symbol (fx). Below this is a "Text" field containing the text "Current\_LoopCounter CLMcount".

This puts the Loop ID and the iteration of the loop and into a variable called CLMkey.



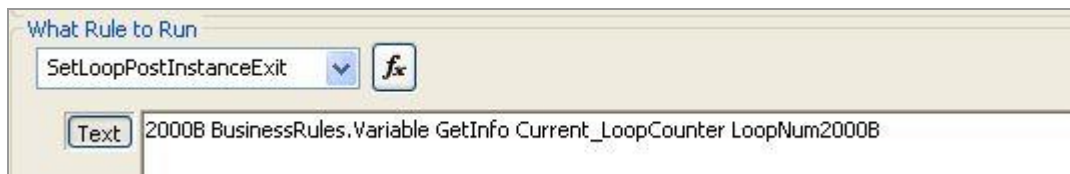
The screenshot shows a window titled "What Rule to Run". It has a dropdown menu with "GetInfo" selected and a button with a function symbol (fx). Below this is a "Text" field containing the text "Current\_LoopKey CLMkey".

By using a DumpVars and showing debug messages, we see a count like this in HIPAA Validator Desktop for each iteration of the loop.

```
CLMkey = "2300_2"  
CLMcount = "2"
```

### Example 2

This puts the loop count in variable LoopNum2000B and executes at the end of each iteration of the 2000B loop:



The screenshot shows a window titled "What Rule to Run". It has a dropdown menu with "SetLoopPostInstanceExit" selected and a button with a function symbol (fx). Below this is a "Text" field containing the text "2000B BusinessRules.Variable GetInfo Current\_LoopCounter LoopNum2000B".

By using a DumpVars and showing debug messages, we see the count in HIPAA Validator Desktop. This data had two 2000B loops and the first one had an error.

```
Start of Transaction Set, Ctl. No. 0111
LoopNum2000B = "1"
Ambiguous Segment placement - multiple Loops/Groups
Trailing Blanks in REF01 (D.E. 128) at col. 5
LoopNum2000B = "2"
End of Transaction Set, Ctl. No. 0111, contains
```

### Example 3

This rule, written on the ISA06, grabs the value in the ISA08 and places it in a variable called ISARceiverID.

What Rule to Run

GetInfo [v] [fx]

Text ENV(8) ISARceiverID

This rule, also on the ISA06, displays an error if the values in the ISA06 and ISA08 are the same.

## GetLength

### All validation programs

Puts the length of a value into a variable.

Format of Parameters

*TargetVar Source*

Where:

|                  |   |
|------------------|---|
| <i>TargetVar</i> | A variable to hold the length.  |
| <i>Source</i>    | A BusinessRules.Variable, Current_Element, or a literal surrounded with double quotes. The number of characters in this value will be counted and the result placed in TargetVar. |

**Example.** This rule counts the number of characters in SubscriberID and places the resulting number in SubscriberIDlength.

What Rule to Run

GetLength [v] [fx]

Text SubscriberIDlength SubscriberID

## GetValueFromSegment

---

### All validation programs

---

Gets a value from a variable that was saved with SaveCurrentSegment.

Format of Parameters:

*SegVariable VarType Element Subelement OutVar*

Where:

|                    |   |
|--------------------|---|
| <i>SegVariable</i> | The variable containing a segment; created with a SaveCurrentSegment rule.  |
| <i>VarType</i>     | Type of information, one of these literals:<br><br>VALUE     OutVar will contain a value from the segment.<br><br>NAME     OutVar will contain a name for the element-subelement ID. The actual name is your choice.<br><br>POS     OutVar will contain the segment's location in the file, where the first segment is 1, the second segment is 2, etc. |
| <i>Element</i>     | The position of the element that you are getting.<br>Examples: CLM*2*200.00***13:A:1**B*W*Y*****2~<br><br>2     Refers to the value 200.00<br><br>5     Refers to the value 13:A:1 (a composite)<br><br>-1     No specific element; refers to the whole segment.  |
| <i>Subelement</i>  | The location of the subelement within the element. Examples (using CLM segment above, and assuming Element was 5):<br><br>1     Refers to the value 13<br><br>3     Refers to the value 1<br><br>-1     No specific subelement. Refers to the whole composite. If the element is not a composite, always use -1.  |
| <i>OutVar</i>      | Variable that will contain the value, ID, or segment position requested.  |

**Examples.** Assume that a SaveCurrentSegment rule has saved the CLM segment in variable CLM2300SEG. Use CLM\*2235057\*460.00\*\*\*25:B:1\*N\*A\*N\*I\*P\*OA\*\*\*\*\*1~ as an example.

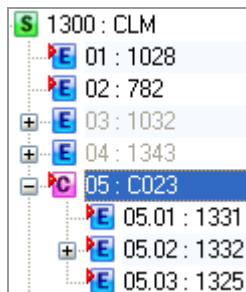
**Example 1.** This rule saves the CLM's position number to variable SEGPOS:

```
BusinessRules.Utilities.GetValueFromSegment:"CLM2300SEG" POS -1 -1 SEGPOS
```

**Example 2.** This rule saves the value in the CLM05 to variable CLM05. In this case, it is a composite. Because the subelement parameter is -1, the whole value 25:B:1 is placed in variable CLM05:

```
BusinessRules.Utilities.GetValueFromSegment:"CLM2300SEG" VALUE 5 -1 CLM05
```

**Example 3.** This rule gives the ID of the CLM05 the name CLM05NAME.



Notice that the subelement parameter is -1, so the name applies to the entire composite ID: C023

```
BusinessRules.Utilities.GetValueFromSegment:"CLM2300SEG" NAME 5 -1 CLM05NAME
```

**Example 4.** This rule saves the CLM0501 (the first subelement in the CLM05) to variable CLM0501. In our example, this will contain 25.

```
BusinessRules.Utilities.GetValueFromSegment:"CLM2300SEG" VALUE 5 1 CLM0501
```

**Example 5.** This rule shows how the variables above might be used by a BuildString rule. It strings together literals and variables and places the result in variable CTXOUTSTRING:

```
BusinessRules.Utilities.BuildString:CTXOUTSTRING "" "CTX|CLM" "" SEGPOS
"" CLM05 "" CLM05NAME ":" CLM0501
```

CTXOUTSTRING might contain something like this:

```
CTX|CLM*32**25:B:1*C023:25:C
```

## IsAlpha

---

### All validation programs

---

Checks a value and takes action if it consists entirely of letters of the alphabet (A-Z and a-z only).

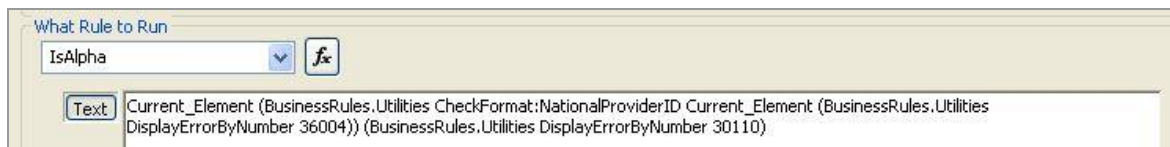
Format of Parameters

*Value (IfTrueAction) (IfFalseAction)*

Where:

|                      |   |
|----------------------|---|
| <i>Value</i>         | The value being checked. This can be a variable, <code>Current_Element</code> , or a literal in double quotes.  |
| <i>IfTrueAction</i>  | An action to be executed if the value contains all letters. Required if you are specifying an <code>IfFalseAction</code> . To do nothing if false, use this:<br>(BusinessRules.Utilities DoNothing) |
| <i>IfFalseAction</i> | Optional. An action to be executed if the value contains something other than letters.  |

**Example.** This rule checks to see if the current element is alphabetic. If so, it checks to see that it conforms to the `NationalProviderID` format and displays a message if it does not. If it is not alphanumeric, it displays error number 30110.



## IsAlphaNum

---

### All validation programs

---

Checks a value and takes action if it consists entirely of numbers and/or letters of the alphabet (0-9, A-Z, and a-z).

#### Format of Parameters

*Value CaseOption (IfTrueAction) (IfFalseAction)*

Where:

|                      |  |
|----------------------|--|
| <i>Value</i>         | The value being checked. This can be a constant in double quotes, a system variable (Current_Element, Current_Date, etc.), or an external variable name.   |
| <i>IfTrueAction</i>  | An action to be executed if the value passes the IsAlphaNum test. Required if you are specifying an IfFalseAction. To do nothing if false, use this:<br><br>(BusinessRules.Utilities DoNothing)                  |
| <i>CaseOption</i>    | Optional. Further limits the check to allow just upper-case or lower-case letters:<br>U = Limit valid characters to numbers and uppercase letters<br>L = Limit valid characters to numbers and lowercase letters |
| <i>IfFalseAction</i> | Optional. An action to be executed if the value fails the IsAlphaNum test.   |

## IsNum

---

### All validation programs

---

Checks a value and takes action if:

- It consists entirely of numbers
- It has a specific number of decimal places
- It has leading and/or trailing signs
- You want to specify certain decimal requirements.

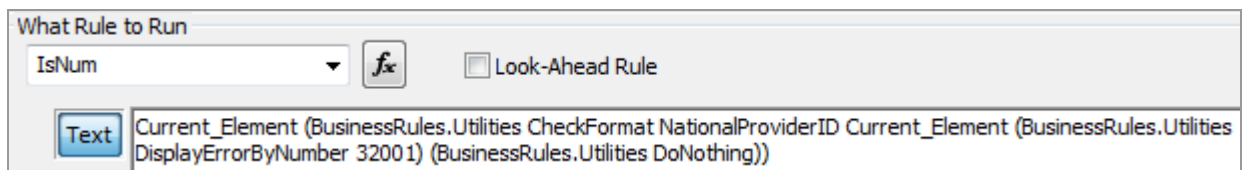
#### Format of Parameters

*Value (DecPlaceArg) (IfTrueAction) (IfFalseAction)*

Where:

|                      |   |
|----------------------|---|
| <i>Value</i>         | The value being checked. This can be a variable, <code>Current_Element</code> , or a literal in double quotes.  |
| <i>DecPlaceArg</i>   | <p>Optional. Check for number of decimal places, leading and/or trailing signs, and decimal place character requirements.</p> <p><i>DecPlaceArg</i> can be a constant in double quotes or a variable containing an option string. It is made up of one or more of the following sequences:</p> <p><b>Dn(-n)</b> – Check the <i>for</i> a an allowable number of decimal places.</p> <p><b>S</b> or <b>T</b> – Means the value must have a leading (S) or trailing (T) sign for the edit to pass. Otherwise, the <code>IfFalseAction</code> will be taken.</p> <p>If you include a comma or period, the decimal point character will have to be that character.</p> <p>See <i>DecPlaceArg</i> Examples below for additional information.</p> |
| <i>IfTrueAction</i>  | <p>An action to be executed if the value contains all numbers. Required if you are specifying an <code>IfFalseAction</code>. To do nothing if false, use this:</p> <p>(BusinessRules.Utilities DoNothing)</p>   |
| <i>IfFalseAction</i> | Optional. An action to be executed if the value contains something other than numbers.  |

**Example.** This rule checks to see if the current element is entirely numeric. If so, it checks to see that it conforms to the `NationalProviderID` and displays error 32001 if it does not.



The parameter is:

```
Current_Element (BusinessRules.Utilities CheckFormat
NationalProviderID Current_Element (BusinessRules.Utilities
DisplayErrorByNumber 32001) (BusinessRules.Utilities DoNothing))
```

## DecPlaceArg Examples

This section provides examples of the variations of the *DecPlaceArg* parameter.

- **Dn(-n)**, checks *Value* for a an allowable number of decimal places.  
**Dn**, where **n** is 0 – 9 requires that *Value* have exactly the specified number of decimal digits.  
For example, D3 will pass 34.123, but not 34.12 or 55.

**D n-n** specifies a range and requires that *Value* have at least the minimum number of decimal places, but no more than the maximum. For example, D2-4 will pass 123.4567 and 44.55, but not 123.4 nor 44.987654.

A number with a decimal point character but no subsequent decimal digits, such as 1234., will always fail.

### Examples

```
IsNum "1234.56" "D2" (IfTrueAction) (IfFalseAction)
```

Causes the *(TrueAction)* to be taken because 1234.56 is a number, and has two decimal places.

```
IsNum "1234.567" "D2" (IfTrueAction) (IfFalseAction)
```

Causes the *(IfFalseAction)* to be taken because, while 1234.567 is a number, it has three, not two, decimal places.

- **S** or **T** specifies requirement for sign characters '+' and '-'.

Use **S** to require leading signs in the value (ex. +123, -5.6).

Use **T** to require trailing signs in the value (ex. 123-, 2.345+).

Use both **S** and **T** to require signs before or after the number (ex. +123, 123-). (Note that in this case a value with both will fail.)

### Examples

```
IsNum "+1234.567" "D2-5S" (IfTrueAction) (IfFalseAction)
```

Causes the *(IfTrueAction)* to be taken because +1234.567 is a signed number.

```
IsNum "1234.567" "D2-5S" " (IfTrueAction) (IfFalseAction)
```

Causes the *(IfFalseAction)* to be taken because 1234.567 does not have the required leading sign character.

- **.** (period) or **,** (comma) forces the decimal point character to be the specified one. For example, D2-5, forces the decimal point character to be a comma and D2-5. forces the decimal point character to be a period.

The default decimal point character is determined from the input file if provided (for example, EDIFACT and its UNA segment). Otherwise, the default is a period.

### Examples

```
IsNum "1234,567" "D2-5" (IfTrueAction) (IfFalseAction)
```

The action taken depends on the decimal point character in effect.

- **For EDIFACT data**, where the decimal point is specified by the UNA as a comma, the *(IfTrueAction)* is taken.

- **For X12 data**, which assumes a period as the decimal point, the value is not recognized as a number, and the *(IfFalseAction)* is taken.

```
IsNum "1234,567" "D2-5, " (IfTrueAction) (IfFalseAction)
```

Causes the *(IfTrueAction)* to be taken because it forces the decimal point to be a comma.



## SaveCurrentSegment

---

### All validation programs

---

Saves the content of the current segment, minus the segment terminator, in a segment variable. The variable can then be used with the GetValueFromSegment rule.

**Note:** SaveCurrentSegment **only** functions with the GetValueFromSegment rule; it cannot be used with any other rule.

### Format of Parameters

*SegVariable*

Where:

|                    |   |
|--------------------|---|
| <i>SegVariable</i> | Storage name for the segment. This can be used with GetValueFromSegment rules but cannot be used as a typical variable. |
|--------------------|---|

### Example

---

This rule on the CLM segment saves the contents of the CLM segment to variable CLM2300SEG:



The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu with "SaveCurrentSegment" selected and a button with a function symbol (fx). Below this is a "Text" field containing the string "CLM2300SEG".

CLM2300SEG might contain something like this. The segment terminator is not included.

CLM\*2235057\*100.00\*\*\*13::1\*N\*A\*Y\*A\*B\*\*\*\*\*P

Please see GetValueFromSegment on page [220](#) for details about how to get specific values out of this variable.

# CheckCTT

## Analyzer Only

(See SetCheckCTT and SetCheckCTTCount on page 196 for Instream and HIPAA Validator Desktop validating)

Checks the value in the CTT-01 and, optionally, in the CTT-02 also.

For best results, place the rule on the ST segment. It has to appear before anything that it counts.

For Function Name, choose:

**SetCheckCTTCount** to check the CTT-01

**SetCheckCTT** to check the CTT-01 and CTT-02

The number of line items (CTT-01) is usually the count of the first loop in Table 2, or the table before the one containing the CTT. It is never an N1 loop.

The Hash total target is usually the first R-type field for amounts (such as element 330 or 782) on or after the line item segment. All hash total targets are type R, as is CTT-02 (element 347). The only hash total targets are elements 330, 782, 358, 380, 382, and 663.

| CTT Checking |                   |                         |                         |
|--------------|-------------------|-------------------------|-------------------------|
| Set          | CTT-01 NO. of ... | CTT-02 Hash Total of... | Element and Description |
| 202          | LX                |                         |                         |
| 205          | MMC               |                         |                         |
| 500          | HL                |                         |                         |
| 561          | HL                | PO102                   | 330 (Quan. Ordered)     |
| 568          | CS                | AMT02 at 2-090          | 782 (Mon. Amt)          |
| 810          | IT1               | IT102                   | 358 (Quan. Invoiced)    |
| 811          | IT1               |                         |                         |
| 819          | JIL               | JIL03                   | 782 (Mon. Amt)          |
| 828          | DAD               | Not Used                |                         |
| 830          | LIN               | FST01                   | 380 (Quantity)          |
| 832          | LIN               | Not Used                |                         |
| 840          | PO1               | PO102                   | 330 (Quan. Ordered)     |
| 843          | PO1               | PO102                   | 330 (Quan. Ordered)     |
| 844          | CON               | QTY02                   | 380 (Quantity)          |
| 845          | CON               | QTY02                   | 380 (Quantity)          |
| 846          | LIN               | QTY02                   | 380 (Quantity)          |
| 847          | HL                | Not Used                |                         |

| CTT Checking |                   |                         |                         |
|--------------|-------------------|-------------------------|-------------------------|
| Set          | CTT-01 NO. of ... | CTT-02 Hash Total of... | Element and Description |
| 849          | CON               | QTY02                   | 380 (Quantity)          |
| 850          | PO1               | PO102                   | 330 (Quan. Ordered)     |
| 851          | LS1               | LS101                   | 380 (Quantity)          |
| 852          | LIN               | Not Used                |                         |
| 853          | TD5               | Not Used                |                         |
| 855          | PO1               | PO102                   | 330 (Quan. Ordered)     |
| 856          | HL                | SN102                   | 382 (Units Shipped)     |
| 860          | POC               | POC03                   | 330 (Quan. Ordered)     |
| 861          | RCD               | RCD02                   | 663 (Quan. Units)       |
| 862          | LIN               | FST01                   | 380 (Quantity)          |
| 865          | POC               | POC03                   | 330 (Quan. Ordered)     |
| 866          | DTM               | QTY02                   | 380 (Quantity)          |
| 867          | LIN               | QTY02                   | 380 (Quantity)          |
| 869          | HL                |                         |                         |
| 870          | HL                | PO102                   | 330 (Quan. Ordered)     |

**Example.** This rule checks the CTT-01.

What Rule to Run

SetCheckCTT

fx

Text

# FSVBEExit.CheckDigit

---

## Analyzer Only

---

For other validators, see CheckFormat on page [160](#).

Checks if data conforms to a length requirement, contains the correct number of leading zeros and has a correct check digit as its final digit.

---

## Check Digit algorithm

All TIBCO Foresight CheckDigit rules use UPC-A system:

1. Add together all the digits in odd-numbered positions and multiply that sum by 3.
  2. Then add each digit in an even-numbered position to that sum.
  3. The check digit will be whatever number you need to add to that end result sum to make it a multiple of 10.
- 

The rule looks like this:

The screenshot shows a dialog box titled "What Rule to Run". It contains a dropdown menu with the selected option "\* Call External Routine" and a function icon. Below this, there are two fields: "Server:" with the value "FSVBEExit.CheckDigit" and "Function:" with the value "ValidateCheckDigit". At the bottom, there is a "Parameters:" label followed by an empty text box.

## X12 234-235 CheckDigit

---

### Analyzer Only

---

CheckDigit will check element 234's last digit against the qualifier in element 235 if element 235 contains a code that has one of these character strings in its *definition*:

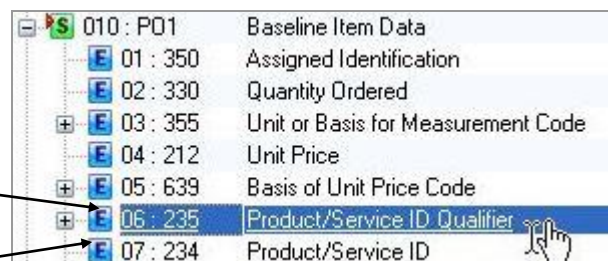
**EAN**

**U.P.C.**

The CheckDigit rule goes on element 234.

If element 235's code has  
EAN or U.P.C. in its  
definition ...

Then element 234's  
data will be checked by  
the corresponding check  
digit formula. Put the  
CheckDigit rule here.



|           |                                    |
|-----------|------------------------------------|
| 010 : P01 | Baseline Item Data                 |
| 01 : 350  | Assigned Identification            |
| 02 : 330  | Quantity Ordered                   |
| 03 : 355  | Unit or Basis for Measurement Code |
| 04 : 212  | Unit Price                         |
| 05 : 639  | Basis of Unit Price Code           |
| 06 : 235  | Product/Service ID Qualifier       |
| 07 : 234  | Product/Service ID                 |

| ID | Value Definition                     |
|----|--------------------------------------|
| EM | Equipment Identification Number      |
| EN | EAN/UCC - 13                         |
| EO | EAN/UCC - 8                          |
| OC | Old U.P.C./EAN Consumer Package Code |
| OE | Original Equipment Number            |

Analyzer will check the data in element 234 to see if it complies with the specified format of the code value that was used.

## EDIFACT 3039-3055 CheckDigit

---

### Analyzer Only

---

CheckDigit will check the last digit in the element 3039 if the data in element 3055 contains a code that has one of these character strings in its *definition*:

**EAN**

**UPC**

The CheckDigit rule goes on element 3039.

*CheckDigit rule goes here.  
Element 3039's data will be  
checked by the check digit  
formula ...*

*...if element 3055's code  
has EAN or UPC in its  
definition ...*

The screenshot shows an EDIFACT structure editor. The top part is a tree view of the EDIFACT structure. The bottom part is a table showing the definition of element 3055.

| ID  | Value Definition               |
|-----|--------------------------------|
| 113 | US, UPC (Uniform product code) |
| ZZZ | Mutually defined               |

Analyzer will check the data in element 3039 to see if it complies with the specified format of the code value that was used.

## Other CheckDigit Options

---

### Analyzer Only

---

See also CheckFormat on page [160](#).

Besides the 234 - 235 and 3039 - 3055 pairs, CheckDigit server can check other numeric values for a correct check digit, min/max length, and the correct number of leading zeros.

You can enter these as parameters when setting up the CheckDigit routine:

|              |  |
|--------------|--|
| <b>EAN8</b>  | Data must be exactly 8 characters with no leading zeros.               |
| <b>EAN13</b> | Data must be exactly 13 characters long with no leading zeros.         |
| <b>EAN14</b> | Data must be exactly 14 characters long with no leading zeros.         |
| <b>SSCC</b>  | Data must be exactly 18 characters long with up to one leading zero.   |
| <b>UPC12</b> | Data must be exactly 12 characters long and can include leading zeros. |

In each of these, the length includes the check digit.

---

### Example: X12 Element 87 (MAN segment in 850)

---

*If element 88 contains UP,  
then the data for the current element must conform to UPC12*



When Analyzer encounters this element 87, it will look at element 88 to see if it contains qualifier UP. If so, it will verify that element 87 conforms to UPC12 (exactly 12 digits long with up to two leading zeros) and has the correct check digit.

## User Defined Check Digit

---

### Analyzer Only

---

You can also check other numeric values, even if they have no qualifier. The data must match the parameter format specified by the guideline or MIG developer in Standards Editor.

The generic format of **min-max Zn** can be used in place of the EDISIM-defined parameters, where:

|            |   |
|------------|---|
| <b>min</b> | is the minimum length of the field  |
| <b>max</b> | is the maximum length of the field (optional)   |
| <b>Zn</b>  | <b>Zn</b> is optional. <b>Z</b> is a literal and <b>n</b> is the number of leading zeros allowed. If omitted, no leading zeros are allowed. |

Examples:

- 5Z0** or **5** Data must be at least 5 characters long with no leading zeros.
- 5-5Z0** Data must be at exactly 5 characters long with no leading zeros.
- 10-12Z2** Data must be between 10 and 12 characters long (inclusive) with up to two leading zeros.
- 5-6** or **5-6Z0** Data must be between 5 and 6 characters long with no leading zeros.

### EDIFACT element 7402 example

---

This is placed on element 7402 in a GIR segment in the ORDERS message:



What Rule to Run

\* Call External Routine

Server: FSVBExit.CheckDigit

Function: ValidateCheckDigit

Parameters: 1020

When Analyzer encounters element **7402**, it will verify that this element is exactly 10 digits long with no leading zeros and has the correct check digit.

## DateTime

---

### Analyzer Only

---

For other validators, see [Date and Time](#) on page 74.

Checks date and time to see if it follows the specified format.

**X12 element 1251** Place the rule on the element **1251** that you want to check.  
Use function **ValidateDateTimeX12**. The rule will check to see if it follows the format specified in element 1250.

**EDIFACT element 2380** Place the rule on the element **2380** that you want to check..  
Use function **ValidateDateTimeUN**. The rule will check to see if it follows the format specified in element. 2379.

Be sure to customize the code values for the corresponding qualifier: X12 element 1250 or EDIFACT element 2379.

**Example.** This rule checks X12 element 1251.



What Rule to Run

ValidateDateTimeX12

Text



# FSVBEit.DisplayMessage

---

## Analyzer Only

---

For other validators, see [DisplayErrorByNumber](#) on page 166.

Displays a customized diagnostic.

- Display the diagnostic *Code value must be 00 or 01* if a value is something else. See example 1 below.
- Display the diagnostic *Vendor Num REF must precede Booking Num REF* if this condition is violated. See example 2 below.
- Displays the message DUNS number must be 7825012250001 or 7825012250022. See example 3 below.

Steps include:

1. Define variables needed by the condition, if any.
2. In the Condition and Result Definitions box, set up the condition in the top and select Invoke External Routine.
3. For Server Name, select **FSVBEit.DisplayMessage**.
4. For Function Name, select **DisplayError**.
5. For Parameters, type the diagnostic that is to display if Analyzer finds that the condition is violated.

## Example

---

This example displays “New PO1 loop” each time this segment appears in the data.

What Rule to Run

\* Call External Routine

Server: FSVBEit.DisplayMessage Function: DisplayError

Parameters: New PO1 loop

## Example: More descriptive diagnostics about code value violation

---

Our purchase order allows 2 code values for the BEG01: 00 and 01. Data that contains any other value causes Analyzer to issue a diagnostic similar to “Code Value "03" not used for BEG01 (D.E. 353) at col. 5.”

This business rule will give an additional diagnostic: “Code value must be 00 or 01.”

The screenshot shows a business rule configuration window with the following sections:

- When to Run Rule:** The **Conditionally** radio button is selected.
- Logic Table:** A table with columns **Local Variable**, **Operator**, and **Constant**. It contains two rows connected by an **Or** operator.

|      | Local Variable   | Operator | Constant |
|------|------------------|----------|----------|
| when | BEG01PurposeCode | NE       | 00       |
| when | BEG01PurposeCode | NE       | 01       |
- What Rule to Run:** The **\* Call External Routine** option is selected, accompanied by a function icon.
- Server:** FSVBExit.DisplayMessage
- Function:** DisplayError
- Parameters:** Code value must be 00 or 01

## Example: Enforcing a particular order for REF segments

Let's assume that we want two or more consecutive REF segments, and they have to be in the same order as they appear in the guideline. Ordinarily, Analyzer would allow consecutive REF segments to appear in any order in the data.

Edit the code values in each REF-01 to be unique.

Assign a local variable to the first REF. In our example we'll use REF01first.

Create this business rule on the second REF:

The screenshot shows the 'When to Run Rule' and 'What Rule to Run' sections of a business rule editor.

**When to Run Rule:**

- ☐ Always
- ☒ Conditionally

**When:**

|        | Local Variable  | Operator      |
|--------|-----------------|---------------|
| Single | when REF01first | DOESN'T EXIST |
|        | when            | EQ            |

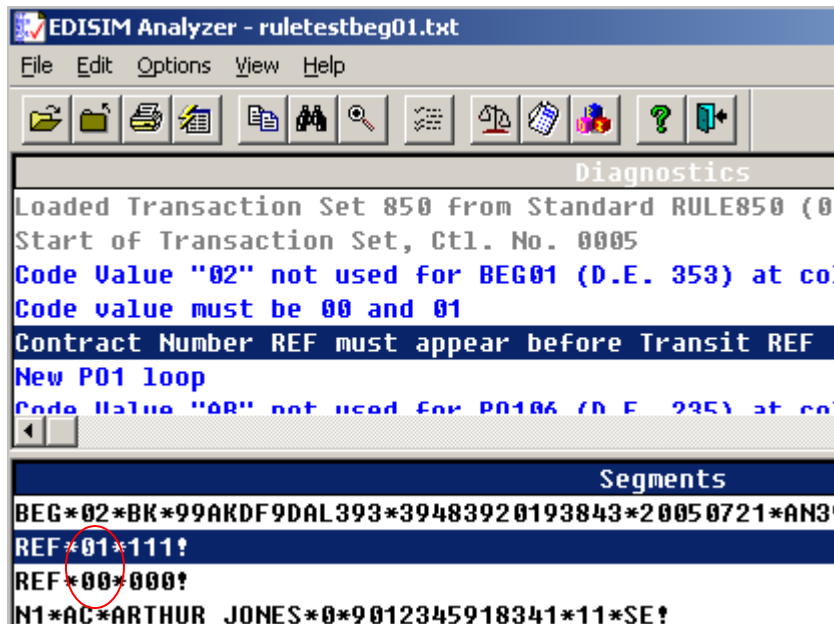
**What Rule to Run:**

\* Call External Routine

Server: FSVBExit.DisplayMessage Function: DisplayError

Parameters: Contract Number REF must appear before Transit REF

If the REFs appear out of order, Analyzer displays the custom message:



## Example: Display the application value in an Analyzer diagnostic

---

Analyzer flags application value violations with a message like this:

Application Value "9012345918341" not found in value list "DUNS"

By using DisplayMessage, you can display an additional message that lists what values *are* acceptable:

DUNS number must be 7825012250001 or 7825012250022

To set this up:

1. Attach the DUNS application value list to the element.
2. Place a variable on the element. In our example we'll use 3100N104.
3. Place this rule on the element:

**Condition and Rule Definition**

When to Run Rule

☐ Always

☒ Conditionally

|    | Local Variable | Operator | Constant      |
|----|----------------|----------|---------------|
| Or | when 3100N104  | NE       | 7825012250001 |
|    | when 3100N104  | NE       | 7825012250022 |

What Rule to Run

\* Call External Routine

Server: FSVBExit.DisplayMessage Function: DisplayError

Parameters: DUNS number must be 7825012250001 or 7825012250022

If Analyzer finds another value at that location, it will display the message that you specified.

## ProductUtilities

---

### Analyzer Only


---


This rule checks segments with repeating pairs of element 235-234 (like the 850's LIN segment in recent vintage X12 versions) by:

- Defining code(s), if any, that must be used in an element 235 in the segment.
- Allowing the 235-234 pairs to appear in any order within the segment.
- Prohibiting duplicate codes in element 235 within the segment.

### Example

This rule on the LIN segment requires at least three 235 elements. They must contain B3, B5, and B6.



|                          |   |
|--------------------------|---|
| What Rule to Run         |   |
| * Call External Routine  |  |
| Server: ProductUtilities | Function: Check235  |
| Parameters: B3,B5,B6     |   |

When analyzing EDI data against this guideline, Analyzer will display diagnostic messages if these conditions are not met: “Check235 Error(s) - Duplicate Codes: B5 Missing Codes: B6” or similar.

# Appendix A: Variables

---

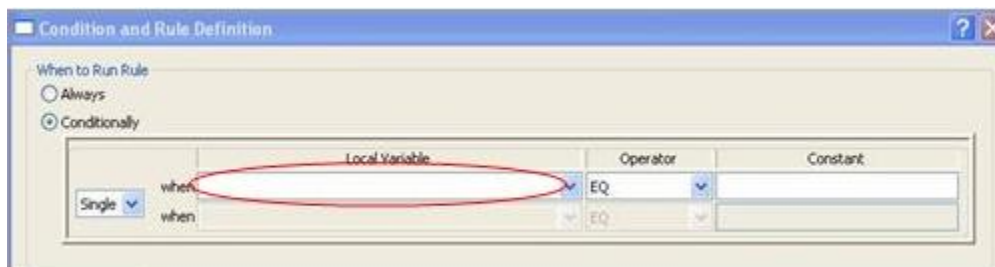
Before using any element other than the current element in a business rule, you will need to assign it a "variable" name. There are two types of variables: **local variables** and **BusinessRules.Variable**.

When do you use a local variable and when do you use a BusinessRules.Variable? That depends on how you want to use it in a rule. See the next two sections for details.

## Local Variables

### When to use Local Variables

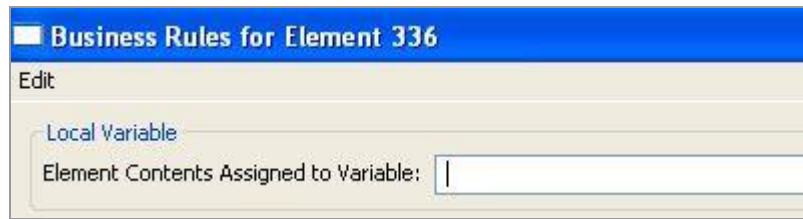
Assign a local variable if you will use it in the *top* of the Business Rules box. You must be on an element.



### Assigning a Local Variable

To assign a local variable:

1. Click on the element.
2. Select **Edit | Advanced | Business Rules**.
3. In the Local Variable area, type a variable name that conforms to the suggestions in Good Variable Names on page [241](#).



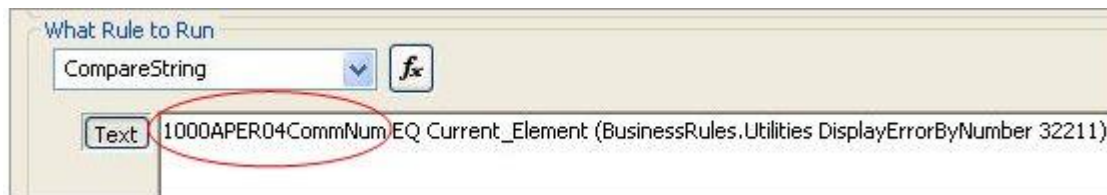
4. Click **OK** to close the dialog boxes.

## BusinessRules.Variable

### When to use BusinessRules.Variable

Use `BusinessRules.Variable` when you want to assign a variable name that will be used in the Parameter area of the Condition and Rule Definition dialog.

In this example, variable `1000APER04CommNum` must be a `BusinessRules.Variable` because it is used in the Parameters area of the rule:



This example compares the contents of variable `1000APER04CommNum` to the current element's value. If they are the same (true condition), then error number 32211 displays.

### Setting up BusinessRules.Variable

Create `BusinessRules.Variable` with business rule functions like `SetVar` or `AddVar`.

To assign a `BusinessRules.Variable`:

1. Click on the element or segment.
2. Choose **Edit | Advanced | Business Rules | New**.
3. Click **Always** and select **\*Call External Routine** from the drop box.
4. For Server Name, choose **BusinessRules.Variable** and then choose the Function Name **SetVar** (see page 202) or **AddVar** (see page 204).

You can also assign variables with these functions under **BusinessRules.Utilities**:

**AppendString** (page 153), **GetToken** (page 173), or **SubString** (page 173).

5. Type the variable's name in the rule definition area, as in the following example. Follow the naming suggestions in Good Variable Names on page 241.



## Good Variable Names

TIBCO Foresight-supplied variable names indicate the location where the variable is assigned. For example, to find 2010ABNM108IDQual:

2010AB = loop 2110AB  
NM1 = segment  
08 = element  
IDQual = short description

It is good practice for you to follow this convention and use the location of the element as part of the name.

Variable names can be any length and should not contain special characters or spaces.

**Example.** If you are assigning a variable to the Statement Date DTP02 in loop 2300 of the Subscriber HL loop of an 837:

|                  | Variable Name   | Explanation  |
|------------------|-----------------|--|
| <b>Good name</b> | S2300DTP02StmDt | S for Subscriber HL level.<br>2300 for loop 2300.<br>DTP02 for segment and element.<br>StmDt for the Statement Date DTP. |
| <b>Poor name</b> | DTP02           | Which DTP? There are many.   |

**Example.** If you are assigning a variable to the NM108 Identification Code Qualifier in loop 2010AB:

|                  | Variable Name     | Explanation   |
|------------------|-------------------|---|
| <b>Good name</b> | 2010ABNM108IDQual | 2010AB for loop 2010AB.<br>NM108 for segment and element.<br>IDQual for the element name. |
| <b>Poor name</b> | IDQual            | Which ID Qualifier? There are many.   |

If you follow these guidelines, a variable's name will tell you where it is assigned.



## Global Variables

To set up a variable that is cleared only by specifically naming it in a **BusinessRules.Variable Clear** rule, assign a name that starts with **GLOBAL\_** (note the underscore).

**Example.** GLOBAL\_2010ABNM108IDQual. The word GLOBAL and the underscore are actually part of the name and should be included when using the name in a rule.

## TIBCO Foresight-Defined Variables

Instream sets up these variable names when the guideline runs a UserExitWithWait business rule (see page 108):

|                           |  |
|---------------------------|--|
| <b>FS_UserExit_Status</b> | Return status from external program, which can be:                           |
| 200                       | Initial state at startup before attempt to call process                      |
| 201                       | Called the process   |
| 202                       | Process completed within the time specified (Only used for UserExitWithWait) |
| 203                       | Process cancelled due to time limit (Only used for UserExitWithWait)         |
| 204                       | Failed to locate Process   |

|                            |  |
|----------------------------|--|
| <b>FS_UserExit_RtnCode</b> | The return code from the external program. |
|----------------------------|--|

The following variable is contained in the the base validation guidelines provided with TIBCO Foresight products, however for it to be made active, a SetVar business rule is required in the companion guideline.

### **FS\_ICD9\_ICD10\_CutoverDate**

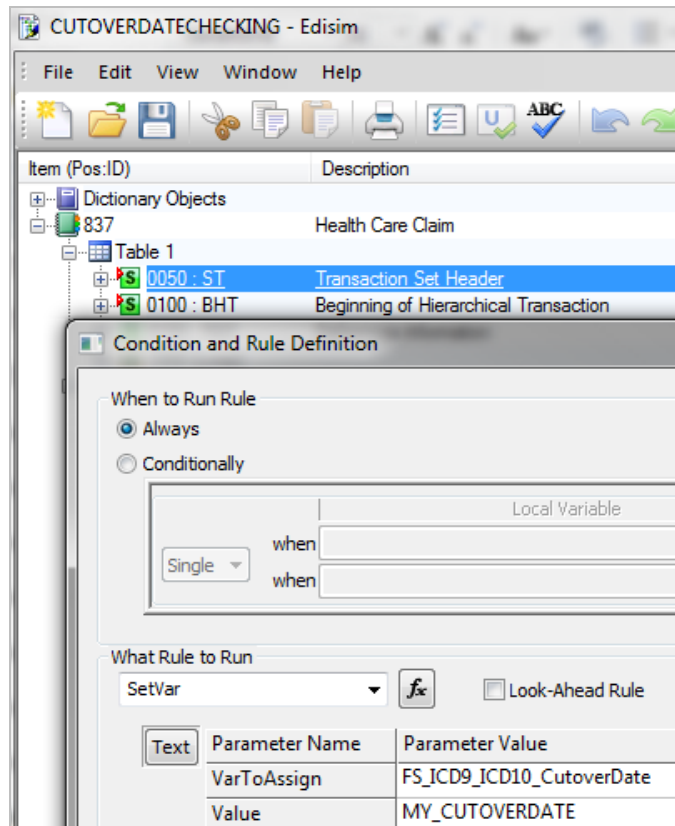
This variable supports the ICD-9 to ICD-10 conversion process cutover and is pre-set to the appropriate date.

To update this value without changing the associated guidelines, override the variable by directing Instream to call an external PreLoadedVariable file. See Procedure: Changing the FS\_ICD9\_ICD10\_CutoverDate Variable on page 243.

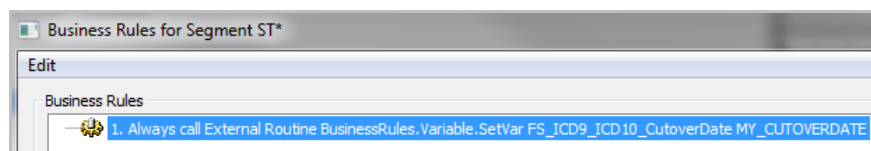
### Procedure: Changing the FS\_ICD9\_ICD10\_CutoverDate Variable

Use the following steps to update the FS\_ICD9\_ICD10\_CutoverDate variable.

1. Open your companion guideline using EDISIM Standards Editor. If you don't have a companion guideline, create a new one. In the examples in this procedure, our companion guideline is called CUTOVERDATECHECKING.
2. On the ST segment of the companion guideline, add a SetVar business rule. We want to assign the TIBCO Foresight-define variable FS\_ICD9\_ICD10\_CutoverDate to a new variable name. Use the variable name of your choice. In this example our new variable name is MY\_CUTOVERDATE.

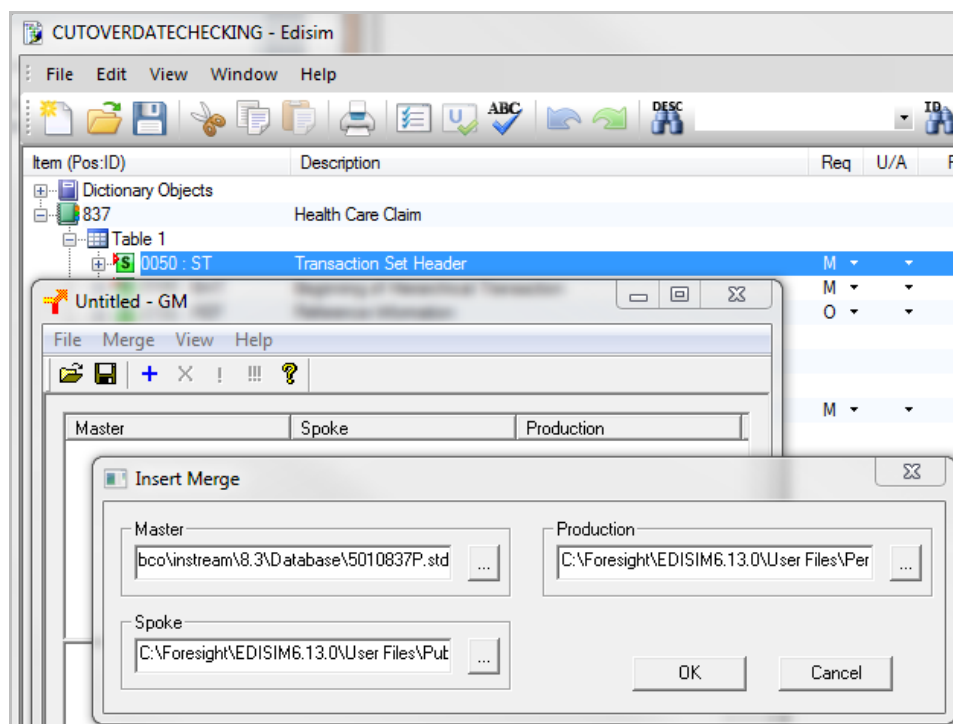


When you are done, your business rule should look something like this:



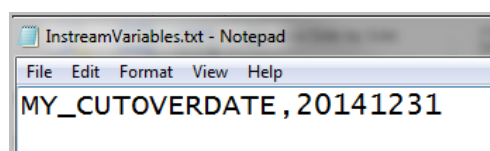
Save your guideline.

3. Use GuideMerge to merge your companion guideline with the base guideline (which includes types 1-7 edits). Save your new guideline. Refer to **GuideMerge.pdf** for more information on merging.

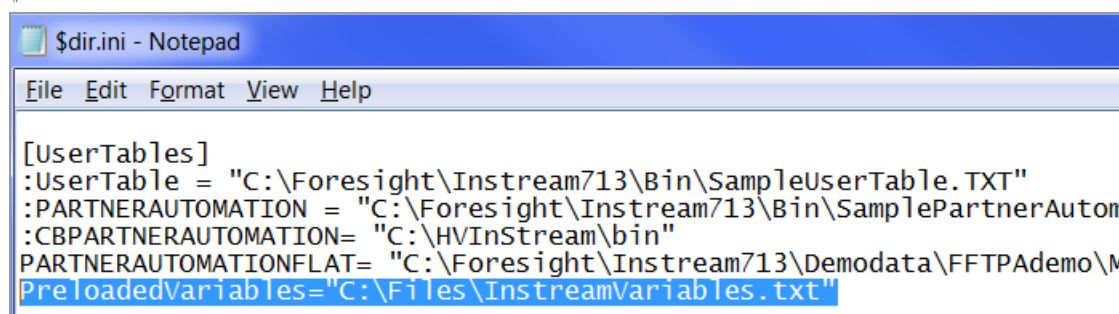


4. Edit your external variables file to update the value for your new variable. If you don't have an external variables file, create one. (See Populating Variables with an External Variables File on page 248.)

We want to use MY\_CUTOVERDATE instead of the TIBCO Foresight-defined variable FS\_ICD9\_ICD10\_CutoverDate and we want MY\_CUTOVERDATE to be set to 20141231. Save the file.



5. Tell Instream where to find your external variables file by adding a **PreloadedVariables** line to your \$dir.ini file. Save the file.



The Preloaded Variables file will now be called during validation.

# Preprocessor Variables

You can define and reference variable names on the fly by bracketing them by percent signs.

These “preprocessor variables” differ from regular variables in these ways:

- For most rules, they are evaluated when the rule is executed.
- For exits, they are evaluated when the rule in the exit is triggered, not when the exit itself is encountered.
- Their value can be another variable.
- They can be used in any part of a business rule parameter.
- They can be used to display a value in error messages for hard-coded rules but not in CustomerFSBRerrs.txt rules.
- They are surrounded by percent signs. Any time a variable bracketed by percent signs is seen (ex. %TESTVAR%), it will be replaced by that variable’s contents.

For example, if we have a variable named “LocID” that contains a string corresponding to the Location ID (say 540) then the statement

```
InsertList %LocID% ShipLOC
```

becomes

```
InsertList 540 ShipLOC
```

**Example.**

A screenshot of a software interface titled "What Rule to Run". It contains a dropdown menu with "AddVar" selected, a button with a function symbol (f), and a text input field with the text "%PO1Count% "1"". There is also a "Text" button next to the input field.

The % delimiters around PO1Count show that this is not the actual name, but a variable that will point to the name. The contents might be another variable, which might contain another variable and so on. At runtime, this chain is resolved as far as it goes.

These “indirect references” are resolved when the rule is run, not when it is encountered.

However, the following exits resolve the variable *when the rule that they contain runs*, not when the exit itself is encountered:

- SetCompositePreExit
- SetElementPostExit
- SetLoopPostExit
- SetLoopPostInstanceExit
- SetSegmentPreExit

| Variable Delimiters                  |  |                                       |
|--------------------------------------|--|---------------------------------------|
|                                      | Variable contains constant                   | Variable contains another variable    |
| Used in CustomerFSBRerrs.txt message | #<br>See example 1                           | cannot be used in external error file |
| Used in hard-coded message           | %<br>See example 2                           | %<br>See example 3                    |
| Used in body of business rule        | unless noted, no delimiters<br>See example 4 | %<br>See examples 5 and 6             |

A number of examples are below. For two complex preprocessor variable examples involving maps, see page [251](#).

**Example 1.** This example used a regular variable, not a preprocessor variable. OrderNum is a variable containing a constant value.

What Rule to Run

DisplayErrorByNumber

Text 32005

32005 Order #OrderNum# received on #Current\_Date#

**Example 2.** OrderNum is a variable containing a constant value. The error message is hard-coded into the business rule and contains two variables, which have to be surrounded by percent signs.

What Rule to Run

DisplayErrorByNumber

Text 0 0 "Order %OrderNum% received on %Current\_Date%"


**Example 3.** While this example appears this same as Example 2, in this case the variable OrderNum contains a variable LastOrderNum which can contain a constant or another variable.

What Rule to Run

DisplayErrorByNumber

Text 0 0 "Order %OrderNum% received on %Current\_Date%"

**Example 4.** OrderNum and MinNum are regular variables containing constants. They are not surrounded by any special characters in the body of the rule except in DisplayErrorByNumber.

|                  |   |
|------------------|---|
| What Rule to Run |   |
| CompareNumeric   |                                  |
| Text             | OrderNum LT MinNum (BusinessRules.Utilities.DisplayErrorByNumber 0 0 "Order number cannot be less than %MinNum%") |

**Example 5.** This example creates a list name on the fly. The rule is inserting the contents of LineItemAMT into the list (name to be determined by resolving the variable OrderID):


|                  |   |
|------------------|---|
| What Rule to Run |   |
| ListInsert       |  |
| Text             | %OrderID% LineItemAMT   |

Assume:

|             |          |             |
|-------------|----------|-------------|
| OrderID     | contains | 20081025A10 |
| LineItemAMT | contains | 72.50       |

The parameters above becomes:  
 20081025A10 72.50 *(insert 72.50 into the list named 20081025A10)*

**Example 6.** Nested variables.

|                  |   |
|------------------|---|
| What Rule to Run |   |
| AddVar           |  |
| Text             | %ABC1% "1"  |

Assume:

|     |          |             |
|-----|----------|-------------|
| ABC | contains | %DEF%       |
| DEF | contains | %GHI%       |
| GHI | contains | JOHNSONJAKE |

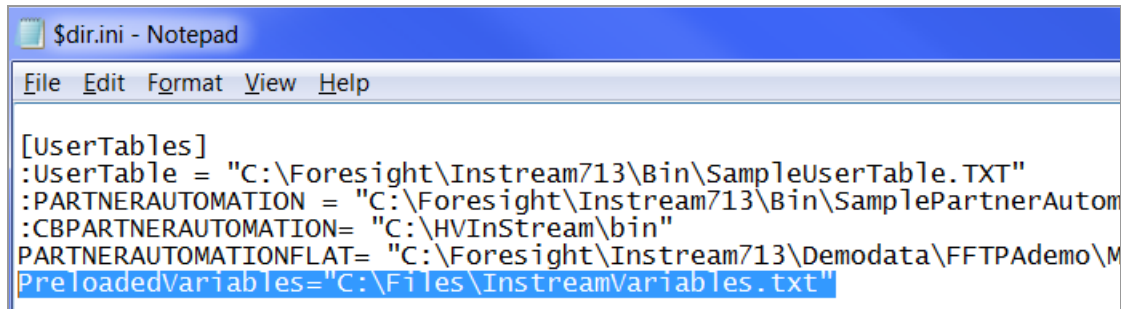
The parameters above becomes:  
 JOHNSONJAKE "1" *(Add 1 to variable JOHNSONJAKE)*

# 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 guideline.

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

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



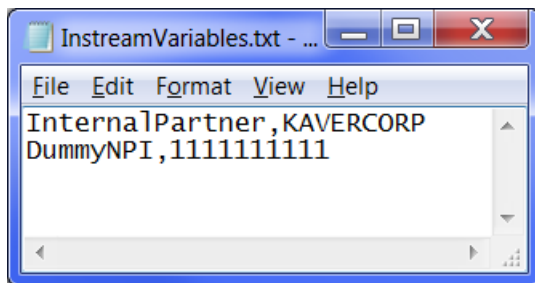
```
$dir.ini - Notepad
File Edit Format View Help

[UserTables]
:UserTable = "C:\Foresight\Instream713\Bin\SampleUserTable.TXT"
:PARTNERAUTOMATION = "C:\Foresight\Instream713\Bin\SamplePartnerAutom
:CBPARTNERAUTOMATION= "C:\HVInStream\bin"
PARTNERAUTOMATIONFLAT= "C:\Foresight\Instream713\Demodata\FFTPAdemo\M
PreloadedVariables="C:\Files\InstreamVariables.txt"
```

Inside the variable file, each line contains:

*variable,value*

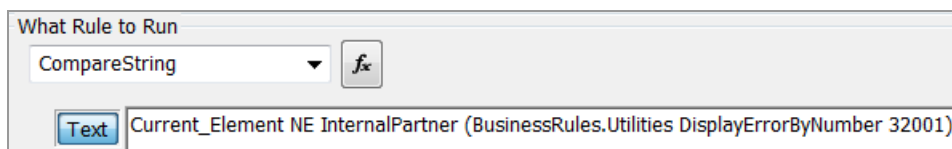
**Example:**



```
InstreamVariables.txt - ...
File Edit Format View Help
InternalPartner,KAVERCORP
DummyNPI,1111111111
```

This file contains two 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 Instream. This file can contain other data also.

# Initializing and Clearing Variables and Lists

Local variables, SetVar variables, and lists retain their current values until:

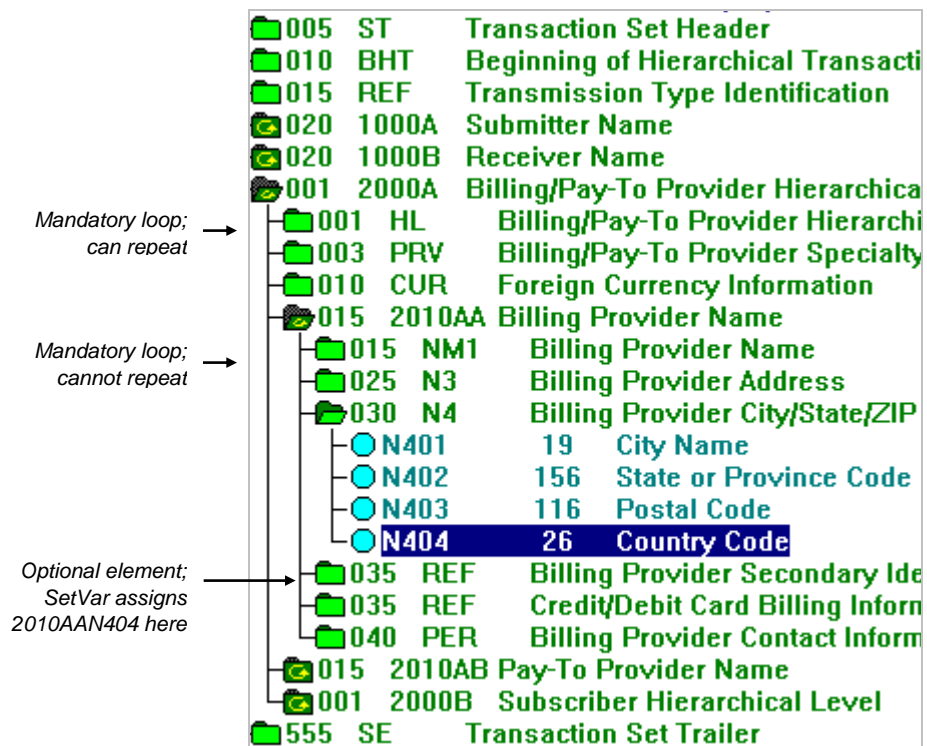
- The value is changed with another rule.
- The value is changed in another iteration of the loop that contains the rule.
- The value is explicitly cleared.
- The transaction set ends.

## Variables in Loops

If a value is set on an element that will always occur, then the value will be overwritten with each iteration of the loop.

If a value is set on an element that may or may not occur, then the value may persist through multiple iterations of the loop. You may need to reset the value at the beginning or end of the loop.

Consider this SetVar variable:



Since N4-04 is optional, we want to provide for loop iterations where the N404 is not present. There are many ways to handle this.



### Example Method 1. Initialize

On the 2000A HL or the 2010AA NM1 segment, initialize the value:

What Rule to Run

SetVar

Text 2010AAN404 "US"

### Example Method 2. Clear at top of loop

On the 2000A HL or the 2010AA NM1 segment, clear the value:

What Rule to Run

Clear

Text 2010AAN404

### Example Method 3. Clear at end of loop

On the ST, clear the variable at the exit of each 2000A or 2101AA loop:

What Rule to Run

SetLoopPostInstanceExit

Text 2000A BusinessRules.Variable Clear 2010AAN404

## Lists in Loops

---

The same principals apply to lists and to variables. You can reset the list with a `BusinessRules.List ClearList` (see page 111) at the beginning of the loop or as a loop exit.

# Variable Maps

Variables can be maps with alphanumeric indices. A map reference is a variable followed by an alphanumeric index in parentheses, like these examples:

|   |   |
|---|---|
| <code>VendorTypeTotals("InState")</code>  | Refers to the "InState" slot of variable <code>VendorTypeTotals</code> .  |
| <code>StateTotals("OH")</code>            | Refers to the "OH" slot of variable <code>StateTotals</code> .  |
| <code>CategoryFlag(1)</code>              | Identical to <code>CategoryFlag("1")</code> .   |
| <code>IDNeeded("")</code>                 | Identical to <code>IDNeeded</code> (just a regular variable).   |
| <code>IDNeeded(MyVar)</code>              | Gets the index from the contents of variable <code>MyVar</code> .   |
| <code>CountryTotals(%MyVar%)</code>       | Gets the index from the contents of the variable <u>contained</u> in <code>MyVar</code> . See Preprocessor Variables on page <a href="#">245</a> .    |
| <code>BuyerIndex(MyVar(BuyerCode))</code> | First uses <code>BuyerCode</code> 's contents as index into <code>MyVar</code> array, which it then uses as index into <code>BuyerIndex</code> array. |

`BusinessRules.Variable.Clear` will clear the entire map, and `BusinessRules.Variable.DumpVars` will display all map members, including their keys.

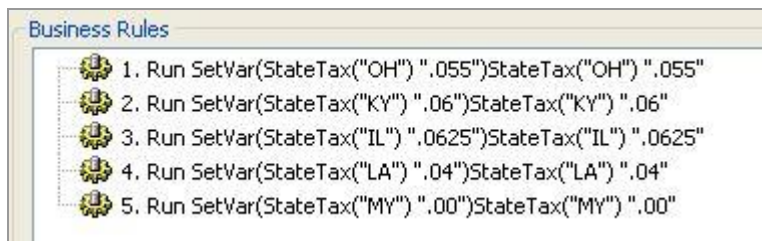
Maps are especially powerful when combined with preprocessor variables (see page [245](#)).

## Example 1

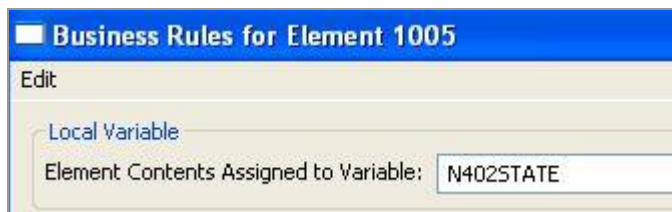
---

We display a message if the sales tax rate is incorrect for the state.

First, we create a map called `StateTax` containing sales tax rates for each state:



We set up a local variable on the State element:



Next, we go to the tax rate element and use the “Single” row at the top to see if the state is OH. If so, we check the current element against our StateTax’s “OH” index. If they don’t match, we display a message like “Tax rate is .055 for Ohio.”

The screenshot shows the 'Condition and Rule Definition' dialog box. Under 'When to Run Rule', 'Conditionally' is selected. The 'Single' row is active, showing a condition where 'N402STATE' is equal to 'OH'. Under 'What Rule to Run', '\* Call External Routine' is selected. The 'Server' is 'BusinessRules.Variable' and the 'Function' is 'CompareNumeric'. The 'Parameters' field contains: 'Current\_Element NE StateTax("OH") (BusinessRules.Utilities DisplayErrorByNumber 0 0 "Tax rate is %StateTax("OH")% for Ohio")'.

We repeat this for each state:

The screenshot shows the 'Condition and Rule Definition' dialog box for Kentucky (KY). The 'Single' row is active, showing a condition where 'N402STATE' is equal to 'KY'. The 'Server' is 'BusinessRules.Variable' and the 'Function' is 'CompareNumeric'. The 'Parameters' field contains: 'Current\_Element NE StateTax("KY") (BusinessRules.Utilities DisplayErrorByNumber 0 0 "Tax rate is %StateTax("KY")% for Kentucky")'.

## Example 2

Like Example 1, this displays a message if the sales tax rate is incorrect for the state, but it uses fewer rules.

First, we create the same map called StateTax containing sales tax rates for each state:

The screenshot shows a list of five business rules under the 'Business Rules' heading:

1. Run SetVar(StateTax("OH") ".055")StateTax("OH") ".055"
2. Run SetVar(StateTax("KY") ".06")StateTax("KY") ".06"
3. Run SetVar(StateTax("IL") ".0625")StateTax("IL") ".0625"
4. Run SetVar(StateTax("LA") ".04")StateTax("LA") ".04"
5. Run SetVar(StateTax("MY") ".00")StateTax("MY") ".00"

We set up a BusinessRule variable called StateCode on the State element:

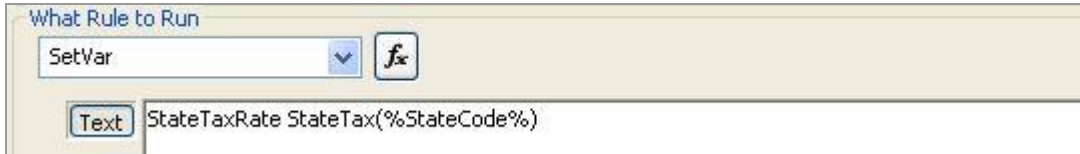


What Rule to Run

SetVar

Text StateCode

Next, we go to the tax rate element and use the StateCode element as an index into our map. We put the tax rate they should be using, considering their state, into a variable called StateTaxRate:

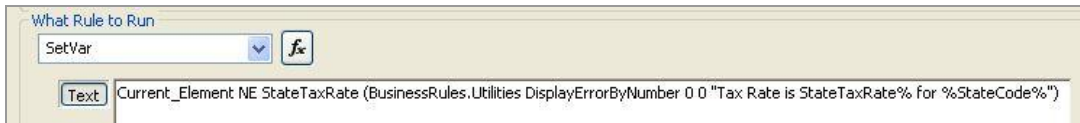


What Rule to Run

SetVar

Text StateTaxRate StateTax(%StateCode%)

In a second rule on the same element, we check the current element against what it should be (StateTaxRate). If it differs, we display a message that shows the correct state tax rate and the state:



What Rule to Run

SetVar

Text Current\_Element NE StateTaxRate (BusinessRules.Utilities.DisplayErrorByNumber 0 0 "Tax Rate is StateTaxRate% for %StateCode%")

This rule will take care of any state in our map. We do not need separate rules for each state.



# Appendix B: Validator Error Messages

---

---

## All validation programs

---

### Viewing TIBCO Foresight-Supplied Error Messages

Please see **ErrorMessageNumbers.pdf** for a list of error message ranges, the files that contain them, and what guidelines use them. These messages are used by the TIBCO Foresight-supplied business rules.

### Creating and Viewing your own Error Messages

To create your own custom error messages:

1. Look at the format of the error messages in **FSBRErrs.txt** in the Bin directory:

Each error message is on a separate line.

Each line starts with the number, then a *Tab*, then the text to be displayed during validation.

2. Use Notepad or another text editor to open the file **CustomerFSBRERRS.TXT** in the Bin directory. Use this file when you want to create your own error messages.

Follow the format you saw in **FSBRErrs.txt**. The lowest error number you can use is 32000, which already contains a dummy error message with text that you can change:

```
32000      Add your own error messages
  Tab  ↗
```

Change the text for error 32000. **Do not use any special characters such as exclamation marks.**

3. To add another error message, add a line for error 32001 followed by a *Tab* and the error text.

You can use the following number ranges for custom error messages:

32000-32999

60000 to 60999

4. Save and close the file.

You now have your own error message and can use it in business rules.

## Variables in Error Messages

---

You can use a `BusinessRules.Variable` in error messages by surrounding the variable with:

- # (pound signs) If the error message is in `CustomerFSBRErrs.txt`.
- % (percent signs) If the error message is hard-coded into the business rule.

This message displays variable `S2300CLM01ClaimNum`:

```
32222      Beginning of claim number #S2300CLM01ClaimNum#
```

The same message hard-coded into the business rule itself will look like this:



Either will display messages like these:

```
Beginning of claim number 2003051520001
Beginning of claim number 2003051520002
```

If variable `S2300CLM01ClaimNum` has no value, the message will display as one of these (depending on whether it is empty due to a `CLEAR` or an empty element):

```
Beginning of claim number S2300CLM01ClaimNum
Beginning of claim number
```

## Using your own Error Messages

To use your own error messages in a rule:

Use `BusinessRules.Utilities.DisplayErrorByNumber`

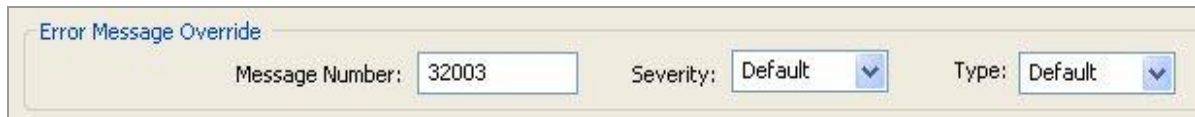
See page [166](#)

Use `Error Message Override`

See below

## Error Message Override

At the bottom of the Condition and Rule Definition box, you can override the message that would normally display if this rule is violated:



This is disabled for some data types.

To do this:

1. Choose any Result Type except **Invoke External Routine**.
2. If you selected **Set Usage/Status**, choose the Usage Setting. If the EDI data does not honor that setting, the error message set up below will be displayed. When you close the business rules box, the item's user attribute will be D for dependent, indicating that the user attribute will be the one in the business rule.

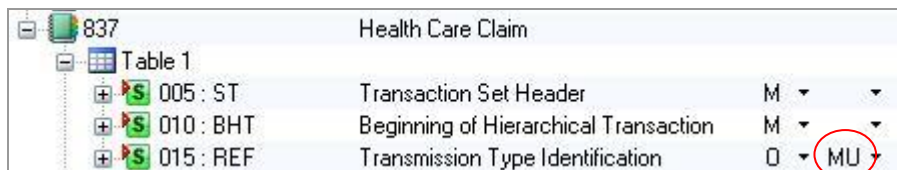
If you selected **Select Internal Code Set**, choose the set that should be used. If the value in the EDI is not in that set, the error message set up below will be displayed. This is available for rules on elements.

If you selected **Select Application Value List**, choose the value list that should be used. If the value in the EDI is not in that list, the error message set up below will be displayed. This is available for rules on elements.

3. Enter the number of your custom error message (32000 or higher) in the Message Number field at the bottom.
4. To override the default severity (Error), select a severity.
5. To override the default type (type 4), select a type.

### Example

This REF segment is marked as Must be Used:



| 837       | Health Care Claim                       |
|-----------|---|
| Table 1   |   |
| 005 : ST  | Transaction Set Header M                |
| 010 : BHT | Beginning of Hierarchical Transaction M |
| 015 : REF | Transmission Type Identification O MU   |

If not present in the data, any TIBCO Foresight validator will issue a generic message:

Missing Segment REF (Transmission Type Identification) at 1-015, though marked "Must Be Used"

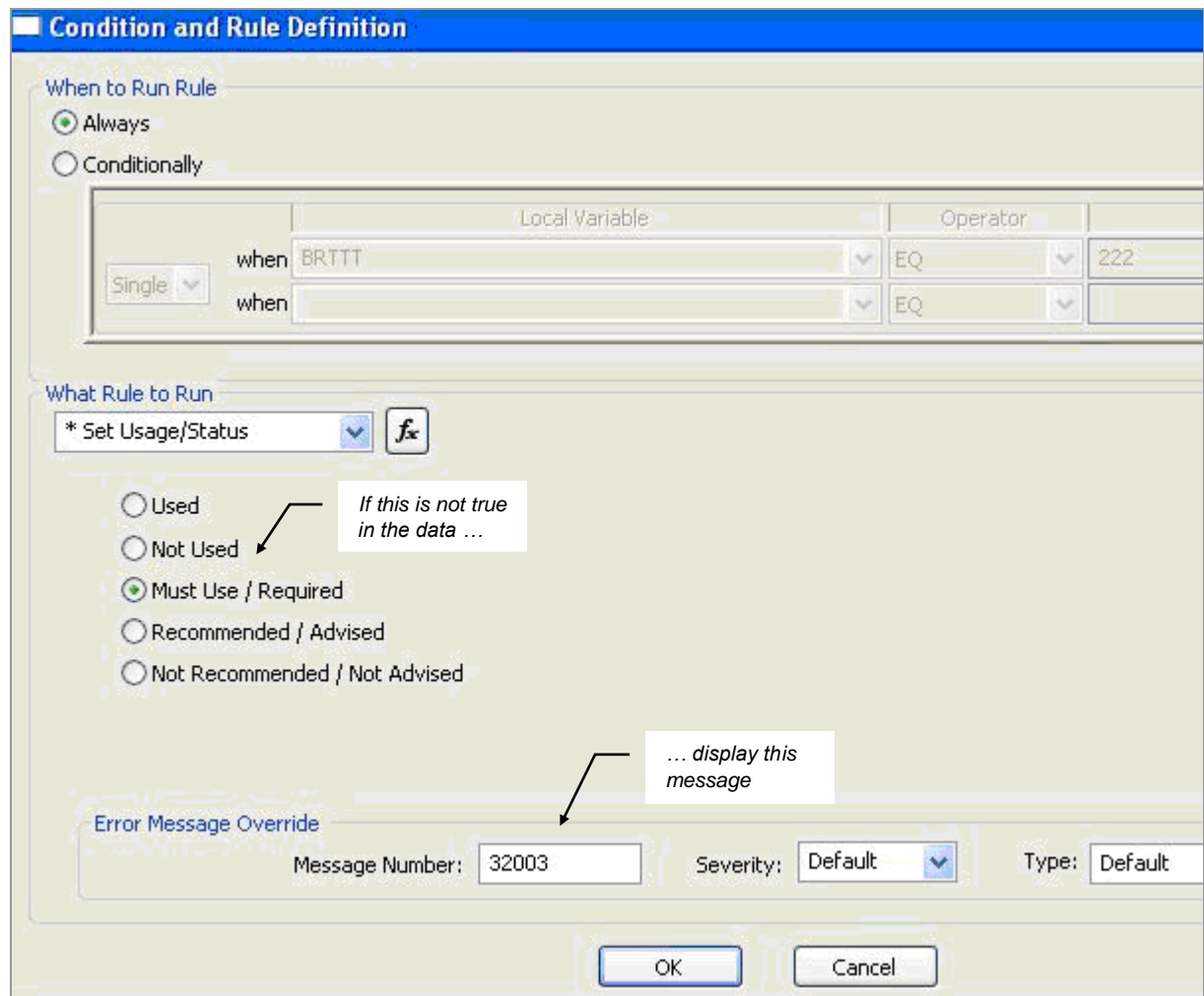
Instead, we want to display this message:

Missing REF segment at 1-015. Transaction set rejected.



To do this:

1. Add a business rule like this one to the REF segment:



The dialog box is titled "Condition and Rule Definition". It has two main sections: "When to Run Rule" and "What Rule to Run".

**When to Run Rule:**

- ☒ Always
- ☐ Conditionally

Below the radio buttons is a table with columns: "Local Variable", "Operator", and a value field.

|        | Local Variable | Operator |     |
|--------|----------------|----------|-----|
| Single | when BRTTT     | EQ       | 222 |
|        | when           | EQ       |     |

**What Rule to Run:**

\* Set Usage/Status

- ☐ Used
- ☐ Not Used
- ☒ Must Use / Required
- ☐ Recommended / Advised
- ☐ Not Recommended / Not Advised

**Error Message Override:**

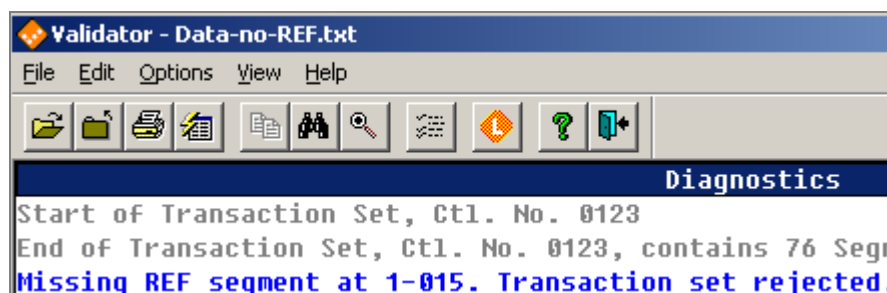
Message Number: 32003      Severity: Default      Type: Default

Buttons: OK, Cancel

*Annotations:*

- An arrow points from the text "If this is not true in the data ..." to the "Not Used" radio button.
- An arrow points from the text "... display this message" to the "Message Number" field.

2. Put this message in CustomerFSBRERRS.TXT:  
32003 Missing REF segment at 1-015. Transaction set rejected.
3. When validating with this guideline, your message will now look like this:

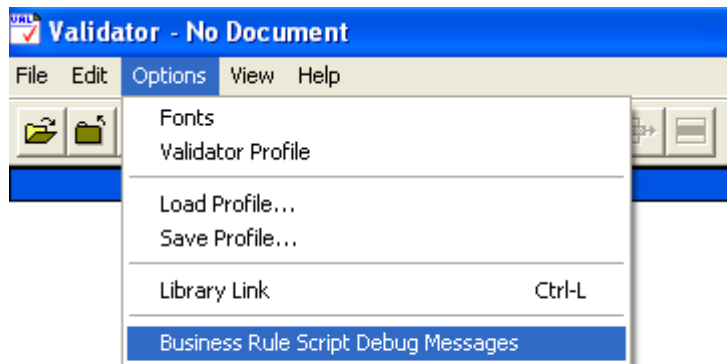


# Troubleshooting Custom Error Messages

- If your error message displays as a blank line in Validator, check your customer errors file to be sure you've separated the number and text with a *Tab*.
- If the error message does not display at all, check its format in your customer errors file and ensure that no special characters are included in the message text.
- Be sure that you have closed all error messages files before validating a file.
- Be sure \$dir.ini points to the file and the line is not commented out:

```
[ErrMsgFile]
ERRMSGFILE1 = "@\bin\FSANERRS.TXT"
ERRMSGFILE2 = "@\bin\FSBREERRS.TXT"
ERRMSGFILE3 = "@\bin\CustomerFSBREERRS.TXT"
```

- EDISIM Validator has a debug mode on the Options menu. Turn it on before validating:





# Appendix C: Code Tables

---

---

## Instream and HIPAA Validator Desktop

---

### Setting up your own Code Tables

To set up your own external code tables:

1. Create a text file containing your external code tables, as shown in Example A below.

You can copy SampleUserTable.txt to a new name and use it. Do not simply edit SampleUserTable.txt itself, since this is overwritten with each update.

Codes should be unique within a table.

2. Go to Instream's or HIPAA Validator Desktop's Bin directory and edit **\$Dir.ini** (Windows) or **fsdir.ini** (UNIX) with a text editor such as Notepad.

Find the [UserTables] section and be sure that the UserTable line is not preceded by a colon (which would comment out the line).

Change the path and filename to point to the location of the text file containing your external code tables. Example:

```
[UserTables]
UserTable="C:\Foresight\Desktop\Bin\OurUserTable.TXT"
```

You can use these code tables with the FindUserCode function (see page 65) and FindUserCodeWithDate function (see page 65).

It's important to know that the validation supports only one entry per key within external tables. If multiple entries for the same key are encountered in an external table, subsequent entries overwrite the previous entries, causing unexpected errors.

## Example A: External Code Table File

### Example contents of file

---

```
^USERAdmissionhour
C0|||Admission hour C0
CR|20020630|20021031|Admission hour CR
0A|20020531||Admission hour 0A
PI||20021130|Admission hour PI
PR|||Admission hour PR
^USERReasonCodes
00|||Reason Code Description 00
01|||Reason Code Description 01
02|||Reason Code Description 02
03|||Reason Code Description 03
04|20020630|20020930|Reason Code Description 04
05|||Reason Code Description 05
06|20020615||Reason Code Description 06
07|||Reason Code Description 07
08||20030710|Reason Code Description 08
09|||Reason Code Description 09
10|||Reason Code Description 10
11|||Reason Code Description 11
12|20020610|Reason Code Description 12
13|||Reason Code Description 13
```

### File Format

---

A code table starts with *^tablename*, where:

|                  |   |
|------------------|---|
| <i>^</i>         | A caret.  |
| <i>tablename</i> | Table's name, with no spaces or special characters. Suggestion:<br>HIPAA users can start your table names with USER to avoid using<br>the same name as a HIPAA table. |

Code lines have this format: *code | startdate | enddate | code description*, where:

|                         |  |
|-------------------------|--|
| <i>code</i>             | The actual code.   |
| <i>startdate</i>        | (optional) Date the code becomes effective. Use <i>yyyymmdd</i> format.  |
| <i>enddate</i>          | (optional) Date the code is no longer effective. Use <i>yyyymmdd</i><br>format.  |
| <i>code description</i> | The code's description.  |
|                         | All three vertical lines must be included, even if the code is always<br>effective and therefore has no start date and end date. |

## Extending existing HIPAA Code Tables

You can add to, modify, or delete codes in TIBCO Foresight-supplied HIPAA code tables, thus allowing these codes. For details, see **ExtendingCodeTables.pdf**.



# Appendix D: Complicated Rules

---

You may find situations in which you can either combine many tests and actions into one rule, or you can make separate rules for each one.

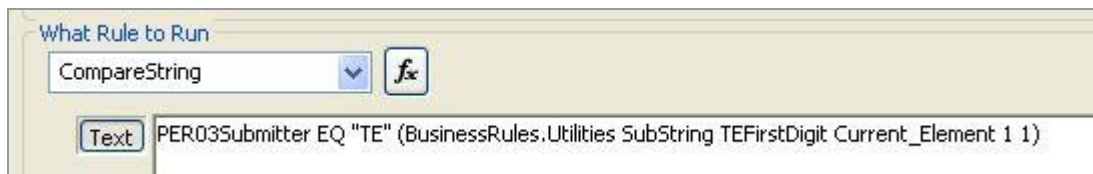
## Simple Rules

The typical format for a simple rule is:

*Test (action if true)*

The following example is a simple rule. It has one test and one action if the test condition evaluates to **true**.

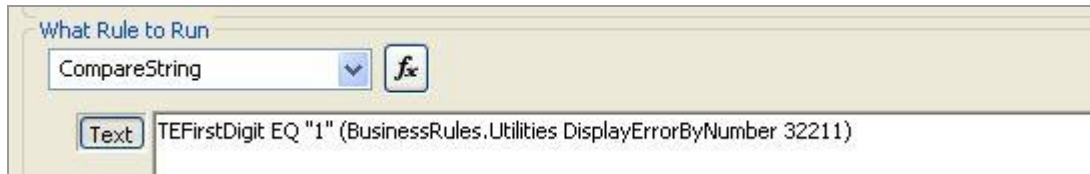
**What it does:** This CompareString function checks to see if the value in variable PER03Submitter contains the literal value TE. If so, it executes the action within the parentheses: take the first character in the current element's value and stores it in a variable TEFIRSTDigit.



You can add another simple rule to the same location. It might test the result of the rule above and take another action if the test condition evaluates to **true**. The next example does this.



**What it does:** The CompareString function checks the contents of variable TEFIRSTDigit (just created by the rule above) to see if it contains a 1. If so, it executes the action within the parentheses. (The action issues error message 32211.)



## Complex Rules

A typical format for a complex rule is:

*test1 (test2 (action if test1 AND test2 is true))*

The following example is a complex rule. It has a test plus an action if the test condition evaluates to **true**. Within *that* action is another test with action to be performed if it is true.

### Example 1: Two Conditions create an Error Message

This rule issues an error message if the NM108 is PI and the NM109 does not equal AB123.

|               |                                |
|---------------|--------------------------------|
| 2010BC (Loop) |                                |
| 015 : NM1     | Responsible Party Name         |
| 01 : 98       | Entity Identifier Code         |
| 02 : 1065     | Entity Type Qualifier          |
| 03 : 1035     | Name Last or Organization Name |
| 04 : 1036     | Name First                     |
| 05 : 1037     | Name Middle                    |
| 06 : 1038     | Name Prefix                    |
| 07 : 1039     | Name Suffix                    |
| 08 : 66       | Identification Code Qualifier  |
| 09 : 67       | Identification Code            |
| 10 : 706      | Entity Relationship Code       |
| 11 : 98       | Entity Identifier Code         |

1. Use SetVar to assign a variable to the **NM108** and call it **2010BCNM108PayName**. For details about using SetVar, see page [240](#).

2. Create this rule on the NM109.

**Condition and Rule Definition**

When to Run Rule

☒ Always

☐ Conditionally

| Local Variable | Operator | Constant |
|----------------|----------|----------|
| when           | EQ       |          |
| when           | EQ       |          |

What Rule to Run

CompareString

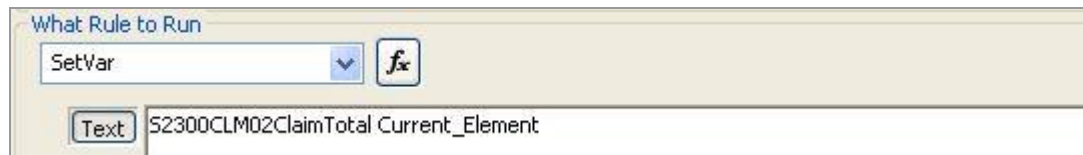
Text: 2010BCNM108PayName EQ "PI" (BusinessRules.Variable CompareString Current\_Element NE "AB123" (BusinessRules.Utilities DisplayErrorByNumber 32212))

| Parameter Text  | Explanation   |
|---|---|
| BusinessRules.Variable<br>CompareString<br>2010BCNM108PayName EQ "PI"   | Call the CompareString function to see if the current element contains a value of PI.<br>If true, continue by performing the action inside the parentheses.       |
| (BusinessRules.Variable<br>CompareString Current_Element NE<br>"AB123" (BusinessRules.Utilities<br>DisplayErrorByNumber 32212)) | Call the CompareString function to see if the current element does not contain AB123.<br>If true, continue by performing the action inside the inner parentheses. |
| (BusinessRules.Utilities<br>DisplayErrorByNumber 32212))  | Call the DisplayErrorByNumber function, to display the text of error 32212.   |

## Example 2: Using Rules in Loops

At the end of each repetition of the CLM loop in the data, we need to see if the claim total matches the total of the service lines. This example will show you how to do this.

1. Go to the CLM02 and use SetVar to assign a variable S2300CLM02ClaimTotal: This variable holds the claim total.

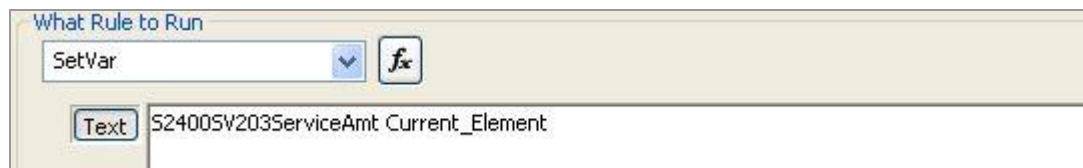


What Rule to Run

SetVar

Text S2300CLM02ClaimTotal Current\_Element

2. Go to the SV203 and use AddVar to total the service line amounts in a variable called S2400SV203ServiceAmt:



What Rule to Run

SetVar

Text S2400SV203ServiceAmt Current\_Element

3. Zero S2400SV203ServiceAmt on the CLM segment so that it will start over again for each repetition of the CLM loop:



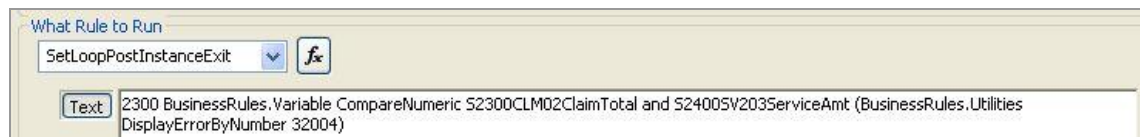
What Rule to Run

SetVar

Text S2400SV203ServiceAmt "0"

4. At the end of each repetition of the CLM loop in the data, see if the claim total matches the totaled service lines. To do this, we need to put a SetLoopPostInstanceExit early in the guideline, before the CLM loop. The usual place for these exits is on the ST segment.

This example rule executes at the end of each repetition of the CLM loop (loop 2300) and compares the numeric values in the variables S2300CLM02ClaimTotal and S2400SV203ServiceAmt. If they are not the same, an error message displays.



What Rule to Run

SetLoopPostInstanceExit

Text 2300 BusinessRules.Variable CompareNumeric S2300CLM02ClaimTotal and S2400SV203ServiceAmt (BusinessRules.Utilities.DisplayErrorByNumber 32004)

For another example of rule usage in loops, see Balance on page [207](#).

## Example 3: Adding and Comparing Numeric Values

In the 835 Health Care Payment Advice, the CLP03 is the amount submitted for a claim, and the CLP04 is the amount paid. If they are different, the differences must be specified in the CAS. The amounts in the CAS03, 06, 09, 12, 15, and 18 must equal the difference between the CLP03 and CLP04.

In other words:

$$\text{CLP03} - \text{CLP04} = (\text{CAS03} + \text{CAS06} + \text{CAS09} + \text{CAS012} + \text{CAS015} + \text{CAS018})$$

### A strategy:

1. Initialize CASTOT, CLP03, and CLP04 variables by setting them equal to 0 on the CLP segment, which is mandatory:

The image shows three sequential screenshots of the 'What Rule to Run' dialog box. Each screenshot has a 'What Rule to Run' header with a dropdown menu set to 'SetVar' and a function button 'fx'. Below the dropdown is a 'Text' field. In the first screenshot, the text field contains 'CASTOT "0"'. In the second screenshot, it contains 'CLP03 "0"'. In the third screenshot, it contains 'CLP04 "0"'. The dialog box has a light beige background and a thin border.

2. On the CLP03, put its value into variable CLP03:

The image shows a screenshot of the 'What Rule to Run' dialog box. The dropdown menu is set to 'SetVar' and the function button 'fx' is visible. The 'Text' field contains 'CLP03'. The dialog box has a light beige background and a thin border.

3. Add up amounts in CAS03, 06, 09, 12, 15, and 18 by putting this AddVar on each one:

The image shows a screenshot of the 'What Rule to Run' dialog box. The dropdown menu is set to 'SetVar' and the function button 'fx' is visible. The 'Text' field contains 'CASTOT'. The dialog box has a light beige background and a thin border.

4. On the CLP04, put its value into variable CLP04:

What Rule to Run

SetVar fx

Text CLP04

5. On the Patient Name NM1 (the next mandatory segment), see if  $CLP03 - CLP04 = CASTOT$ ; if not, display an error message:

What Rule to Run

Balance fx

Text CLP03 - CLP04 = CASTOT (BusinessRules.Utilities DisplayErrorByNumber 32005)

# Appendix E: ODBC Examples

---

---

## Instream and HIPAA Validator Desktop

---

### ODBC Tutorials and Demos

Please see page [111](#) for the format of each ODBC business rule.

This section includes two tutorials:

- A simple example that looks up the provider ID in an Access database and issues a message if it is not found. See ODBC Example 1 below.
- An extensive example starts on page [279](#).

HIPAA Validator Desktop and HIPAA Instream ship with ODBC demos:

#### **HIPAA Validator Desktop**

Validate 837I-Demo3DB.txt or 837I\_4010\_H\_Demo3DB.txt with guideline ODBCEX1 and then ODBCEX2.

#### **Instream**

Go to Instream's DemoData\ODBC directory and see the directions in \_readme\_ODBC.txt.

# ODBC Example 1

This example uses a sample Access database called **FSDemo.mdb** that has been installed in HIPAA Validator Desktop's DemoData directory and Instream's DemoData\ODBC directory.

You will use EDISIM to create rules on an 837I Addenda that take the provider ID from the EDI being validated and look it up in the database. If it is not in the database, your rule will display an error message.

|                            |                               |
|----------------------------|-------------------------------|
| Database                   | FSDemo.mdb                    |
| DSN                        | FSODBCdemo                    |
| Database name in rules     | MYFSdemo                      |
| Guideline containing rules | ODBCEX1                       |
| EDI file                   | 837I_4010_H_ErrorEvenClms.txt |

## Setting up a system DSN

Windows provides two applications for setting up ODBC Data Sources, one for 64-bit and one for 32-bit.

Instream requires a 64-bit ODBC driver

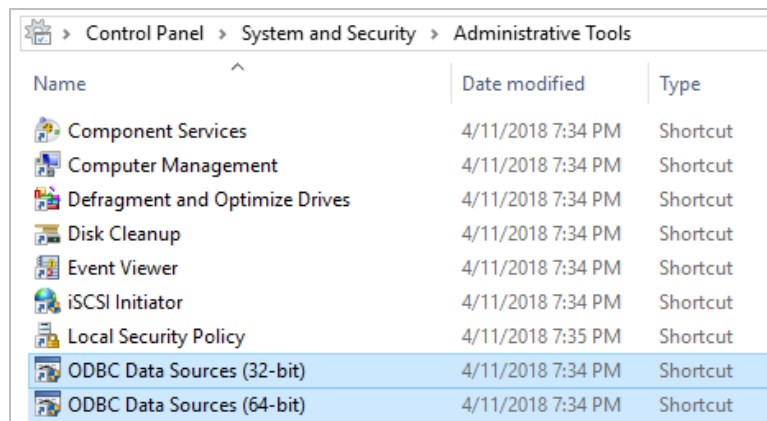
HIPAA Validator Desktop and EDISIM require 32-bit ODBC drivers.

Note that, if you have a combination of the above products (such as Instream and HIPAA Validator Desktop), you will need both drivers.

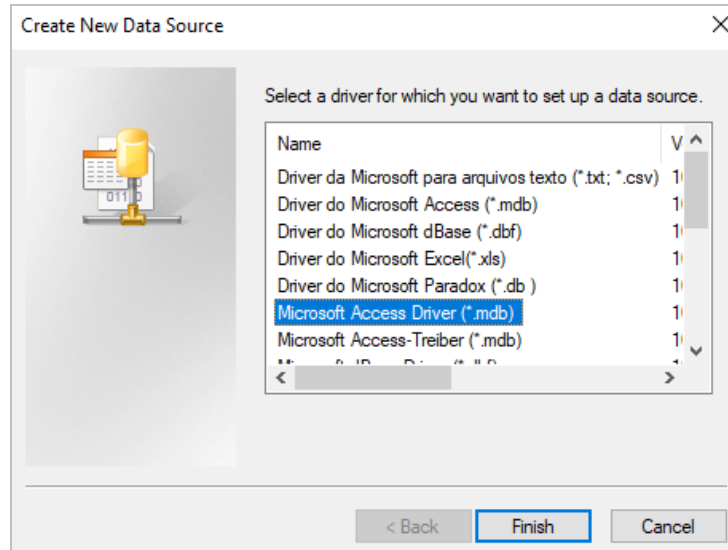
### Set up DSN Name

On the PC where you will run HIPAA Validator Desktop or Instream, set up a system-wide Data Set Name (DSN) called **FSODBCdemo** or **FSODBCdemo64** for the sample database:

- 1 Under **Control Panel**, choose **Administrative Tools** and select the appropriate ODBC Data Source (32- or 64-bit)



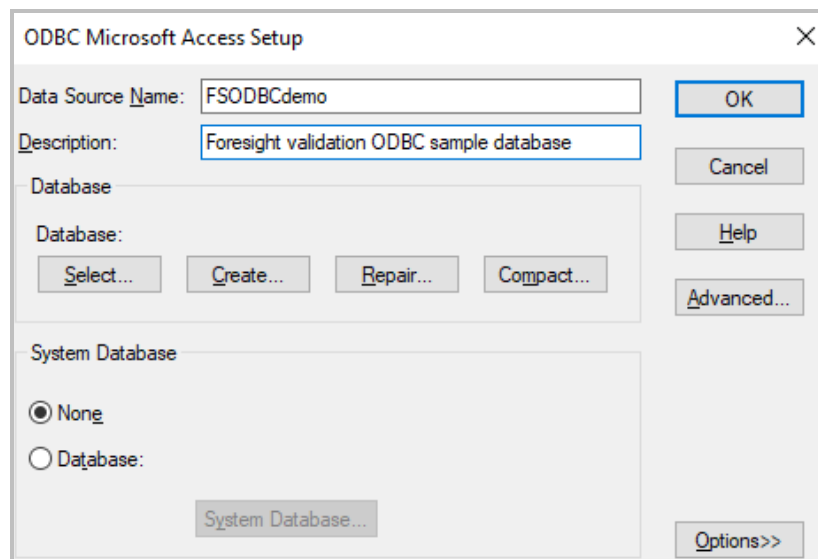
2. Choose the **System DSN** tab. Is there a listing for **FSODBCdemo/FSODBCdemo64**?
  - Yes – your DSN is already set up and you can proceed to Looking at the Database on page 274.
  - No – Choose **Add | Microsoft Access Driver (\*.mdb) | Finish** and continue with Step 4.



**Note:** If the **Microsoft Access Driver (\*.mdb)** driver name is not listed, you must install the correct ODBC driver for your operating system. (Refer to <https://www.microsoft.com/en-us/download>).

3. Fill out the fields as follows, and then click **Select**.

**Note:** Use FSODBCdemo for a 32-bit data source or FSODBCdemo64 for a 64-bit data source.





4. Navigate to HIPAA Validator Desktop's DemoData directory or Instream's DemoData\ODBC directory and choose **FSDemo.mdb**. Click **OK** until you have closed out of all Control Panel dialog boxes.

(These files are exactly the same for HIPAA Validator Desktop and Instream; if you are using the demos for both products, choose either one.)

The DSN name **FSODBCdemo** is like a nickname for the database file. If you move the database, you need only change the path under **Control Panel | Administrative Tools | Data Sources (ODBC)**. No business rules will have to be changed.

It is possible to hard-code the database path in your business rules, but setting up a system DSN as shown above is preferable. It enables you to move the database, and store the database in different places on different validation machines, without editing business rules.

### **Edit \$Dir.ini**

Edit \$Dir.ini in HIPAA Validator Desktop or Instream's Bin directory and give the DSN a name:

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

## **Looking at the Database**

1. Open the Access database **FSDemo.mdb** in the HIPAA Validator Desktop's DemoData directory or Instream's DemoData\ODBC directory. Look at the **Master** table.

| Master : Table |      |           |            |        |            |
|----------------|------|-----------|------------|--------|------------|
|                | Type | ID        | Name       | Status | StatusDate |
| 24             |      | 432021111 | Jones, Sam | A      | 1/1/2004   |
| 24             |      | 432032222 | Jones, Tom | I      | 11/1/2003  |

This exercise checks the provider ID in an EDI file to see if it appears in column 2 of this database. If not, an error message appears.

2. Close the database.

## Setting up the Error Message

1. Back up your current **CustomerFSBRERRS.TXT** and **\$dir.ini** files in HIPAA Validator Desktop's or Instream's **Bin** directory.
2. Check **\$dir.ini** to see that  
`ERRMSGFILE3 ="@\\bin\\CustomerFSBRERRS.TXT"`  
and that the line is not preceded with a colon (which would comment it out).
3. Edit **CustomerFSBRERRS.TXT** and add this error message all on one line (use a *Tab* after the number):

```
35000 Database Lookup Error: The Pay-To Provider ID is not found in the  
Master Table in FSDemo.MDB.
```

4. Save and close the file.

We will create a rule that uses this error message.

## Creating the Rules

### Note

To import a guideline that already has these rules created, open Standards Editor and choose **File | Import | Import Single SEF and open**. Go to EDISIM's **AddlStds** directory and select **ODBCEX1.sef**.

Create rules that check the provider ID and issue a message if it is not found in FSDemo.mdb.

- 1 In Standards Editor, create a new guideline based on **837AQ320** and save it as **ODBCEX1**.
- 2 On the ST segment, create a business rule that opens FSDemo.mdb by using its DSN name **FSODBCdemo**.

This rule names the database MYFSDemo and that is a name you will use for it in any rules throughout the guideline.

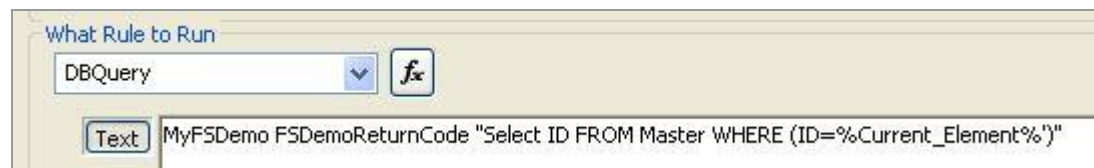


- 3 The next rule compares the EDI data in 2010AA NM1-09:

NM1\*85\*2\*JONES\*\*\*\*\*24\*432021111~

... to the second column in the database's Master table.

To set up this checking, see if the EDI value in 2010AA NM1-09 is in column 2 of the database by adding this rule to 2010AA NM1-09:



Note the double quotes around the Select statement and the single quotes around %Current\_Element%.

This says to look in MYFSDemo, check the Master table, and find records where the EDI value of the current element matches some value in the ID column of the database. Put the number of matched database records in the variable **FSDemoReturnCode**.

- 4 The next rule, on the same element, issues message 35000 if the number of records matched equals zero.



5. Finally, go to the SE segment and close the database:



What Rule to Run

DBCclose

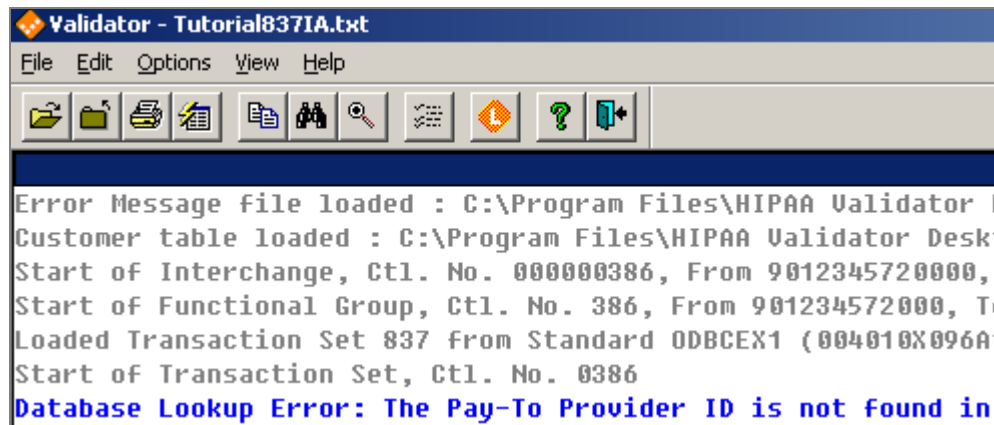
Text MYFS Demo

6. Save the guideline.

## Testing the Rules

### HIPAA Validator Desktop

- 1 Find EDISIM's **User Files\Public Guidelines\ODBCEX1.std** and copy it to HIPAA Validator Desktop's **Database** directory for testing.
- 2 From within HIPAA Validator Desktop, open **837I\_4010\_H\_ErrorEvenClms.txt** or **Tutorial837IA.txt** in HIPAA Validator Desktop's DemoData directory.
- 3 Choose **ODBCEX1** for the guideline. After validation, look for the error message "*Database Lookup Error: The Pay-To Provider ID is not found in the Master Table in FSDemo.MDB.*" This is our rule in action.



### Instream

- 1 Find EDISIM's **User Files\Public Guidelines\ODBCEX1.STD** and copy it to Instream's **Database** directory for testing.
- 2 Go to Instream's DemoData\ODBC directory and check the paths in ODBC\_EX1.bat. Save and close the file.
- 3 Execute ODBC\_EX1.bat.
- 4 After validation, look in Instream's Output directory for results file **837I\_4010\_H\_ErrorEvenClms\_Results.txt**. Search for "Database Lookup Error." This is our rule in action.

## ODBC Example 2

In this example, the Bill-To and Pay-To provider numbers will be validated against the same Access database table that was used in Example 1. Providers are first checked to see if they are in the database table, and an error is displayed if they are not. The rules then check a 'Status Flag' returned from the database to see if they are inactive. An error message is displayed if the provider is inactive.

We assume FSDemo.mdb has a DSN name of **FSODBCdemo**. If not, please follow the steps on page [272](#).

Database.....FSDemo.mdb  
DSN.....FSODBCdemo  
Database name in rules.....MYFSdemo  
Return code variable .....DBResultVar  
Guideline containing rules .....ODBCEX2

The rules have already been placed in the guideline ODBCEX2, which is in HIPAA Validator Desktop's Database directory and Instream's DemoData\ODBC directory.

## Running the Demo in HIPAA Validator Desktop

1. Set up the custom error messages:
  - a. Back up CustomerFSBREERRS.TXT file in HIPAA Validator Desktop's Bin directory.
  - b. Open Add-to-CustomerFSBREERRS.TXT in HIPAA Validator Desktop's DemoData directory and copy the contents to the end of your CustomerFSBREERRS.TXT.
  - c. Close both TXT files.
2. Validate data using the guideline:
  - a. Open HIPAA Validator Desktop.
  - b. Open **837I-Demo3DB.txt** or **837I\_4010\_H\_Demo3DB.txt** in HIPAA Validator Desktop's DemoData directory.
  - c. Use guideline **ODBCEX2**.
  - d. Check Use for all occurrences of 004010X096A1.
3. When complete, view the ODBC messages by looking for messages that start with 'ODBC Message:'

This demo data file contains four claims:

- The first provider's number is OK.
- The second provider is on file but not active.
- The remaining two providers are not on file.

## Running the Demo in Instream

Go to Instream's DemoData\ODBC directory and follow the directions in \_readme\_ODBC.txt.

## The Rules that made it happen

How did we accomplish this? Go to Standards Editor to see.

1. In Standards Editor, choose **File | Import | Import Single SEF and open** and import **ODBCEX2** from HIPAA Validator Desktop's **DemoData** directory.
2. Look at the business rules described below.

### ***ST segment***

This business rule on the ST segment opens the database and nicknames it **MYFSdemo** within the guideline.

To see it, right-click on the **ST** segment, chose **Business Rules**, click on the rule, and choose **Edit**.



Where:

|                    |  |
|--------------------|--|
| <b>MYFSdemo</b>    | The name to be used for this database connection in subsequent ODBC Business Rules in this guideline.                          |
| <b>DBResultVar</b> | A variable name to hold a return code. If this return code is not zero, then an error occurred. See page <a href="#">140</a> . |
| <b>FSODBCdemo</b>  | The DSN name for database FSDemo.mdb. This name was previously assigned through control panel (see page <a href="#">272</a> ). |



## 2010AA NM1-08 Element

This business rule on the Billing Provider Name's identification code qualifier stores the contents of this element into variable **BillProvIDQual**:



What Rule to Run

SetVar

Text BillProvIDQual

## 2010AA NM1-09 Element

### Rule #1

---

This business rule on the Billing Provider Name's identification code stores the contents of this element into variable **BillProvID**:



What Rule to Run

SetVar

Text BillProvID

### Rule #2

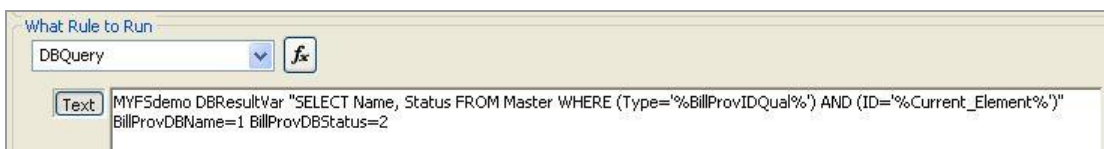
---

This rule executes a SELECT statement against the MYFSDemo database.

Before executing the statement, HIPAA Validator Desktop replaces the placeholders:

**%BillProvIDQual%** with the contents of the BillProvIDQual variable (see the rule on the NM1-08)

**%Current\_Element%** with the contents of the current element (the provider's ID).



What Rule to Run

DBQuery

Text MYFSDemo DBResultVar "SELECT Name, Status FROM Master WHERE (Type='%BillProvIDQual%') AND (ID='%Current\_Element%')"  
BillProvDBName=1 BillProvDBStatus=2

Where:

|                         |   |
|-------------------------|---|
| <b>Name and Status</b>  | Field names on the Master table from the FSDemo.mdb file.   |
| <b>DBResultVar</b>      | Variable to contain the number of records that matched during the SELECT. If this returns a negative number, it is an error code (see page 143).                          |
| <b>BillProvDBName=1</b> | Assigns whatever is in Field #1 of the first record selected to variable BillProvDBName. Field #1 in this SELECT example is the Name field (SELECT <u>Name</u> , Status). |

**BillProvDBStatus=2** Assigns the contents of Field #2 (SELECT Name, Status) from the first record selected to variable BillProvDBStatus.

If no records are found, then the two assignment variables are cleared.

### Rule #3

---

This rule compares the contents of variable DBResultVar with the string '0' (zero).

If equal, no records matched the ID and Qualifier of the provider, and error message 32501 is displayed.



Where:

**DBResultVar** Variable containing the number of records selected by DBQuery in the previous rule.

**32501** Error message in CustomerFSBRErrs.txt.

### Rule #4

---

This rule compares the contents of variable BillProvDBStatus with the letter 'I'. If equal, it means that the first matching Provider record had a status of Inactive, and error message 32502 is displayed.



Where:

**BillProvDBStatus** Variable containing the first selected record's Status field from the DBQuery in the Rule 2 above.

**32502** Error message telling the user that the Provider is Inactive (from CustomerFSBRErrs).

## Rule #5

---

This rule compares the contents of variable with the letter 'A'. If equal, the first matching Provider record had a status of Active, and message 32503 is displayed.



What Rule to Run

CompareString

Text: BillProvDBStatus EQ "A" (BusinessRules.Utilities.DisplayErrorByNumber 32503)

Where:

**BillProvDBStatus** The variable containing the first selected record's Status field from the DBQuery in the Rule 2 above.

**32503** "Error" message telling the user that the Provider is OK (from CustomerFSBRErrs).

## SE Segment

This rule closes the database.



What Rule to Run

DBCLOSE

Text: MYFS Demo

## 2010AB NM1 Rules

The guideline has another series of the same rules on the Pay-To Provider. The rules are the same but the variable names are different. The data files **837I-Demo3DB.txt** and **837I\_4010\_H\_Demo3DB.txt** do not have a Pay-to Provider, so no messages were generated by HIPAA Validator Desktop.

# Appendix F: Guideline Merge

---

---

(Windows Only) Instream, HIPAA Validator Desktop, and EDISIM

---

## Overview

Guideline Merge lets you merge the changes that you have made to your guideline in EDISIM Standards editor with:

- TIBCO Foresight-supplied guidelines containing the full set of HIPAA rules
- Another EDI or flat file guideline with the same structure.

This lets you have a “master” guideline and then separate guidelines with additional rules for specific partners.

Please see **GuideMerge.pdf** for details.



# Appendix G: Debug

---

## EDISIM Validator Debug

Enable **Options | Business Rule Script Debug Messages** before validating.

This menu choice is a toggle. Your current setting will stick until you change it.

After validating, green debug messages show the steps Validator takes to enforce your business rules.

To prevent some business rules from displaying debug messages, use a text editor to edit BusinessRules.properties in EDISIM's Bin directory. Set rules to 0 if you don't want them to show debug messages.

## HIPAA Validator Desktop and Instream Debug

**Important**      Setting up debug reduces performance, even if the error messages are not being displayed or output.

During validation, you can display debug messages like these:

```
**FS Debug DLL Invoking SetVar: S2300DTP02StmDt
**FS Debug DLL Invoking FindCodeWithDate: FacilityType S2300CLM0501FacilityType
**FS Debug DLL GetVarAsString: Variable S2300DTPStatementDate Does not exist in
**FS Debug DLL FindCodeWithDate: Exiting No DateToCheck to Lookup in FacilityTy
**FS Debug DLL Invoking CompareString: S2300DTP02StmDt EQ "D8" (BusinessRules.
**FS Debug DLL Compare String D8 EQ D8 is TRUE
**FS Debug DLL Invoking Action: (BusinessRules.CodeLookup FindCodeWithDate Faci
**FS Debug DLL Invoking Function: BusinessRules.CodeLookup FindCodeWithDate Faci
```

To use debug:

1. Be sure that the file **BusinessRule.properties** is in the Bin directory. If not, contact TIBCO Foresight Technical Support.
2. To see the debug messages:

**In HIPAA Validator Desktop**, choose **Options | Validator Profile | Filter** and select the checkbox for "User #1" types.

**In Instream** (Windows only), go to the \Bin directory and open \$fsdeflt.apf or the other APF file that you are using. Change WT\_User1=0 to WT\_User1=1.

3. Validate a file and note the extra debugging messages.
4. To stop displaying the debug messages:

In Validator HIPAA Validator Desktop, clear the checkbox for **“User #1” types**.

In Validator Instream, open \$fsdeflt.apf and change **WT\_User1=1** to **WT\_User1=0**.

You can use DumpVars to display variables and their contents at a particular location. See Divide on page 205 for details.

# Appendix H: Troubleshooting Checklist

---

## **If your rule does not fire at all:**

- Be sure the data should have triggered the rule.
- Be sure you saved your guideline.
- Close and re-open Validator.
- Be sure that the rule is in the proper location.
- Check spelling and capitalization of variables and other parts of business rules.
- Be sure that literals are surrounded by quotes and variables aren't.
- Check parentheses.
- Ensure that a space appears before and after each operator.
- See if you need to clear a list or a variable in a repeating loop.
- Turn on debug in Validator (**Options | Business Rules Script Debug Messages**) and revalidate the file.
- In Standards Editor, use Print | Print Rules | by Variable.
- Add DumpVars rules.
- Which validator are you using: EDISIM Validator, HIPAA Validator Desktop, Instream, or Analyzer? If you have HIPAA Validator Desktop installed, the icon appears on the Standards Editor toolbar.

## **If the rule fires but you get a blank line:**

- Close your error messages file.
- Check your customer errors file to see if one and only one rule with that number is there.
- Be sure there is a tab between the number and the text.
- Be sure you saved and closed the message file.
- Check the path to the customer errors file in \$dir.ini.



- Which validator are you using: EDISIM Validator, HIPAA Validator Desktop, Instream, or Analyzer? If you have HIPAA Validator Desktop installed, the icon appears on the Standards Editor toolbar.

Other things to check:

- If you are validating with Instream or HIPAA Validator Desktop, be sure that the guideline and customer errors file has been copied to its Database directory and its \$Dir.ini updated accordingly.
- Be sure that the business rule is added to all locations where it is needed. For instance, 837s have claim loops at the subscriber and patient level. Be sure your rules are in both claim loops.
- When running rules in Analyzer, be sure that the business rules that you use are supported by Analyzer.
- See if you need to clear a list or a variable in a repeating loop. See page [242](#).

# Appendix I: Processing Order

---

Demo      Please copy **837P\_5010\_ProcessingOrder.std** from EDISIM's Samples directory or Instream's DemoData directory to EDISIM's User Files\Public Guidelines directory and validate a 5010 837P.

1. As validation processes each **segment**:

It executes the business rules that are directly attached to it, in top-down order if there are multiple rules.

It then executes the SetSegmentPreExits for that segment in bottom-up order.

2. It then looks at each **element** in the segment.

It executes any business rules directly attached to the first **element** in the segment, in top-down order if there are multiple rules.

It then executes SetElementPostExit rules for the first element in bottom-up order.

Likewise, it continues to the next element in the segment.

3. If the segment is the last one in a **loop**, the validator then processes any SetLoopPostInstanceExit rules for that loop in bottom-up order.

At the end of all iterations of the loop, it processes the SetLoopPostExit rules for that loop in bottom-up order.

4. For nested loops that end on the same segment (like 837 2300 and 2400 loops), the inner loop is processed first.

The bottom-up rules include:

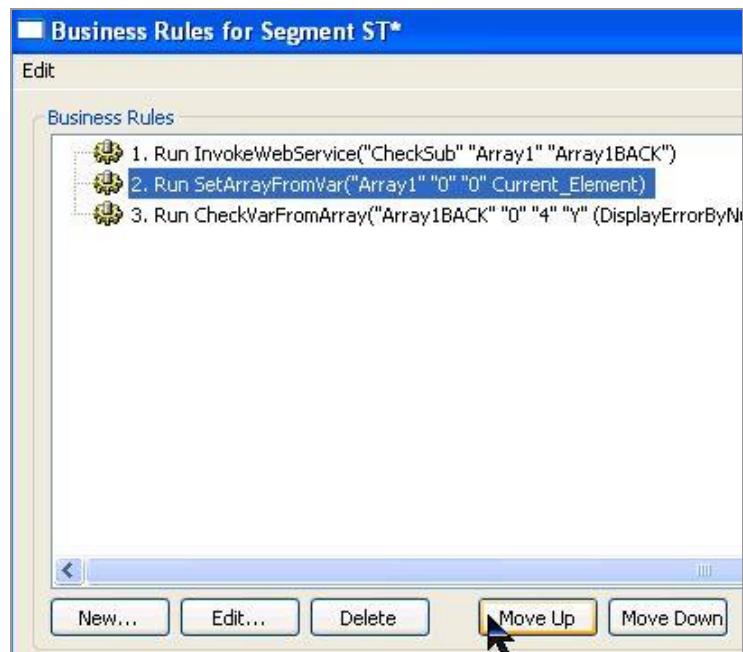
SetCompositePreExit  
SetElementPostExit  
SetLoopPostExit  
SetLoopPostInstance Exit  
SetSegmentPreExit

The **BusinessRules.Exits KeepOrder** rule causes them to execute in top-down order. See KeepOrder on page 99 for details.

Please see **KeepOrder** on page 99 for a way to force top-down processing of loop Exit rules.

## Rearranging Business Rules

You can use the Move Up and Move Down buttons in the business rules box to rearrange rules so that they process in a different order:



### Example 1. Processing Order of Rules

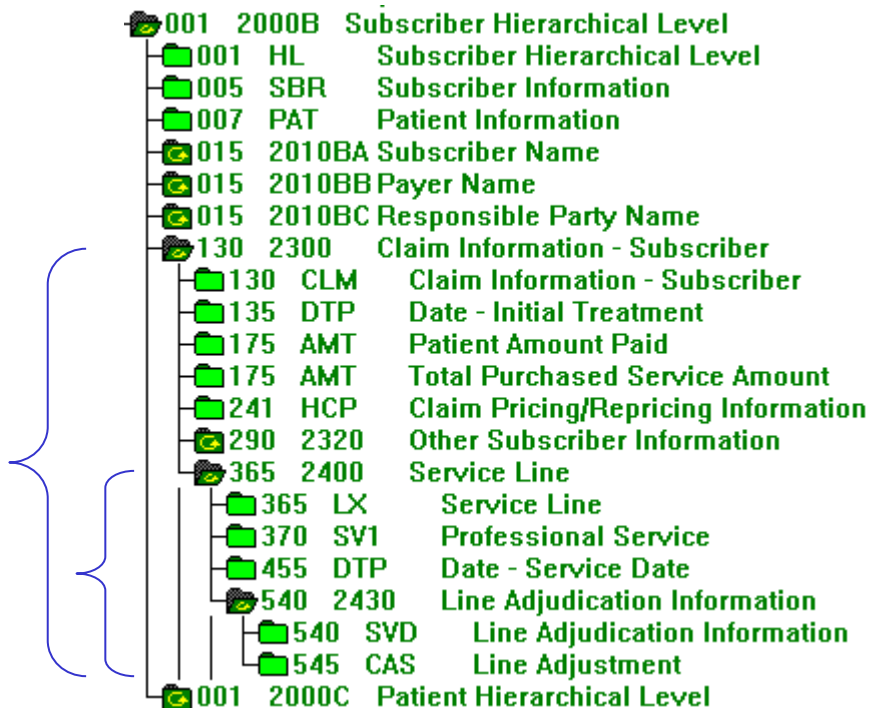
|     |        |  |
|-----|--------|--|
| 005 | ST     | Transaction Set Header                     |
| 010 | BHT    | Beginning of Hierarchical Transaction      |
| 015 | REF    | Transmission Type Identification           |
| 020 | 1000A  | Submitter Name                             |
| 020 | 1000B  | Receiver Name                              |
| 001 | 2000A  | Billing/Pay-to Provider Hierarchical Level |
| 001 | HL     | Billing/Pay-to Provider Hierarchical Lev   |
| 003 | PRV    | Billing/Pay-to Provider Specialty Inform   |
| 010 | CUR    | Foreign Currency Information               |
| 015 | 2010AA | Billing Provider Name                      |
| 015 | 2010AB | Pay-to Provider Name                       |
| 001 | 2000B  | Subscriber Hierarchical Level              |
| 001 | HL     | Subscriber Hierarchical Level              |
| 005 | SBR    | Subscriber Information                     |
| 007 | PAT    | Patient Information                        |
| 015 | 2010BA | Subscriber Name                            |
| 015 | 2010BB | Payer Name                                 |
| 015 | 2010BC | Responsible Party Name                     |
| 130 | 2300   | Claim Information - Subscriber             |
| 130 | CLM    | Claim Information - Subscriber             |
| 135 | DTP    | Date - Initial Treatment                   |
| 250 | 2310A  | Referring Provider Name                    |
| 250 | 2310B  | Rendering Provider Name                    |
| 290 | 2320   | Other Subscriber Information               |
| 365 | 2400   | Service Line                               |

| Assume these rules |                                  | Execution order |
|--------------------|----------------------------------|-----------------|
| ISA                | rule 1                           | rule 1          |
| ISA05              | rule 2                           | rule 2          |
|                    | rule 3                           | rule 3          |
| ST                 | rule 4                           | rule 4          |
|                    | rule 5                           | rule 5          |
|                    | Exit instance for 2310A, rule 6  | rule 8          |
|                    | Exit instance for 2310A, rule 7  | rule 11         |
|                    | rule 8                           | rule 12         |
|                    | Exit instance for 2310B, rule 9  | rule 13         |
|                    | Exit instance for 2310C, rule 10 | rule 7          |
|                    | rule 11                          | rule 6          |
| BHT                | rule 12                          | rule 9          |
| BHT01              | rule 13                          | rule 10         |
|                    | rule 14                          | rule 14         |

In this case, the two business rules (6 and 7) written for Loop 2310A will process from bottom up.

## Example 2. Two loops ending on same segment

Loop 2300 and its nested 2400 loop end on the same segment.



When two loops end on the same segment, the rules at the end of the loop execute in this order:

1. Rules placed directly on the segment.
2. Rules placed directly on the elements in the segment.
3. Exit rules for the inner loop (2400), in reverse order if there are more than one.
4. Exit rules for the outer loop (2300), in reverse order if there are more than one.

**Assume these rules:**

|       |                                |         |
|-------|--------------------------------|---------|
| ST    | rule 1                         | rule 1  |
|       | rule 2                         | rule 2  |
|       | Exit instance for 2300, rule 3 | rule 7  |
|       | Exit instance for 2300, rule 4 | rule 8  |
|       | Exit instance for 2400, rule 5 | rule 9  |
|       | Exit instance for 2400, rule 6 | rule 10 |
|       | rule 7                         | rule 6  |
| BHT   | rule 8                         | rule 5  |
| CAS   | rule 9                         | rule 4  |
| CAS19 | rule 10                        | rule 3  |
| SE    | rule 11                        | rule 11 |

\* Since the 2400 loop is inside the 2300 loop, it will execute before rules on the 2300.

# Appendix J: LookAhead and Array Extended Example

---

## Array, Lookahead, and Web Services Demos

**This demo is for experienced business rule developers.**

This example is based on a 4010 837P guideline called WEBSRV1\_837P (in Instream's and HIPAA Validator Desktop's DemoData directories).

To see rules that are already in this guideline:

- Import this file into EDISIM Standards Editor and look at the rules.
- Copy the guideline to HIPAA Validator Desktop's Database folder and then validate a 4010 837P with debug turned on.

In this example, we simulate seeing if the subscribers and dependents were enrolled on the dates provided in the data, and display messages if not. The tricky part is knowing the dates, which are far down in the data, when you are validating the identification codes near the top of the data.

To do this, we capture names, ID's and dates in an array on a first pass through the data. We then send our demo array to a web service to check our "database" and send back an array that tells us if they were covered on the dates provided.

We then use additional business rules to check the returned array and display error messages if necessary.

Lookahead ranges are:

- 2000A (range ends at 2000B Lookahead start)
- 2000B through end of loop

Please see these pages for additional information:

|                            |                          |
|----------------------------|--------------------------|
| Current_LoopCount.....     | page <a href="#">27</a>  |
| Current_LoopKey.....       | page <a href="#">27</a>  |
| Array Business Rules ..... | page <a href="#">34</a>  |
| Lookahead.....             | page <a href="#">123</a> |

# Summary of Rules - Top-Down

This set of rules is an example only. It will not actually run at your site since your web services will have different names and perform different functions. These are the rules in the order in which they appear in the sample guideline WEBSRV1\_837P (in the DemoData directory).

The next section (page 300) shows the same rules in the order in which they will execute, along with annotation.

| Segment | Rules  |
|---------|--|
| ST      | (Table 1 Transaction Set Header)<br>BusinessRules. Array. CreateArray: WSin<br>BusinessRules. Exits. SetLoopPostInstanceExit: 2000B BusinessRules.Lookahead ExitLookahead<br>BusinessRules. Exits. SetLoopPostInstanceExit: 2000B BusinessRules.Lookahead InvokeWebService WEBSRV1 WSin "WSout" (BusinessRules.Utilities DisplayErrorByNumber 29001)<br>BusinessRules. Array. CreateArray: WSout<br><br>----- Start of first Lookahead range ----- |
| HL      | (2000A Billing/Pay-to Provider Hierarchical Level)<br>BusinessRules. Lookahead. ClearArray: WSin<br>BusinessRules. Lookahead. ClearArray: WSout  |
| NM1     | (2010AA Billing Provider Name)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 0 0 "2"  |
| NM109   | (2010AA Identification Code)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 0 1 Current_Element<br>BusinessRules. Array. CheckVarFromArray: WSout 0 4 "0" (BusinessRules.Utilities DisplayErrorByNumber 32160)   |
| REF02   | (2010AA Reference Number)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 0 2 Current_Element   |
| NM1     | (2010AB Pay-to Provider Name)<br>BusinessRules. Lookahead. RunNoData: (BusinessRules.DBServer InvokeWebService WEBSRV1 WSin "WSout" (BusinessRules.Utilities DisplayErrorByNumber 29001))<br>BusinessRules. Lookahead. RunNoData: (BusinessRules.Array ExitLookahead)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 1 0 "2"   |

**Segment   Rules**

---

|  |   |
|--|---|
| NM109  | (2010AB Identification Code)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 1 1 Current_Element<br>BusinessRules. Array. CheckVarFromArray: WSout 1 4 "0" (BusinessRules.Utilities<br>DisplayErrorByNumber 32161)   |
| REF02  | (2010AB Reference Number)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 1 2 Current_Element<br>BusinessRules. Lookahead. InvokeWebService: WEBSRV1 WSin WSout (BusinessRules.Utilities<br>DisplayErrorByNumber 29001)<br>BusinessRules. Lookahead. ExitLookahead:  |
| <hr/> <b>----- Start of second Lookahead range -----</b> <hr/> |   |
| HL   | (2000B Subscriber Hierarchical Level)<br>BusinessRules. Lookahead. ClearArray: WSin<br>BusinessRules. Lookahead. ClearArray: WSout  |
| NM103  | (2010BA Name Last)<br>BusinessRules. Array. CheckVarFromArray: WSout 0 3 Current_Element (BusinessRules.Utilities<br>DisplayErrorByNumber 32152)  |
| NM104  | (2010BA Name First)<br>BusinessRules. Variable. SetVar: SubFirstName Current_Element  |
| NM109  | (2010BA Identification Code)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 0 1 Current_Element<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 0 0 "1"<br>BusinessRules. Array. GetVarFromArray: WSout 0 1 "SubscriberID"<br>BusinessRules. Variable. CompareNString: SubscriberID "NE" Current_Element (1;1;3;)<br>(BusinessRules.Utilities DisplayErrorByNumber 32150)<br>BusinessRules. Variable. CompareNString: SubscriberID "NE" Current_Element (4;4;9;)<br>(BusinessRules.Utilities DisplayErrorByNumber 32151) |
| DMG02  | (2010BA Date Time Period)<br>BusinessRules. Variable. SetVar: DMG02 Current_Element   |
| DMG03  | (2010BA Gender Code)<br>BusinessRules. Array. SearchVarsInArray: WSout "1" "" "" SubFirstName Current_Element DMG02<br>(BusinessRules.Utilities DisplayErrorByNumber 32153)   |



| <b>Segment</b> | <b>Rules</b>   |
|----------------|--|
| DTP03          | (2300 Date Time Period)<br>BusinessRules. Lookahead. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element "D8"   |
| NM1            | (2310B Rendering Provider Name)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 1 0 "2"   |
| NM109          | (2310B Identification Code)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 1 1 Current_Element<br>BusinessRules. Array. SearchVarsInArray: WSout "2" Current_Element "" "" "0" (BusinessRules.Utilities DisplayErrorByNumber 32162)                                |
| REF02          | (2310B Reference Number)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 1 2 Current_Element  |
| LX             | (2400 Service Line)<br>BusinessRules. Lookahead. GetInfo: Current_LoopCounter 200B_2400LoopCounter<br>BusinessRules. Variable. GetInfo: Current_LoopCounter 200B_2400LoopCounter   |
| DTP03          | (2400 Date Time Period)<br>BusinessRules. Lookahead. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element  |
| NM1            | (2420A Rendering Provider Name)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin Next_Row 0 "2"<br>BusinessRules. Lookahead. SetArrayFromVar: WSin Current_Row 1 200B_2400LoopCounter  |
| NM109          | (2420A Identification Code)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin Current_Row 2 Current_Element<br>BusinessRules. Array. SearchVarsInArray: WSout "2" 200B_2400LoopCounter Current_Element "" "" "0" (BusinessRules.Utilities DisplayErrorByNumber 32163) |
| REF02          | (2420A Reference Number)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin Current_Row 3 Current_Element  |
| DTP03          | (2430 Date Time Period)<br>BusinessRules. Lookahead. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element "D8"   |
| NM104          | (2010CA Name First)<br>BusinessRules. Variable. SetVar: IndividualFirstName Current_Element  |

| <b>Segment</b> | <b>Rules</b>  |
|----------------|---|
| DMG02          | (2010CA Date Time Period)<br>BusinessRules. Variable. SetVar: 2010CADMG02 Current_Element   |
| DMG03          | (2010CA Gender Code)<br>BusinessRules. Array. SearchVarsInArray: WSout "1" "" IndividualFirstName "" Current_Element<br>2010CADMG02 (BusinessRules.Utilities DisplayErrorByNumber 32153)  |
| DTP03          | (2300 Date Time Period)<br>BusinessRules. Lookahead. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element   |
| NM1            | (2310B Rendering Provider Name)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 1 0 "2"  |
| NM109          | (2310B Identification Code)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 1 1 Current_Element<br>BusinessRules. Array. SearchVarsInArray: WSout "2" Current_Element "" "" "0" (BusinessRules.Utilities DisplayErrorByNumber 32162)                                   |
| REF02          | (2310B Reference Number)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin 1 2 Current_Element   |
| LX             | (2400 Service Line)<br>BusinessRules. Variable. GetInfo: Current_LoopCounter 200C_2400LoopCounter<br>BusinessRules. Lookahead. GetInfo: Current_LoopCounter 200C_2400LoopCounter  |
| DTP03          | (2400 Date Time Period)<br>BusinessRules. Lookahead. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element "D8"  |
| NM1            | (2420A Rendering Provider Name)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin Next_Row 0 "2"<br>BusinessRules. Lookahead. SetArrayFromVar: WSin Current_Row 1 200C_2400LoopCounter   |
| NM109          | (2420A Identification Code)<br>BusinessRules. Lookahead. SetArrayFromVar: WSin Current_Row 2 Current_Element<br>BusinessRules. Array. SearchVarsInArray: WSout "2" 200C_2400LoopCounter Current_Element "" "" "0"<br>(BusinessRules.Utilities DisplayErrorByNumber 32163) |
| REF02          | (2420A Reference Number)  |

BusinessRules. Lookahead. SetArrayFromVar: WSin Current\_Row 3 Current\_Element

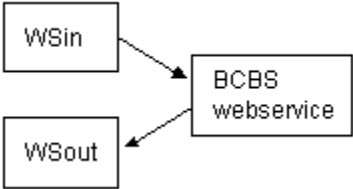
## Annotated Summary of Rules in Execution Order

This set of rules is an example only. These are the same rules as in the previous section, but they are in the order in which they will execute during validation.

Explanations are in *Italic*.

These rules are based on a 4010 837P called WEBSRV1\_837P (in the DemoData directory).

| Step  | Segment | Rules   |
|---|---------|---|
| ----- Process rules before Lookahead range -----                                      |         |   |
|   | ST      | (Table 1 Transaction Set Header)  |
| 1   |         | BusinessRules. Array. CreateArray: WSin<br><i>Create an array to serve as input to the web service.</i>   |
| 2   |         | BusinessRules. Array. CreateArray: WSout<br><i>Create an array to serve as output from the web service.</i>   |
| ----- Starting 2000A Lookahead range -----<br>Process Lookahead rules to end of range |         |   |
|   | HL      | (2000A Billing/Pay-to Provider Hierarchical Level)  |
| 3   |         | BusinessRules. <b>Lookahead</b> . ClearArray: WSin  |
| 4   |         | BusinessRules. <b>Lookahead</b> . ClearArray: WSout<br><i>Empty both arrays.</i>  |
|   | NM1     | (2010AA Billing Provider Name)  |
| 5   |         | BusinessRules. <b>Lookahead</b> . SetArrayFromVar: WSin 0 0 "2"<br><i>Load the literal value 2 into the first cell in array WSin (at index 0,0).</i><br><div> <div>2</div> <div></div> <div></div> <div></div> <div></div> <div></div> </div> |
|   | NM109   | (2010AA Identification Code)  |

| Step              | Segment           | Rules   |   |                   |                   |   |  |  |
|-------------------|-------------------|---|---|-------------------|-------------------|---|--|--|
| 6                 |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 0 1 Current_Element<br/> <i>Load the contents of the NM109 into the second cell in the first row of WSin (index 0,1).</i></p> <table border="1"> <tr> <td>2</td><td>(2010AA<br/>NM109)</td><td></td></tr> <tr> <td></td><td></td><td></td></tr> </table>  | 2 | (2010AA<br>NM109) |                   |   |  |  |
| 2                 | (2010AA<br>NM109) |   |   |                   |                   |   |  |  |
|                   |                   |   |   |                   |                   |   |  |  |
|                   | REF02             | (2010AA Reference Number)   |   |                   |                   |   |  |  |
| 7                 |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 0 2 Current_Element<br/> <i>Put the value from this REF02 into WSin's index 0,2.</i></p> <table border="1"> <tr> <td>2</td><td>(2010AA<br/>NM109)</td><td>(2010AA<br/>REF02)</td></tr> <tr> <td></td><td></td><td></td></tr> </table>   | 2 | (2010AA<br>NM109) | (2010AA<br>REF02) |   |  |  |
| 2                 | (2010AA<br>NM109) | (2010AA<br>REF02)   |   |                   |                   |   |  |  |
|                   |                   |   |   |                   |                   |   |  |  |
|                   | NM1               | (2010AB Pay-to Provider Name)   |   |                   |                   |   |  |  |
| 8<br>(if no data) |                   | <p>BusinessRules. <b>Lookahead</b>. RunNoData: (BusinessRules.DBServer InvokeWebService WEBSRV1 WSin "WSout" (BusinessRules.Utilities DisplayErrorByNumber 29001))<br/> <i>Lookahead to invoke the web server WEBSRV1 if this NM1 has no data. Send it the WSin array, receive WSout back.</i></p>  <pre> graph LR     WSin[WSin] --&gt; BCBS[BCBS webservice]     BCBS --&gt; WSout[WSout] </pre> |   |                   |                   |   |  |  |
| 9<br>(if no data) |                   | <p>BusinessRules. <b>Lookahead</b>. RunNoData: (BusinessRules.Array ExitLookahead)<br/> <i>Stop the Lookahead if this NM1 is not present in the data. This will skip Lookahead rules 8-12 below.</i></p>  |   |                   |                   |   |  |  |
| 8                 |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 1 0 "2"<br/> <i>Lookahead to populate the WSin array's 1,0 with the literal value 2.</i></p> <table border="1"> <tr> <td>2</td><td>(2010AA<br/>NM109)</td><td>(2010AA<br/>REF02)</td></tr> <tr> <td>2</td><td></td><td></td></tr> </table>  | 2 | (2010AA<br>NM109) | (2010AA<br>REF02) | 2 |  |  |
| 2                 | (2010AA<br>NM109) | (2010AA<br>REF02)   |   |                   |                   |   |  |  |
| 2                 |                   |   |   |                   |                   |   |  |  |

| Step | Segment           | Rules   |   |                   |                   |   |                   |                   |  |  |  |  |
|------|-------------------|---|---|-------------------|-------------------|---|-------------------|-------------------|--|--|--|--|
|      | NM109             | (2010AB Identification Code)  |   |                   |                   |   |                   |                   |  |  |  |  |
| 9    |                   | <div>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 1 1 Current_Element</div> <div>Put the value of this NM109 in WSin 1,1.</div> <table><tr><td>2</td><td>(2010AA<br/>NM109)</td><td>(2010AA<br/>REF02)</td></tr><tr><td>2</td><td>(2010AB<br/>NM109)</td><td></td></tr></table>   | 2 | (2010AA<br>NM109) | (2010AA<br>REF02) | 2 | (2010AB<br>NM109) |                   |  |  |  |  |
| 2    | (2010AA<br>NM109) | (2010AA<br>REF02)   |   |                   |                   |   |                   |                   |  |  |  |  |
| 2    | (2010AB<br>NM109) |   |   |                   |                   |   |                   |                   |  |  |  |  |
|      | REF02             | (2010AB Reference Number)   |   |                   |                   |   |                   |                   |  |  |  |  |
| 10   |                   | <div>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 1 2 Current_Element</div> <div>Put the value of this REF02 into WSin's 1,2:</div> <table><tr><td>2</td><td>(2010AA<br/>NM109)</td><td>(2010AA<br/>REF02)</td></tr><tr><td>2</td><td>(2010AB<br/>NM109)</td><td>(2010AB<br/>REF02)</td></tr></table>   | 2 | (2010AA<br>NM109) | (2010AA<br>REF02) | 2 | (2010AB<br>NM109) | (2010AB<br>REF02) |  |  |  |  |
| 2    | (2010AA<br>NM109) | (2010AA<br>REF02)   |   |                   |                   |   |                   |                   |  |  |  |  |
| 2    | (2010AB<br>NM109) | (2010AB<br>REF02)   |   |                   |                   |   |                   |                   |  |  |  |  |
| 11   |                   | <div>BusinessRules. <b>Lookahead</b>. InvokeWebService: WEBSRV1 WSin WSout</div> <div>(BusinessRules.Utilities DisplayErrorByNumber 29001)</div> <div>Lookahead to again invoke the web server WEBSRV1. Send it the WSin array, receive WSout back.</div>   |   |                   |                   |   |                   |                   |  |  |  |  |
| 12   |                   | <div>BusinessRules. <b>Lookahead</b>. ExitLookahead:</div> <div>Stop the 2000A Lookahead.</div>   |   |                   |                   |   |                   |                   |  |  |  |  |
|      | NM109             | (2010AA Identification Code)  |   |                   |                   |   |                   |                   |  |  |  |  |
| 13   |                   | <div>BusinessRules. Array. CheckVarFromArray: WSout 0 4 "0" (BusinessRules.Utilities DisplayErrorByNumber 32160)</div> <div>We have completed the Lookahead rules for the first Lookahead range, which ends with the start of the second Lookahead range at 2000B. We return to the top of the 2000A and start doing its non-Lookahead rules.</div> <div>Check array WSout, which has been returned from the web service. If it does not contain 0 in index 0,4, display error 32160.</div> <table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table> |   |                   |                   |   |                   |                   |  |  |  |  |
|      |                   |   |   |                   |                   |   |                   |                   |  |  |  |  |
|      |                   |   |   |                   |                   |   |                   |                   |  |  |  |  |

| Step   | Segment           | Rules   |   |                   |                         |  |  |  |  |  |  |  |
|--|-------------------|---|---|-------------------|-------------------------|--|--|--|--|--|--|--|
| 14   |                   | <div>BusinessRules. Array. CheckVarFromArray: WSout 1 4 "0" (BusinessRules.Utilities DisplayErrorByNumber 32161)</div> <div>Check array WSout, which has been returned from the web service. If it does not contain 0 in index 1,4, display error 32161.</div> <div><table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></div> |   |                   |                         |  |  |  |  |  |  |  |
|  |                   |   |   |                   |                         |  |  |  |  |  |  |  |
|  |                   |   |   |                   |                         |  |  |  |  |  |  |  |
| <div>----- Starting 2000B Lookahead range -----</div> <div>Automatically ends 2000A Lookahead range</div> <div>Process Lookahead rules to end of 2000B range</div> |                   |   |   |                   |                         |  |  |  |  |  |  |  |
|  | HL                | (2000B Subscriber Hierarchical Level)   |   |                   |                         |  |  |  |  |  |  |  |
| 15   |                   | <div>BusinessRules. Lookahead. ClearArray: WSin</div>   |   |                   |                         |  |  |  |  |  |  |  |
| 16   |                   | <div>BusinessRules. Lookahead. ClearArray: WSout</div> <div>Lookahead to clear both arrays at the beginning of the 2000B.</div>   |   |                   |                         |  |  |  |  |  |  |  |
|  | NM109             | (2010BA Identification Code)  |   |                   |                         |  |  |  |  |  |  |  |
| 17   |                   | <div>BusinessRules. Lookahead. SetArrayFromVar: WSin 0 1 Current_Element</div> <div>Put contents of this NM109 into WSin index 0,1:</div> <div><table><tr><td></td><td>(2010BA<br/>NM109)</td><td></td></tr><tr><td></td><td></td><td></td></tr></table></div>  |   | (2010BA<br>NM109) |                         |  |  |  |  |  |  |  |
|  | (2010BA<br>NM109) |   |   |                   |                         |  |  |  |  |  |  |  |
|  |                   |   |   |                   |                         |  |  |  |  |  |  |  |
| 18   |                   | <div>BusinessRules. Lookahead. SetArrayFromVar: WSin 0 0 "1"</div> <div>Put literal value 1 in first cell of WSin:</div> <div><table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td></td></tr><tr><td></td><td></td><td></td></tr></table></div>  | 1 | (2010BA<br>NM109) |                         |  |  |  |  |  |  |  |
| 1  | (2010BA<br>NM109) |   |   |                   |                         |  |  |  |  |  |  |  |
|  |                   |   |   |                   |                         |  |  |  |  |  |  |  |
|  | DTP03             | (2300 Date Time Period)   |   |                   |                         |  |  |  |  |  |  |  |
| 19   |                   | <div>BusinessRules. Lookahead. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element "D8"</div> <div>Puts the date from this DTP03 into WSin index 0,2.</div> <div><table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(date<br/>from<br/>DTP03)</td></tr><tr><td></td><td></td><td></td></tr></table></div>   | 1 | (2010BA<br>NM109) | (date<br>from<br>DTP03) |  |  |  |  |  |  |  |
| 1  | (2010BA<br>NM109) | (date<br>from<br>DTP03)   |   |                   |                         |  |  |  |  |  |  |  |
|  |                   |   |   |                   |                         |  |  |  |  |  |  |  |

| Step | Segment        | Rules  |   |                |                   |   |               |               |
|------|----------------|--|---|----------------|-------------------|---|---------------|---------------|
|      | NM1            | (2310B Rendering Provider Name)  |   |                |                   |   |               |               |
| 20   |                | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 1 0 "2"</p> <p><i>Put the literal value 2 into WSin 1,0.</i></p> <table> <tr> <td>1</td><td>(2010BA NM109)</td><td>(date from DTP03)</td></tr> <tr> <td>2</td><td></td><td></td></tr> </table>   | 1 | (2010BA NM109) | (date from DTP03) | 2 |               |               |
| 1    | (2010BA NM109) | (date from DTP03)  |   |                |                   |   |               |               |
| 2    |                |  |   |                |                   |   |               |               |
|      | NM109          | (2310B Identification Code)  |   |                |                   |   |               |               |
| 21   |                | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 1 1 Current_Element</p> <p><i>Put the value of this NM109 into WSin 1,1.</i></p> <table> <tr> <td>1</td><td>(2010BA NM109)</td><td>(date from DTP03)</td></tr> <tr> <td>2</td><td>(2310B NM109)</td><td></td></tr> </table>              | 1 | (2010BA NM109) | (date from DTP03) | 2 | (2310B NM109) |               |
| 1    | (2010BA NM109) | (date from DTP03)  |   |                |                   |   |               |               |
| 2    | (2310B NM109)  |  |   |                |                   |   |               |               |
|      | REF02          | (2310B Reference Number)   |   |                |                   |   |               |               |
| 22   |                | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 1 2 Current_Element</p> <p><i>Put the value of this REF02 into WSin 1,2.</i></p> <table> <tr> <td>1</td><td>(2010BA NM109)</td><td>(date from DTP03)</td></tr> <tr> <td>2</td><td>(2310B NM109)</td><td>(2310B REF02)</td></tr> </table> | 1 | (2010BA NM109) | (date from DTP03) | 2 | (2310B NM109) | (2310B REF02) |
| 1    | (2010BA NM109) | (date from DTP03)  |   |                |                   |   |               |               |
| 2    | (2310B NM109)  | (2310B REF02)  |   |                |                   |   |               |               |
|      | LX             | (2400 Service Line)  |   |                |                   |   |               |               |
| 23   |                | <p>BusinessRules. <b>Lookahead</b>. GetInfo: Current_LoopCounter 200B_2400LoopCounter</p> <p><i>Put the iteration of the 2400 loop into variable 200B_2400LoopCounter.</i></p>   |   |                |                   |   |               |               |

| Step | Segment           | Rules   |   |                   |                     |   |                  |                  |   |   |                  |
|------|-------------------|---|---|-------------------|---------------------|---|------------------|------------------|---|---|------------------|
|      | DTP03             | (2400 Date Time Period)   |   |                   |                     |   |                  |                  |   |   |                  |
| 24   |                   | <p>BusinessRules. <b>Lookahead</b>. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element<br/> <i>Put the date from the current element into WSin 0,2 if it is earlier than the date that is currently there.</i></p> <table> <tr> <td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td></tr> <tr> <td>2</td><td>(2310B<br/>NM109)</td><td>(2310B<br/>REF02)</td></tr> </table>                               | 1 | (2010BA<br>NM109) | (earliest<br>DTP03) | 2 | (2310B<br>NM109) | (2310B<br>REF02) |   |   |                  |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)   |   |                   |                     |   |                  |                  |   |   |                  |
| 2    | (2310B<br>NM109)  | (2310B<br>REF02)  |   |                   |                     |   |                  |                  |   |   |                  |
|      | NM1               | (2420A Rendering Provider Name)   |   |                   |                     |   |                  |                  |   |   |                  |
| 25   |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin Next_Row 0 "2"<br/> <i>Put the literal value 2 into the first cell of the next row</i></p> <table> <tr> <td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td></tr> <tr> <td>2</td><td>(2310B<br/>NM109)</td><td>(2310B<br/>REF02)</td></tr> <tr> <td>2</td><td></td><td></td></tr> </table>  | 1 | (2010BA<br>NM109) | (earliest<br>DTP03) | 2 | (2310B<br>NM109) | (2310B<br>REF02) | 2 |   |                  |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)   |   |                   |                     |   |                  |                  |   |   |                  |
| 2    | (2310B<br>NM109)  | (2310B<br>REF02)  |   |                   |                     |   |                  |                  |   |   |                  |
| 2    |                   |   |   |                   |                     |   |                  |                  |   |   |                  |
| 26   |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin Current_Row 1<br/> 200B_2400LoopCounter<br/> <i>Put the contents of the 2400 loop counter (see step 23) into the current row's cell 1.</i></p> <table> <tr> <td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td></tr> <tr> <td>2</td><td>(2310B<br/>NM109)</td><td>(2010B<br/>REF02)</td></tr> <tr> <td>2</td><td>1</td><td></td></tr> </table> | 1 | (2010BA<br>NM109) | (earliest<br>DTP03) | 2 | (2310B<br>NM109) | (2010B<br>REF02) | 2 | 1 |                  |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)   |   |                   |                     |   |                  |                  |   |   |                  |
| 2    | (2310B<br>NM109)  | (2010B<br>REF02)  |   |                   |                     |   |                  |                  |   |   |                  |
| 2    | 1                 |   |   |                   |                     |   |                  |                  |   |   |                  |
|      | NM109             | (2420A Identification Code)   |   |                   |                     |   |                  |                  |   |   |                  |
| 27   |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin Current_Row 2 Current_Element<br/> <i>Put the contents of this NM109 into the current row's cell 2.</i></p> <table> <tr> <td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td></tr> <tr> <td>2</td><td>(2310B<br/>NM109)</td><td>(2010B<br/>REF02)</td></tr> <tr> <td>2</td><td>1</td><td>(2420A<br/>NM109)</td></tr> </table>                   | 1 | (2010BA<br>NM109) | (earliest<br>DTP03) | 2 | (2310B<br>NM109) | (2010B<br>REF02) | 2 | 1 | (2420A<br>NM109) |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)   |   |                   |                     |   |                  |                  |   |   |                  |
| 2    | (2310B<br>NM109)  | (2010B<br>REF02)  |   |                   |                     |   |                  |                  |   |   |                  |
| 2    | 1                 | (2420A<br>NM109)  |   |                   |                     |   |                  |                  |   |   |                  |



| Step                    | Segment           | Rules   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
|-------------------------|-------------------|---|------------------|-------------------|---------------------|--|---|------------------|------------------|--|---|---|------------------|------------------|
|                         | REF02             | (2420A Reference Number)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 28                      |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin Current_Row 3 Current_Element</p> <p>Put the contents of this REF02 into the current row's cell 3.</p> <table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B<br/>NM109)</td><td>(2010B<br/>REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A<br/>NM109)</td><td>(2420A<br/>REF02)</td></tr></table>                                     | 1                | (2010BA<br>NM109) | (earliest<br>DTP03) |  | 2 | (2310B<br>NM109) | (2010B<br>REF02) |  | 2 | 1 | (2420A<br>NM109) | (2420A<br>REF02) |
| 1                       | (2010BA<br>NM109) | (earliest<br>DTP03)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2                       | (2310B<br>NM109)  | (2010B<br>REF02)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2                       | 1                 | (2420A<br>NM109)  | (2420A<br>REF02) |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
|                         | DTP03             | (2430 Date Time Period)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 29                      |                   | <p>BusinessRules. <b>Lookahead</b>. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element "D8"</p> <p>Put this date into WSin's 0,2 if it is earlier than the current contents. Format is YYYYMMDD.</p> <table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B<br/>NM109)</td><td>(2010B<br/>REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A<br/>NM109)</td><td>(2420A<br/>REF02)</td></tr></table> | 1                | (2010BA<br>NM109) | (earliest<br>DTP03) |  | 2 | (2310B<br>NM109) | (2010B<br>REF02) |  | 2 | 1 | (2420A<br>NM109) | (2420A<br>REF02) |
| 1                       | (2010BA<br>NM109) | (earliest<br>DTP03)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2                       | (2310B<br>NM109)  | (2010B<br>REF02)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2                       | 1                 | (2420A<br>NM109)  | (2420A<br>REF02) |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| Start of Dependent 2300 |                   |   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
|                         | DTP03             | (2300 Date Time Period)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 30                      |                   | <p>BusinessRules. <b>Lookahead</b>. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element</p> <p>Put this date into WSin's 0,2 if it is earlier than the current contents.</p> <table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B<br/>NM109)</td><td>(2010B<br/>REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A<br/>NM109)</td><td>(2420A<br/>REF02)</td></tr></table>                          | 1                | (2010BA<br>NM109) | (earliest<br>DTP03) |  | 2 | (2310B<br>NM109) | (2010B<br>REF02) |  | 2 | 1 | (2420A<br>NM109) | (2420A<br>REF02) |
| 1                       | (2010BA<br>NM109) | (earliest<br>DTP03)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2                       | (2310B<br>NM109)  | (2010B<br>REF02)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2                       | 1                 | (2420A<br>NM109)  | (2420A<br>REF02) |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
|                         | NM1               | (2310B Rendering Provider Name)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |

| Step | Segment           | Rules  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
|------|-------------------|--|------------------|-------------------|---------------------|--|---|------------------|------------------|--|---|---|------------------|------------------|
| 31   |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 1 0 "2"</p> <p>Put 2 in this cell:</p> <table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B<br/>NM109)</td><td>(2010B<br/>REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A<br/>NM109)</td><td>(2420A<br/>REF02)</td></tr></table>                                      | 1                | (2010BA<br>NM109) | (earliest<br>DTP03) |  | 2 | (2310B<br>NM109) | (2010B<br>REF02) |  | 2 | 1 | (2420A<br>NM109) | (2420A<br>REF02) |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2    | (2310B<br>NM109)  | (2010B<br>REF02)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2    | 1                 | (2420A<br>NM109)   | (2420A<br>REF02) |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
|      | NM109             | (2310B Identification Code)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 32   |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 1 1 Current_Element</p> <p>Put the contents of this NM109 in this cell:</p> <table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B<br/>NM109)</td><td>(2010B<br/>REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A<br/>NM109)</td><td>(2420A<br/>REF02)</td></tr></table> | 1                | (2010BA<br>NM109) | (earliest<br>DTP03) |  | 2 | (2310B<br>NM109) | (2010B<br>REF02) |  | 2 | 1 | (2420A<br>NM109) | (2420A<br>REF02) |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2    | (2310B<br>NM109)  | (2010B<br>REF02)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2    | 1                 | (2420A<br>NM109)   | (2420A<br>REF02) |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
|      | REF02             | (2310B Reference Number)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 33   |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin 1 2 Current_Element</p> <p>Put the contents of this REF02 in this cell:</p> <table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B<br/>NM109)</td><td>(2310B<br/>REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A<br/>NM109)</td><td>(2420A<br/>REF02)</td></tr></table> | 1                | (2010BA<br>NM109) | (earliest<br>DTP03) |  | 2 | (2310B<br>NM109) | (2310B<br>REF02) |  | 2 | 1 | (2420A<br>NM109) | (2420A<br>REF02) |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2    | (2310B<br>NM109)  | (2310B<br>REF02)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 2    | 1                 | (2420A<br>NM109)   | (2420A<br>REF02) |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
|      | LX                | (2400 Service Line)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |
| 34   |                   | <p>BusinessRules. <b>Lookahead</b>. GetInfo: Current_LoopCounter 200C_2400LoopCounter</p> <p>Put the current loop count into the variable 200C_2400LoopCounter.</p>  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |

| Step | Segment           | Rules   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
|------|-------------------|---|------------------|-------------------|---------------------|--|---|------------------|------------------|--|---|---|------------------|------------------|---|-----------------|--|--|
|      | DTP03             | (2400 Date Time Period)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 35   |                   | <p>BusinessRules. <b>Lookahead</b>. UpdateArrayFromDate: WSin 0 2 "LT" Current_Element "D8"</p> <p>Put this date in 0,2 if it is earlier than the one that is currently there.</p> <table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B<br/>NM109)</td><td>(2310B<br/>REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A<br/>NM109)</td><td>(2420A<br/>REF02)</td></tr></table>   | 1                | (2010BA<br>NM109) | (earliest<br>DTP03) |  | 2 | (2310B<br>NM109) | (2310B<br>REF02) |  | 2 | 1 | (2420A<br>NM109) | (2420A<br>REF02) |   |                 |  |  |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 2    | (2310B<br>NM109)  | (2310B<br>REF02)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 2    | 1                 | (2420A<br>NM109)  | (2420A<br>REF02) |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
|      | NM1               | (2420A Rendering Provider Name)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 36   |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin Next_Row 0 "2"</p> <p>Put the literal value 2 in the next row's first cell.</p> <table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B<br/>NM109)</td><td>(2310B<br/>REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A<br/>NM109)</td><td>(2420A<br/>REF02)</td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>  | 1                | (2010BA<br>NM109) | (earliest<br>DTP03) |  | 2 | (2310B<br>NM109) | (2310B<br>REF02) |  | 2 | 1 | (2420A<br>NM109) | (2420A<br>REF02) | 2 |                 |  |  |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 2    | (2310B<br>NM109)  | (2310B<br>REF02)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 2    | 1                 | (2420A<br>NM109)  | (2420A<br>REF02) |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 2    |                   |   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 37   |                   | <p>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin Current_Row 1 200C_2400LoopCounter</p> <p>Put the value from the loop counter into the current row's cell 1.</p> <table><tr><td>1</td><td>(2010BA<br/>NM109)</td><td>(earliest<br/>DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B<br/>NM109)</td><td>(2310B<br/>REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A<br/>NM109)</td><td>(2420A<br/>REF02)</td></tr><tr><td>2</td><td>(loop<br/>count)</td><td></td><td></td></tr></table> | 1                | (2010BA<br>NM109) | (earliest<br>DTP03) |  | 2 | (2310B<br>NM109) | (2310B<br>REF02) |  | 2 | 1 | (2420A<br>NM109) | (2420A<br>REF02) | 2 | (loop<br>count) |  |  |
| 1    | (2010BA<br>NM109) | (earliest<br>DTP03)   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 2    | (2310B<br>NM109)  | (2310B<br>REF02)  |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 2    | 1                 | (2420A<br>NM109)  | (2420A<br>REF02) |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |
| 2    | (loop<br>count)   |   |                  |                   |                     |  |   |                  |                  |  |   |   |                  |                  |   |                 |  |  |

| Step   | Segment        | Rules  |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
|--|----------------|--|---------------|----------------|------------------|--|---|---------------|---------------|--|---|---|---------------|---------------|---|--------------|---------------|---------------|
|  | NM109          | (2420A Identification Code)  |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 38   |                | <div>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin Current_Row 2 Current_Element</div> <div>Put the value of this NM109 into the current row's cell 2.</div> <table><tr><td>1</td><td>(2010BA NM109)</td><td>(earliest DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B NM109)</td><td>(2310B REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A NM109)</td><td>(2420A REF02)</td></tr><tr><td>2</td><td>(loop count)</td><td>(2420A NM109)</td><td></td></tr></table>              | 1             | (2010BA NM109) | (earliest DTP03) |  | 2 | (2310B NM109) | (2310B REF02) |  | 2 | 1 | (2420A NM109) | (2420A REF02) | 2 | (loop count) | (2420A NM109) |               |
| 1  | (2010BA NM109) | (earliest DTP03)   |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 2  | (2310B NM109)  | (2310B REF02)  |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 2  | 1              | (2420A NM109)  | (2420A REF02) |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 2  | (loop count)   | (2420A NM109)  |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
|  | REF02          | (2420A Reference Number)   |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 39   |                | <div>BusinessRules. <b>Lookahead</b>. SetArrayFromVar: WSin Current_Row 3 Current_Element</div> <div>Put the value of this REF02 into the current row's cell 3.</div> <table><tr><td>1</td><td>(2010BA NM109)</td><td>(earliest DTP03)</td><td></td></tr><tr><td>2</td><td>(2310B NM109)</td><td>(2310B REF02)</td><td></td></tr><tr><td>2</td><td>1</td><td>(2420A NM109)</td><td>(2420A REF02)</td></tr><tr><td>2</td><td>(loop count)</td><td>(2420A NM109)</td><td>(2420A REF02)</td></tr></table> | 1             | (2010BA NM109) | (earliest DTP03) |  | 2 | (2310B NM109) | (2310B REF02) |  | 2 | 1 | (2420A NM109) | (2420A REF02) | 2 | (loop count) | (2420A NM109) | (2420A REF02) |
| 1  | (2010BA NM109) | (earliest DTP03)   |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 2  | (2310B NM109)  | (2310B REF02)  |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 2  | 1              | (2420A NM109)  | (2420A REF02) |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 2  | (loop count)   | (2420A NM109)  | (2420A REF02) |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| <div>----- End of 2000B Lookahead range -----</div> <div>Start 2000B end-of-loop Lookahead rules</div> |                |  |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
|  | ST             | (Table 1 Transaction Set Header)   |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 40   |                | <div>BusinessRules. Exits. SetLoopPostInstanceExit: 2000B BusinessRules.<b>Lookahead</b></div> <div>InvokeWebService WEBSRV1 WSin "WSout" (BusinessRules.Utilities</div> <div>DisplayErrorByNumber 29001)</div> <div>Send the WSin array to the WEBSRV1 web service, return information in the WSout array, and</div> <div>display error 29001 if the web service is not available.</div> <div>Exit rules execute in reverse order. The bottom one executes first.</div>                               |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |
| 41   |                | <div>BusinessRules. Exits. SetLoopPostInstanceExit: 2000B BusinessRules.<b>Lookahead</b></div> <div>ExitLookahead</div> <div>End the Lookahead range after each 2000B loop.</div>  |               |                |                  |  |   |               |               |  |   |   |               |               |   |              |               |               |

| Step   | Segment | Rules   |  |  |  |  |  |  |  |  |  |  |
|--|---------|---|--|--|--|--|--|--|--|--|--|--|
| ----- End of 2000B end-of-loop Lookahead rules -----<br>Starting 2000B non-Lookahead rules |         |   |  |  |  |  |  |  |  |  |  |  |
|  | NM103   | (2010BA Name Last)  |  |  |  |  |  |  |  |  |  |  |
| 42   |         | <div>BusinessRules. Array. CheckVarFromArray: WSout 0 3 Current_Element<br/>(BusinessRules.Utilities DisplayErrorByNumber 32152)</div> <div>If 0,3 in the WSout array returned from the web service does not match the contents this NM103, display error 32152.</div> <div><table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></div> |  |  |  |  |  |  |  |  |  |  |
|  |         |   |  |  |  |  |  |  |  |  |  |  |
|  |         |   |  |  |  |  |  |  |  |  |  |  |
|  | NM104   | (2010BA Name First)   |  |  |  |  |  |  |  |  |  |  |
| 43   |         | <div>BusinessRules. Variable. SetVar: SubFirstName Current_Element</div> <div>Put the value from the current element into variable SubFirstName.</div>  |  |  |  |  |  |  |  |  |  |  |
|  | NM109   | (2010BA Identification Code)  |  |  |  |  |  |  |  |  |  |  |
| 44   |         | <div>BusinessRules. Array. GetVarFromArray: WSout 0 1 "SubscriberID"</div> <div>Put the value from the WSout array 0,1 into variable SubscriberID.</div> <div><table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></div>   |  |  |  |  |  |  |  |  |  |  |
|  |         |   |  |  |  |  |  |  |  |  |  |  |
|  |         |   |  |  |  |  |  |  |  |  |  |  |
| 45   |         | <div>BusinessRules. Variable. CompareNString: SubscriberID "NE" Current_Element<br/>(1;1;3;) (BusinessRules.Utilities DisplayErrorByNumber 32150)</div> <div>If the contents of SubscriberID and the contents of this NM109 do not match, display error 32150.</div>  |  |  |  |  |  |  |  |  |  |  |
| 46   |         | <div>BusinessRules. Variable. CompareNString: SubscriberID "NE" Current_Element<br/>(4;4;9;) (BusinessRules.Utilities DisplayErrorByNumber 32151)</div> <div>Compare 9 characters of SubscriberID and the value in the current element, starting at position 4 in each. If they do not match, display error 32151.</div>  |  |  |  |  |  |  |  |  |  |  |
|  | DMG02   | (2010BA Date Time Period)   |  |  |  |  |  |  |  |  |  |  |
| 47   |         | <div>BusinessRules. Variable. SetVar: DMG02 Current_Element</div> <div>Put contents of this DMG02 into variable DMG02.</div>  |  |  |  |  |  |  |  |  |  |  |

| Step | Segment          | Rules   |                |                   |                  |                |                   |
|------|------------------|---|----------------|-------------------|------------------|----------------|-------------------|
|      | DMG03            | (2010BA Gender Code)  |                |                   |                  |                |                   |
| 48   |                  | <div>BusinessRules. Array. SearchVarsInArray: WSout "1" "" "" SubFirstName<br/>Current_Element DMG02 (BusinessRules.Utilities DisplayErrorByNumber 32153)<br/>Display error 32153 if array WSout does not contain a row that starts with these values:</div> <table><tr><td>1</td><td></td><td></td><td>(SubFirstName)</td><td>(2010BA<br/>DMG02)</td></tr></table>         | 1              |                   |                  | (SubFirstName) | (2010BA<br>DMG02) |
| 1    |                  |   | (SubFirstName) | (2010BA<br>DMG02) |                  |                |                   |
|      | NM109            | (2310B Identification Code)   |                |                   |                  |                |                   |
| 49   |                  | <div>BusinessRules. Array. SearchVarsInArray: WSout "2" Current_Element "" "" "0"<br/>(BusinessRules.Utilities DisplayErrorByNumber 32162)<br/>Display error 32162 if array WSout does not contain a row that starts with these values:</div> <table><tr><td>2</td><td>(2310B<br/>NM109)</td><td></td><td></td><td>0</td></tr></table>                                      | 2              | (2310B<br>NM109)  |                  |                | 0                 |
| 2    | (2310B<br>NM109) |   |                | 0                 |                  |                |                   |
|      | LX               | (2400 Service Line)   |                |                   |                  |                |                   |
| 50   |                  | <div>BusinessRules. Variable. GetInfo: Current_LoopCounter 200B_2400LoopCounter<br/>Put the current loop count into variable 200B_2400LoopCounter.</div>  |                |                   |                  |                |                   |
|      | NM109            | (2420A Identification Code)   |                |                   |                  |                |                   |
| 51   |                  | <div>BusinessRules. Array. SearchVarsInArray: WSout "2" 200B_2400LoopCounter<br/>Current_Element "" "" "0" (BusinessRules.Utilities DisplayErrorByNumber 32163)<br/>Display error 32163 if array WSout does not contain a row that starts with these values:</div> <table><tr><td>2</td><td>(loop<br/>count)</td><td>(2420A<br/>NM109)</td><td></td><td>0</td></tr></table> | 2              | (loop<br>count)   | (2420A<br>NM109) |                | 0                 |
| 2    | (loop<br>count)  | (2420A<br>NM109)  |                | 0                 |                  |                |                   |
|      | NM104            | (2010CA Name First)   |                |                   |                  |                |                   |
| 52   |                  | <div>BusinessRules. Variable. SetVar: IndividualFirstName Current_Element<br/>Put the value from this NM104 into variable IndividualFirstName.</div>  |                |                   |                  |                |                   |
|      | DMG02            | (2010CA Date Time Period)   |                |                   |                  |                |                   |
| 53   |                  | <div>BusinessRules. Variable. SetVar: 2010CADMG02 Current_Element<br/>Put the value in this DMG02 into variable 2010CADMG02.</div>  |                |                   |                  |                |                   |
|      | DMG03            | (2010CA Gender Code)  |                |                   |                  |                |                   |

| Step | Segment          | Rules   |                   |                   |                   |                   |                   |
|------|------------------|---|-------------------|-------------------|-------------------|-------------------|-------------------|
| 54   |                  | <div>BusinessRules. Array. SearchVarsInArray: WSout "1" "" IndividualFirstName ""<br/>Current_Element 2010CADMG02 (BusinessRules.Utilities DisplayErrorByNumber 32153)</div> <div>Display error 32153 if array WSout does not contain a row that starts with these values:</div> <table><tr><td>1</td><td></td><td>(2010CA<br/>NM104)</td><td>(2010CA<br/>DMG03)</td><td>(2010CA<br/>DMG02)</td></tr></table> | 1                 |                   | (2010CA<br>NM104) | (2010CA<br>DMG03) | (2010CA<br>DMG02) |
| 1    |                  | (2010CA<br>NM104)   | (2010CA<br>DMG03) | (2010CA<br>DMG02) |                   |                   |                   |
|      | NM109            | (2310B Identification Code)   |                   |                   |                   |                   |                   |
| 55   |                  | <div>BusinessRules. Array. SearchVarsInArray: WSout "2" Current_Element "" "" "0"<br/>(BusinessRules.Utilities DisplayErrorByNumber 32162)</div> <div>Display error 32162 if array WSout does not contain a row that starts with these values:</div> <table><tr><td>2</td><td>(2310B<br/>NM109)</td><td></td><td></td><td>0</td></tr></table>   | 2                 | (2310B<br>NM109)  |                   |                   | 0                 |
| 2    | (2310B<br>NM109) |   |                   | 0                 |                   |                   |                   |
|      | LX               | (2400 Service Line)   |                   |                   |                   |                   |                   |
| 56   |                  | <div>BusinessRules. Variable. GetInfo: Current_LoopCounter 200C_2400LoopCounter</div> <div>Increment the loop counter.</div>  |                   |                   |                   |                   |                   |
|      | NM109            | (2420A Identification Code)   |                   |                   |                   |                   |                   |
| 57   |                  | <div>BusinessRules. Array. SearchVarsInArray: WSout "2" 200C_2400LoopCounter<br/>Current_Element "" "" "0" (BusinessRules.Utilities DisplayErrorByNumber 32163)</div> <div>Display error 32163 if array WSout does not contain a row that starts with these values:</div> <table><tr><td>2</td><td>(loop<br/>count)</td><td>(2420A<br/>NM109)</td><td>0</td></tr></table>                                     | 2                 | (loop<br>count)   | (2420A<br>NM109)  | 0                 |                   |
| 2    | (loop<br>count)  | (2420A<br>NM109)  | 0                 |                   |                   |                   |                   |

# Appendix K: Building Business Rules

---

## Overview

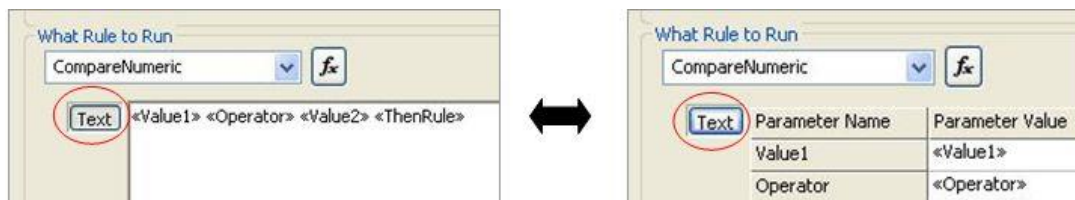
EDISIM Standards Editor provides two methods of entering text when building a business rule.

- Text Entry - in which rule text can be entered manually. Optionally, help text can be provided, displaying the expected format of the selected rule.
- Prompted Entry - which provides a “fill-in-the-blank” style rule builder. This format also allows the use of the Rule Selector to embed other rules within the rule being built.

## Text Button

For new rules, the text entry method with help is displayed by default.

The Text button allows you to toggle between text entry and prompted entry methods.





# Entry Examples

This section illustrates the various rule entry methods, each building the same business rule.

Our business rule checks the total number in a variable called “PLBAdjustmentAmt”. IF the number exceeds 10000, THEN error message 32214 displays.

## Text Entry

Text Entry allows you to enter text with no prompting or reference cues for the format of the rule. This method of rule entry is for experienced users who know the rule formats.

The screenshot shows a dialog box titled "What Rule to Run". It has a dropdown menu set to "SetArrayFromVar" and a button with a function symbol (fx). To the right is a checkbox labeled "Look-Ahead Rule" which is unchecked. Below this is a "Text" button and a text input field containing the string: "Array1Back" "0" "5" "0".

## Text Entry with Help

Text Entry with help is the same as Text Entry but adds reference cues for the proper formatting of the rule. Select each cue and overwrite with the desired parameter. This method of entry is useful for intermediate-level users, who are familiar with rule building but haven't memorized the required parameters.

The screenshot shows the same "What Rule to Run" dialog box. The dropdown menu is still "SetArrayFromVar" and the "Look-Ahead Rule" checkbox is unchecked. The "Text" button is present. The text input field contains the template: <arrayName> <rowIndex> <columnIndex> <resultVar>.

In this case:

<arrayName> is replaced with "Array1Back"

<rowIndex> is replaced with "0"

<columnIndex> is replaced with "5"

<resultVar> is replaced with "0"

## Grid Entry Method

The grid allows you to build a rule in a fill-in-the-blank manner. The structure of the rule is transferred to the input area and you are prompted to enter the parameters the rule requires to run. Parameter notes are provided for reference. This format also allows the use of the Rule Selector to embed other rules within the one being built.

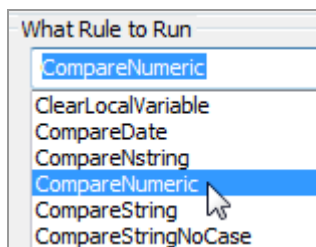
This method is the default for rules selected from the Rule Selector Dialog and from the common rules listed on the Select Rule drop box and is useful for all users, especially those learning rule building.

### Example: Prompted Entry Method

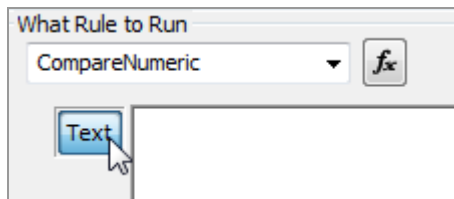
---

To enter a business rule using the prompted entry method:

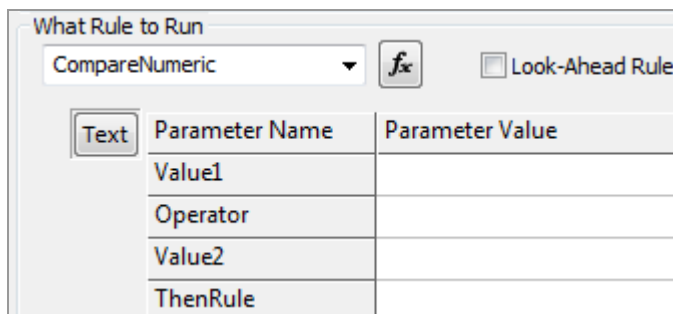
1. Access the Condition and Rule Definition dialog.
2. Select the desired rule from the drop box. In this example, we will choose CompareNumeric.



If a grid doesn't appear, click **Text**:





Parameter fields appear for the selected rule:


A screenshot of the "What Rule to Run" dialog box. The "CompareNumeric" rule is selected in the dropdown menu. To the right of the dropdown is a button with the symbol "fx". To the right of the "fx" button is a checkbox labeled "Look-Ahead Rule". Below the dropdown and "fx" button is a "Text" button. To the right of the "Text" button is a table with two columns: "Parameter Name" and "Parameter Value". The table has four rows: "Value1", "Operator", "Value2", and "ThenRule".

| Parameter Name | Parameter Value |
|----------------|-----------------|
| Value1         |                 |
| Operator       |                 |
| Value2         |                 |
| ThenRule       |                 |

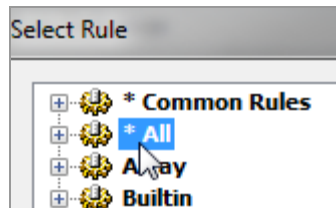
- Click in the Parameter Value area for the first parameter. This causes the Parameter Notes area on the right side to show helpful information:

| Parameter Name | Parameter Value | LA  | Parameter Notes   |
|----------------|-----------------|---|---|
| Value1         |                 |   | First value to compare. May be a variable, system variable, or a constant in double-quotes. This parameter must be a variable name or numeric constant. |
| Operator       |                 |   |   |
| Value2         |                 |   |   |
| ThenRule       |                 |  |   |

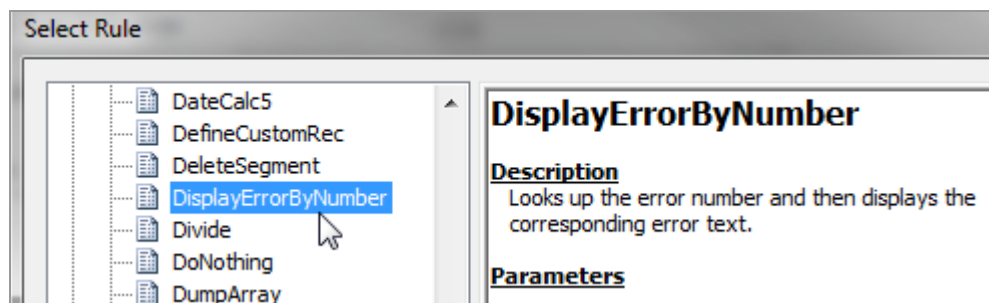
- Enter values for all parameters.
- If the rule is an “IF” test, requiring a sub-rule, use the small  button on the right side of the ThenRule entry box to access the Select Rule dialog.

| Parameter Name | Parameter Value  |   |
|----------------|------------------|---|
| Value1         | PLBAdjustmentAmt |   |
| Operator       | GT               |   |
| Value2         | 10000            |   |
| ThenRule       |                  |  |

In the Select Rule Dialog, click the rule you want to run when the “if” conditions are met for our rule followed by Select. In our example, we Click \*All:




... and then choose DisplayErrorByNumber:



6. This information is transferred to our rule and causes additional parameters to be displayed:

What Rule to Run


CompareNumeric  ☐ Look-Ahead Rule

**Text**


| Parameter Name | Parameter Value      |
|----------------|----------------------|
| Value1         | PLBAdjustmentAmt     |
| Operator       | GT                   |
| Value2         | 10000                |
| ThenRule       | DisplayErrorByNumber |
| > ErrorNumber  |                      |
| > Severity     |                      |
| > MessageText  |                      |

Finish filling out parameters.

What Rule to Run


CompareNumeric  ☐ Look-Ahead Rule

**Text**

| Parameter Name | Parameter Value      |   | LA                       |
|----------------|----------------------|---|--------------------------|
| Value1         | PLBAdjustmentAmt     |   |                          |
| Operator       | GT                   |   |                          |
| Value2         | 10000                |   |                          |
| ThenRule       | DisplayErrorByNumber |  | <input type="checkbox"/> |
| > ErrorNumber  |                      |   |                          |
| > Severity     |                      |   |                          |
| > MessageText  |                      |   |                          |

7. We want this to be a Lookahead rule, so we check the box at the top:

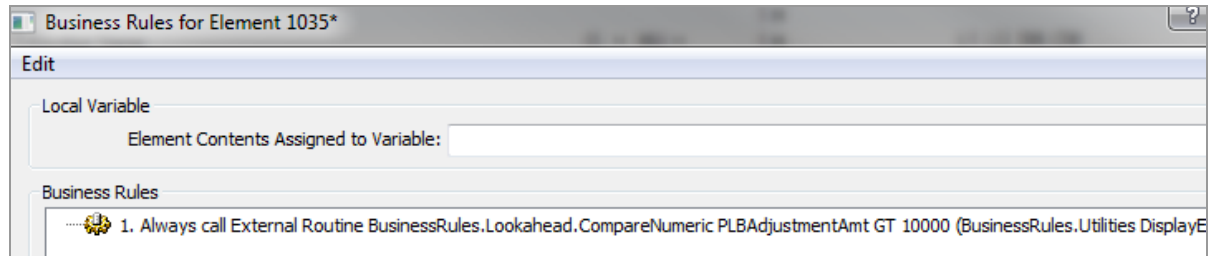
What Rule to Run

CompareNumeric  ☒ Look-Ahead Rule

**Text**

| Parameter Name | Parameter Value      |
|----------------|----------------------|
| Value1         | PLBAdjustmentAmt     |
| Operator       | GT                   |
| Value2         | 10000                |
| ThenRule       | DisplayErrorByNumber |
| > ErrorNumber  | 32010                |
| > Severity     |                      |
| > MessageText  |                      |

8. Save the rule and look at the complete rule in the outer Business Rules box:



# TIBCO Documentation and Support Services

---

## How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit <https://docs.tibco.com>.

## Product-Specific Documentation

Documentation for TIBCO® Foresight® EDISIM® is available on the [TIBCO Foresight® EDISIM® Documentation](#) page.

The following documents for this product can be found on the TIBCO Documentation site:

- *TIBCO Foresight® EDISIM® Release Notes*
- *TIBCO Foresight® EDISIM® Data Types*
- *TIBCO Foresight® EDISIM® Documentation and Demo Data Index*
- *TIBCO Foresight® EDISIM® Supported File Formats*
- *TIBCO Foresight® EDISIM® Installation Guide*
- *TIBCO Foresight® EDISIM® Introduction to EDISIM®*
- *TIBCO Foresight® EDISIM® DocStarter: Creating a Guideline from EDI Data*
- *TIBCO Foresight® EDISIM® Guideline Merge*
- *TIBCO Foresight® EDISIM® Document Builder User's Guide*
- *TIBCO Foresight® EDISIM® Error Message Numbers, Editing, and Management*
- *TIBCO Foresight® EDISIM® Validator User's Guide*
- *TIBCO Foresight® EDISIM® Using Flat Files*
- *TIBCO Foresight® EDISIM® Library User's Guide*
- *TIBCO Foresight® EDISIM® Validation Profile Files (APF)*
- *TIBCO Foresight® EDISIM® Using XML*
- *TIBCO Foresight® EDISIM® Comparator User's Guide*
- *TIBCO Foresight® EDISIM® Analyzer User's Guide*
- *TIBCO Foresight® EDISIM® Standards and Guidelines Reference Manual*
- *TIBCO Foresight® EDISIM® Test Data Generator User's Guide*
- *TIBCO Foresight® EDISIM® Self-Paced Tutorial: Introduction to EDISIM® (X12 Standards)*

- *TIBCO Foresight® EDISIM® Self-Paced Tutorial: Introduction to EDISIM® EDIFACT D99A Orders*
- *TIBCO Foresight® EDISIM® Standards Editor User's Guide*
- *TIBCO Foresight® EDISIM® Business Rules*

## **How to Contact TIBCO Support**

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit <http://www.tibco.com/services/support>.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at <https://support.tibco.com>.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to <https://support.tibco.com>. If you do not have a user name, you can request one by clicking Register on the website.

## **How to Join TIBCO Community**

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to <https://community.tibco.com>

## Legal and Third-Party Notices

---

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 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, the TIBCO logo, the TIBCO O logo, and EDISIM are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

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

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.txt 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.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 1991-2021. TIBCO Software Inc. All Rights Reserved.