Developing DTL Transformations

Version 2024.1
2024-05-02

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Developing DTL Transformations
InterSystems IRIS Data Platform Version 2024.1 2024-05-02

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

1

Introduction to DTL Tools

This topic introduces the tools that InterSystems IRIS® provides to enable you to develop and test DTL transformations.

1.1 Background

A *data transformation* creates a new message that is a transformation of another message. It is common for a production to use data transformations, to adjust outgoing messages to the requirements of the target systems.

You can create and edit a *DTL transformation* visually in the DTL editor, available in either the Management Portal or Studio. The DTL editor is meant for use by nontechnical users. The term *DTL* represents Data Transformation Language, which is the XML-based language that InterSystems IRIS uses internally to represent the definition of a transformation that you create in this editor.

You can invoke a data transformation from a business process, another data transformation, or a business rule. Note that there is overlap among the options available in business processes, data transformations, and business rules. For a comparison, see Comparison of Business Logic Tools. You can also try using these tools yourself by Creating a Data Transformation.

1.2 Introduction to the Data Transformation Builder Page

The **Data Transformation Builder** page enables you to create, edit, and compile DTL transformations.

To access this page in the Management Portal, select **Interoperability** > **Build** > **Data Transformations**.

When you display this page, it shows the last transformation you opened in this namespace, if any. This page has the following areas:

- The ribbon bar that the top displays options you can use to create and open DTL transformations, compile the currently displayed transformation, change the zoom display of the diagram, and so on.

  For information on these options, see Creating Data Transformations.
- The upper part of the left area displays the DTL diagram. The next section provides details on this area.
- The lower part of the left area displays a table that lists the actions defined in the DTL transformation. When InterSystems IRIS uses this transformation, it performs these actions in order as listed here.
- The right area displays three tabs:

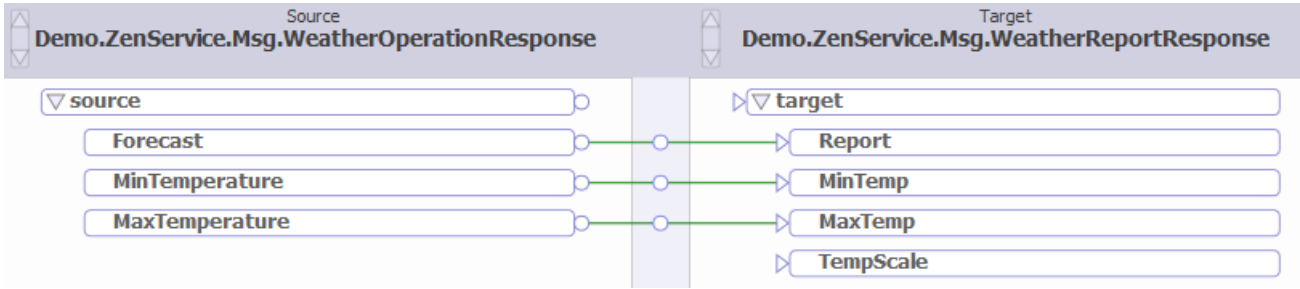
  - **Transform** — Enables you to edit information about the transformation. For details, see Specifying Transformation Details.
  - **Action** — Enables you to edit details of the selected action. Other topics describe the details for assign actions and for other kinds of actions.
  - **Tools** — Enables you to launch a wizard to test the currently displayed transformation. For details, see Testing Data Transformations.

- You can resize these three areas.

Tip: If you open a DTL transformation class in Studio, Studio displays a similar version of this page. You can view and edit the XML definition of the Data Transformation by selecting **View other code** from the Studio Data Transformation Builder page.

1.3 Introduction to the DTL Diagram

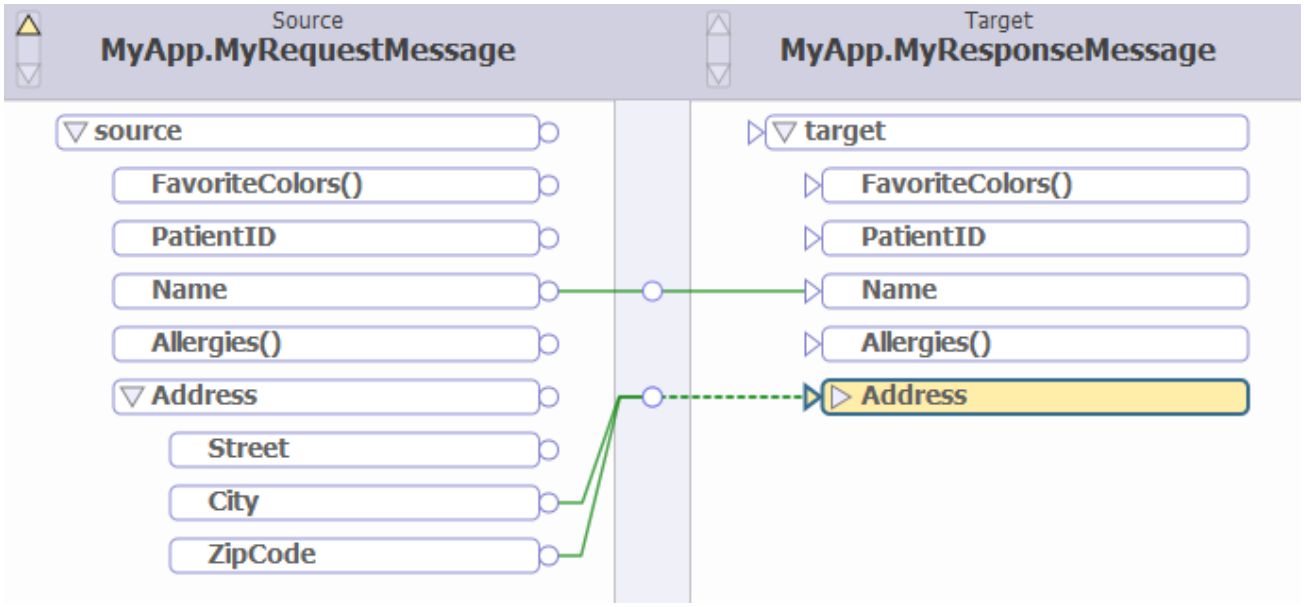
The following shows the DTL diagram for a DTL class:



Note the following points:

- The left area displays the source message. The header above the column displays the name of the source message class, and the boxes in the column display properties of the source message.
- The right area displays the target message in the same way.
- The top area includes a scroll button for each of these areas.
- The diagram shows connectors that represent actions within the transformations. The actions displayed here copy values from source properties to target properties.
- The center divider (the blue column) displays an icon on each connector line. The purpose of these icons is to enable you to select the connectors more easily. (You can select a connector line anywhere on its length, but it is easier to click the icons shown in this center divider.)

The following shows another example:



In this case, the source and target classes are more complex. Note the following additional points:

- The FavoriteColors property is defined as list of strings. This property is displayed here with parentheses () at the end of its name.

  In this example, Allergies is another collection property.

- The Address property is defined as an object that has the Street, City, and ZipCode properties. Notice that the box for this property contains a triangle inside it.

  In the left column, this property is displayed in expanded mode, so that you can see the properties. The triangle in the box is not solid and is pointing down.

  The right column, this property is displayed in collapsed mode. The triangle in this box is solid and is pointing to the right.

- For the Address properties, the connector is shown with a dashed line on the side where the Address is collapsed. This indicates that there are hidden sub-properties on this side of the **assign** action.

1.4 Controlling the Display

You can control the display of the Data Transformation Builder page in multiple ways:

- You can click a **View** option in the ribbon bar:

  Use the buttons to view both the transform diagram and the action list in the left pane of the page, or instead to collapse the section you do not want to see.

- You can select a zoom option from the drop-down list in the ribbon bar. By default, this list displays **100%**. Click a value in the list to shrink or enlarge the size of the DTL diagram.

- Use the scroll bars in the header area of the DTL diagram, as described in the previous section.

- Collapse and expand the display of properties, as described in the previous section.

1.5 Introduction to the Data Transformation List Page

The **Interoperability** > **List** > **Data Transformations** page lists the data transformation classes defined in the current namespace.

This page lists two kinds of transformations:

- DTL transformations are displayed in blue. You can double-click one to open it in the Data Transformation Builder.
- Custom transformations are displayed in black. These classes are based on Ens.DataTransform and do not use DTL. You must edit these in a supported IDE such as Studio.

To use this page, select a transformation class and then click one of the following commands in the ribbon bar:

- **Edit** — *(DTL transformations only)* Click to change or view the data transformation using the Data Transformation Builder.
- **Test** — Click to test the selected transformation class using the Test Transform wizard.

  For details, see Testing Data Transformations.
- **Delete** — Click to delete the selected transformation class.
- **Export** — Click to export the selected transformation class to an XML file.
- **Import** — Click to import a data transformation that was exported to an XML file.

You can also export and import these classes as you do any other class in InterSystems IRIS. You can use the **System Explorer** > **Globals** page of the Management Portal or use the **Export** and **Import** commands on the **Tools** menu in Studio.

1.6 Other Tools

You can also invoke a data transformation programmatically, which can be useful for testing purposes. For details, see Testing Data Transformations.

Also, because data transformations are classes, you can edit them and work with them in the same way that you do any other class.

1.7 Using Data Transformations

You can invoke a data transformation from the following parts of a production:

- From another DTL data transformation. See Adding a Subtransform Action.
- From a BPL business process. See <transform>.
- From a business rule. See Selecting the Transformation and Target of a Send Action.
- From a custom business process or a custom DTL transformation. To do so, execute it programmatically as described in Testing Data Transformations.

Note: This section applies to both DTL transformations and custom transformations.

2

Creating Data Transformations

This topic describes generally how to create and edit data transformations.

Other topics describe the syntax to use in data transformations, details for assign actions, and details for other kinds of actions.

You can also visit Online Learning to try Creating a Data Tranformation yourself.

Important: After a period of inactivity, the InterSystems Management Portal may log you out and discard any unsaved changes. Inactivity is the time between calls to the InterSystems IRIS server. Not all actions constitute a call to the server. For example, clicking **Save** constitutes a call to the server, but typing in a text field does not. Consequently, if you are editing a data transformation, but have not clicked **Save** for longer than **Session Timeout** threshold, your session will expire and your unsaved changes will be discarded. After a logout, the login page appears or the current page is refreshed. For more information, see Automatic Logout Behavior in the Management Portal.

2.1 Creating a Transformation

To create a transformation:

1. Click **New**.

   If you are currently viewing a transformation and you have made changes but have not yet saved them, InterSystems IRIS® prompts you to confirm that you want to proceed (which will discard those changes).

   InterSystems IRIS then displays a dialog box where you can specify the basic information for the transformation.

2. Specify some or all of the following information:

   - **Package** (required) — Enter a package name or click the arrow to select a package in the current namespace.

     Do not use a reserved package name; see Reserved Package Names.

   - **Name** (required) — Enter a name for your data transformation class.
   - **Description** — Enter an description for the data transformation; this becomes the class description.
   - **Source Type** and **Source Class** — Specifies the type of messages that this transformation will receive as input.

     Choose one of the following:

     – **All Messages** — This transformation can be used with any input message type.

  - **X12** — The input messages are instances of EnsLib.EDI.X12.Document.
  - **EDIFACT** — The input messages are instances of EnsLib.EDI.EDIFACT.Document.
  - **XML** — The input messages are instances of EnsLib.EDI.XML.Document.

  Or click the search icon for **Source Class** and then select the class.

- **Source Document Type** (applicable only if the messages are virtual documents) — Enter or choose the document type of the source messages. You can choose any type defined in the applicable schemas loaded into this namespace.
- **Target Type** and **Target Class** — Specifies the type of messages that this transformation will generate as output. See the choices for **Source Type** and **Source Class**.
- **Target Document Type** (applicable only if the messages are virtual documents) — Enter or choose the document type of the target messages. You can choose any type defined in the applicable schemas loaded into this namespace.

Apart from **Package** and **Name**, you can edit all these details later.

3. Specify details on the **Transform** tab. See Specifying Transformation Details.

2.2 Opening an Existing Transformation

To open a transformation:

1. Click **Open**.

   If you are currently viewing a transformation and you have made changes but have not yet saved them, InterSystems IRIS prompts you to confirm that you want to proceed (which will discard those changes).

2. Click the package that contains the transformation.

   Then click the subpackage as needed.

3. Click the transformation class.
4. Click **OK**.

2.3 Specifying Transformation Details

For a transformation, the Transform tab displays details that apply to the transformation as a whole. You may or may not have already specified some of these details. Other details can be edited only here. These details are as follows:

- **Name** (read-only) — Complete package and class name of the data transformation class.
- **Create** — Specifies how the transformation should create the target message. Choose one of the following:
  - **new** — Create a new object of the target class (and type, if applicable), before executing the elements within the data transformation. This is the default.
  - **copy** — Create a copy of the source object to use as the target object, before executing the elements within the transform.
  - **existing** — Use an existing object, provided by the caller of the data transformation, as the target object. See the following subsection.

- **Source Class** — Specifies the type of messages that this transformation will receive as input. For details, see Creating a Transformation.
- **Source Document Type** (applicable only if the messages are virtual documents) — Specifies the document type of the source messages.
- **Target Class** — Specifies the type of messages that this transformation will generate as output. For details, see Creating a Transformation.
- **Target Document Type** (applicable only if the messages are virtual documents) — Specifies the document type of the target messages.
- **Language** — Specifies the language you will use in any expressions in this DTL. This should be **objectscript**.
- **Report Errors** — Specifies whether InterSystems IRIS should log any errors that it encounters when executing this transform. If you select this option, InterSystems IRIS logs the errors as *Warnings* in the Event Log. InterSystems IRIS also returns a composite status code containing all errors as its return value. This option is selected by default.
- **Ignore missing source segments and properties** — Specifies whether to ignore errors caused by attempts to get field values out of absent source segments of virtual documents or properties of objects. If you select this option, InterSystems IRIS suppresses these errors and does not call subtransforms where the named source is absent. This option is selected by default.

  You can precisely control the behavior by including tests and conditional logic branches to confirm that any required elements are present.
- **Treat empty repeating fields as null** — Specifies whether InterSystems IRIS skips the following actions for repeating fields when the fields are empty:
  - **foreach** actions — If you select this option, InterSystems IRIS does not execute **foreach** actions on repeating fields that are empty.
  - **assign** actions — If you select this option, InterSystems IRIS does not execute **assign** actions on repeating fields if you use shortcut notation to indicate that both the `source` and `target` fields are repeating fields, and the `source` field is empty. For example, if the `source.{PV1:AdmittingDoctor()}` field is empty and you select this option, then InterSystems IRIS does not execute the following action:

    ```
    <assign value='source.{PV1:AdmittingDoctor()}'
    property='target.{PV1:AdmittingDoctor()}' action='set'.
    ```

    However, InterSystems IRIS *does* execute the following similar action since the `target` field is not a repeating field:

    ```
    <assign value='source.{PV1:AdmittingDoctor()}'
    property='target.{PV1:AdmittingDoctor(1)}' action='set' />
    ```

    This option is cleared by default.
- **Description** — Description of the data transformation.

2.3.1 Using the Create existing Option

For **Create**, the **existing** option enables you to specify the target as an existing object, which results in a performance improvement. This option applies when you invoke a series of transformations programmatically (or perform other sequential processing). You would use this option in cases like the following scenario:

- You have three transformations that you want to perform in sequence:
  1. MyApp.ADTTransform — Uses the **new** option for **Create**.
  2. MyApp.MRNTransform — Uses the **existing** option for **Create**.

3. MyApp.LabXTransform — Uses the **existing** option for **Create**.

- You invoke the transforms as follows:

```
do MyApp.ADTTransform.Transform(message,.target)
do MyApp.MRNTransform(target,.newtarget)
do MyApp.LabXTransform(newtarget,.outmessage)
```

2.4 Editing Transformation Actions

This section describes generally how to add and edit the actions in a transformation. It includes the following subsections:

- Adding an Action
- Editing an Action

Other topics describe the syntax to use in data transformations, details for assign actions, and details for other kinds of actions.

2.4.1 Adding an Action

To add an action, you can always do the following:

1. Optionally click a source or target property, depending on the kind of action you want to add.
2. Select an action from the **Add Action** drop-down list in the ribbon bar.
3. Edit the details for this action on the **Action** tab.

   If applicable, the property that you selected is shown in the **Property** field, for use as a starting point. Optionally, you can disable the action with the **Disabled** check box. If you disable a **foreach** or **if** action, all actions within the block are also disabled.

Other techniques are possible for **assign** actions, as discussed in Adding Assign Actions.

2.4.2 Editing an Action

To edit an action, first select it. To do so:

- Click the corresponding row in the table below the diagram.
- If the diagram displays the action, click the icon on the corresponding connector line, in the center divider.

  When you click on an item, it changes color, the connector line turns bold, and the source and target properties are highlighted in color. This means the connector is selected, as shown in the following figure.

Now edit the values on the **Action** tab. Optionally, you can disable the action with the **Disabled** check box. If you disable a **foreach** or **if** action, all actions within the block are also disabled.

Tip: If you double-click a property in the diagram, InterSystems IRIS updates the currently selected action, if applicable. If you double-click a field in the source, then the editor interprets it as your wanting to set the value for the selected action. Similarly, if you double-click a target field, the editor interprets it as your wanting to set the target for the selected action.

2.5 Rearranging Actions

InterSystems IRIS executes the actions in the order they are listed in the table below the diagram.

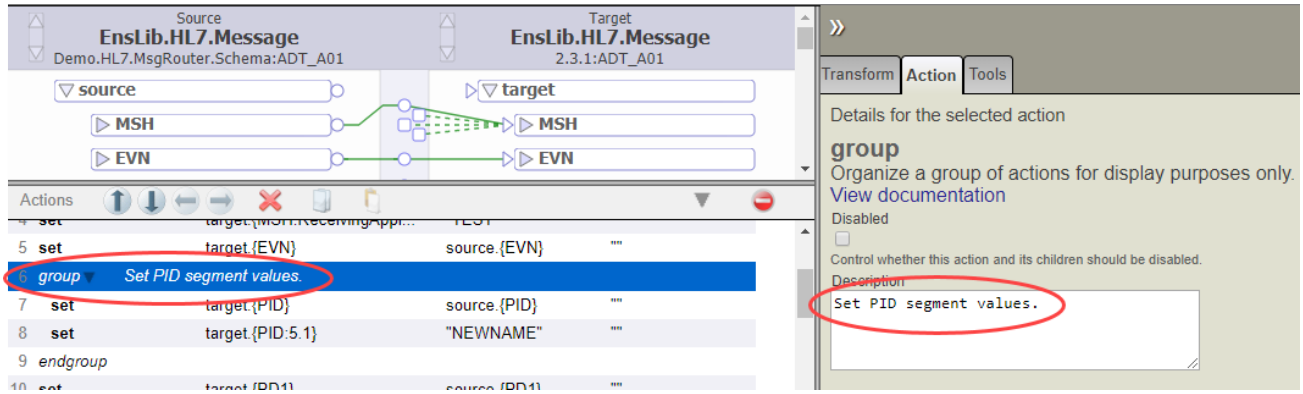
To rearrange actions, you must use the table below the DTL diagram, as follows:

1. Click the row corresponding to that action.
2. Click one of the following icons in that row, as needed:

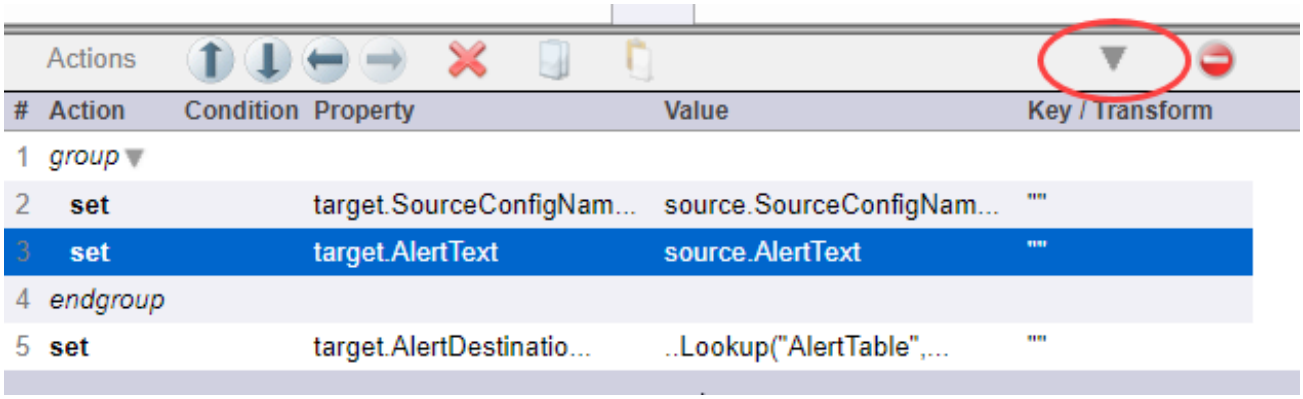
Tool	Description
	Move the selected action up one position. If the action is the first action in a group, for example a **for each** or **if** block, then this moves the action up and out of the group.
	Move the selected action down one position. If the action is the last action in a group, then this moves the action just after the group. For example, if the action is the last action in an **if** block, the action is moved right after the block. If the action is the last action in an **if** block just before the **else**, then this moves the action into the first position in the **else** block.
	Move the selected action out of the current group, for example a **for each** or **if** block. This moves the action out of the current group to the position immediately before the group.
	Move the selected action into the next group of actions, for example, a **for each** or **if** block.
	Remove all the actions of the data transformation.
	Remove the action in this row.

2.6 Working with Groups of Actions

You can gather actions into a display group by using the **group** action. Grouping actions helps organize them in the table below the diagram. The description that you define on the **Action** tab appears in the list to help you identify a group.



To move an action in or out of a group, select the action in the list and click (move into group) or (move out of group). To make the list more readable, you can collapse or expand groups as you review the list. To collapse a group and hide the actions it contains, click next to the action name. To expand a group, click . You can also collapse and expand all groups at once using the and buttons in the table's **Actions** bar.



Note: You can also expand and collapse blocks of actions created by **if** , **for each** , **switch** , and **case** actions.

2.7 Undoing a Change

To undo the previous change, click the Undo button .

2.8 Saving a Transformation

To save a transformation, do one of the following:

- Click **Save**.
- Click **Save As**. Then specify a new package, class name, and description and click **OK**.
- Click **Compile**. This option saves the transformation and then compiles it.

2.9 Compiling a Transformation

To compile a transformation, click **Compile**. This option saves the transformation and then compiles it.

2.10 Deleting a Transformation

To delete a transformation, you must use a different page, the **Interoperability** > **List** > **Data Transformations** page.

To delete a transformation:

1. Click the row that displays its name.
2. Click the **Delete** button.
3. Click **OK** to confirm this action.

3

Syntax Rules

This topic describes the syntax rules for referring to properties and for creating expressions within various DTL actions.

3.1 References to Message Properties

In most actions within a transformation, it is necessary to refer to properties of the source or target messages. The rules for referring to a property are different depending on the kind of messages you are working with.

- For messages other than virtual documents, use syntax like the following:

  ```
  source.propertyname
  ```

  Or:

  ```
  source.propertyname.subpropertyname
  ```

  Where *propertyname* is a property in the source message, and *subpropertyname* is a property of that property.

  If the message includes a collection property, see Special Variations for Repeating Fields. Some of the information there applies to both virtual documents and standard messages.
- For virtual documents other than XML virtual documents, use the syntax described in Syntax Guide for Virtual Property Paths. Also see the following subsection.
- For XML virtual documents, see Routing XML Virtual Documents in Productions.

3.2 Literal Values

When you assign a value to a target property, you often specify a literal value. Literal values are also sometimes suitable in other places, such as the value in a **trace** action.

A literal value is either of the following:

- A numeric literal is just a number. For example: 42.3
- A string literal is a set of characters *enclosed by double quotes*. For example: `"ABD"`

Note: This string cannot include XML reserved characters. For details, see XML Reserved Characters.

For virtual documents, this string cannot include separator characters used by that virtual document format. See Separator Characters in Virtual Documents and When XML Reserved Characters Are Also Separators.

3.2.1 XML Reserved Characters

Because DTL transformations are saved as XML documents, you must use XML entities in the place of XML reserved characters:

To include this character...	Use this XML entity...
>	>
<	<
&	&
'	'
"	"

For example, to assign the value `Joe's "Good Time" Bar & Grill` to a target property, set **Value** equal to the following:

```
"Joe&apos;s &quot;Good Time&quot; Bar &amp; Grill"
```

This restriction does not apply inside **code** and **sql** actions, because InterSystems IRIS® automatically wraps a CData block around the text that you enter into the editor. (In the XML standard, a CData block encloses text that should not be parsed as XML. Thus you can include reserved characters in that block.)

3.2.2 Separator Characters in Virtual Documents

In most of the virtual document formats, specific characters are used as separators between segments, between fields, between subfields, and so on. If you need to include any of these characters as literal text when you are setting a value in the message, you must instead use the applicable escape sequence, if any, for that document format.

See the following topics:

- EDIFACT Separators
- X12 Separators

Important: In a data transformation, the separator characters and escape sequences can be different for the source and target messages. InterSystems IRIS automatically adjusts values as needed, after performing the transformation. This means that you should consider only the separator characters and escape sequences that apply to the *source* message.

3.2.3 When XML Reserved Characters Are Also Separators

- If the character (for example, &) is a separator and you want to include it as a literal character, use the escape sequence that applies to the virtual document format.
- In all other cases, use the XML entity as shown previously in XML Reserved Characters.

3.2.4 Numeric Character Codes

You can include decimal or hexadecimal representations of characters within literal strings.

The string &#*n*; represents a Unicode character when *n* is a decimal Unicode character number. One example is é for the Latin e character with acute accent mark (é).

Alternatively, the string &#x*h*; represents a Unicode character when *h* is a hexadecimal Unicode character number. One example is ¿ for the inverted question mark (¿).

3.3 Valid Expressions

When you assign a value to a target property, you can specify an expression, in the language that you selected for the data transformation. You also use expressions in other places, such as the condition for an **if** action, the value in a **trace** action, statements in a **code** action, and so on.

The following are all valid expressions:

- Literal values, as described in the previous section.
- Function calls (InterSystems IRIS provides a set of utility functions for use in business rules and data transformations. For details, see Utility Functions for Use in Productions.) InterSystems IRIS provides a wizard for these.
- References to properties, as described in References to Properties.
- References to the `aux` variable passed by the rule. If the data transformation is called from a rule, it supplies the following information in the **aux** variable:
  - aux.BusinessRuleName—Name of the rule.
  - aux.RuleReason—Reason that the rule was fired. It is the same name as used in the logging. An example value is 'rule#1:when#1'. If the RuleReason is longer than 2000 characters, it is truncated to 2000 characters.
  - aux.RuleUserData—Value that was assigned in the rule to the property 'RuleUserData'. The value of 'RuleUserData' is always the last value that it was set to.
  - aux.RuleActionUserData—Value that was assigned in the rule when or otherwise clause to the property 'RuleActionUserData'.

  If the data transformation is called directly from code and not from a rule, the code can pass the auxiliary data in the third parameter. If your data transformation may be called from code that does not set the third parameter, your DTL code should check that the `aux` variable is an object in an **if** action using the **$ISOBJECT** function.
- Any expression that combines these, using the syntax of the scripting language you chose for data transformation. See Specifying Transformation Details. Note the following:
  - In ObjectScript, the concatenation operator is the _ (underscore) character, as in:

    ```
    value='"prefix"_source.{MSH:ReceivingApplication}_"suffix"'
    ```
  - To learn about useful ObjectScript string functions, such as $CHAR and $PIECE, see the ObjectScript Reference.
  - For a general introduction, see Using ObjectScript.

4

Adding Assign Actions

This topic provides details on adding different kinds of **assign** actions to a DTL transformation.

Important: For virtual documents, do not manually change escape sequences in the data transformation; InterSystems IRIS® handles these automatically.

For information on adding other kinds of actions, see Adding Other Actions.

For information about working with groups of actions, see Working with Groups of Actions .

4.1 Introduction

There are five kinds of **assign** actions: **set**, **clear**, **remove**, **append**, and **insert**. After you create any kind of **assign** action, you can change the type. To do so, select a different value in the **Action** drop-down in the **Action** tab.

InterSystems IRIS represents each **assign** action with a connector line in the DTL diagram.

4.1.1 Objects and Object References

If you **assign** from the top-level source object or any object property of another object as your source, the target receives a cloned copy of the object rather than the object itself. This prevents inadvertent sharing of object references and saves the effort of generating cloned objects yourself.

If you instead want to share object references between source and target, you must **assign** from the source to an intermediate temporary variable, and then **assign** from that variable to the target.

4.2 Copying the Source Message

To create an **assign** action that copies the source message:

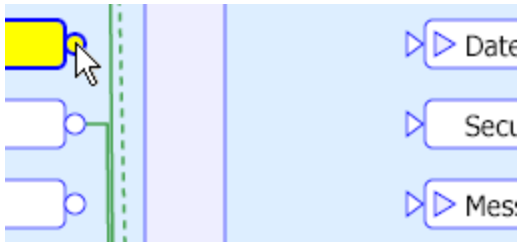
1. Drag the circle tab to the right of the source message. Hold down the mouse button.
2. Drag the cursor to the triangle tab to the left of the target message until its box changes color.
3. Release the left mouse button.

4.3 Copying a Value from a Source Property to a Target Property

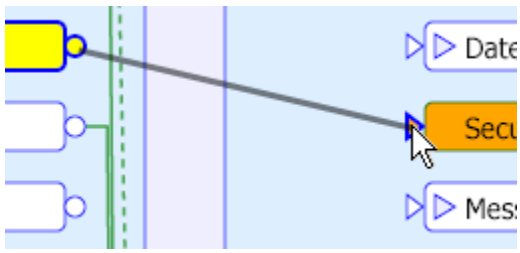
To create an **assign** action that copies a value from a source property to a target property:

1. Drag a value from a source property to a target property. To do this:

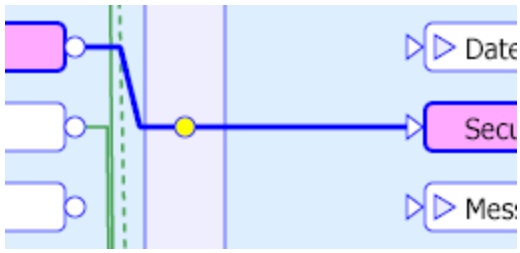
   a. Click the circle tab to the right of the source property. Hold down the mouse button. The display looks similar to this:



   b. Drag the cursor to the triangle tab to the left of the target property until its box changes color. The display looks similar to this:



   c. Release the left mouse button. The display looks similar to this:



   The table below the diagram now shows the details of the assign action.

2. Optionally edit the details on the **Action** tab.

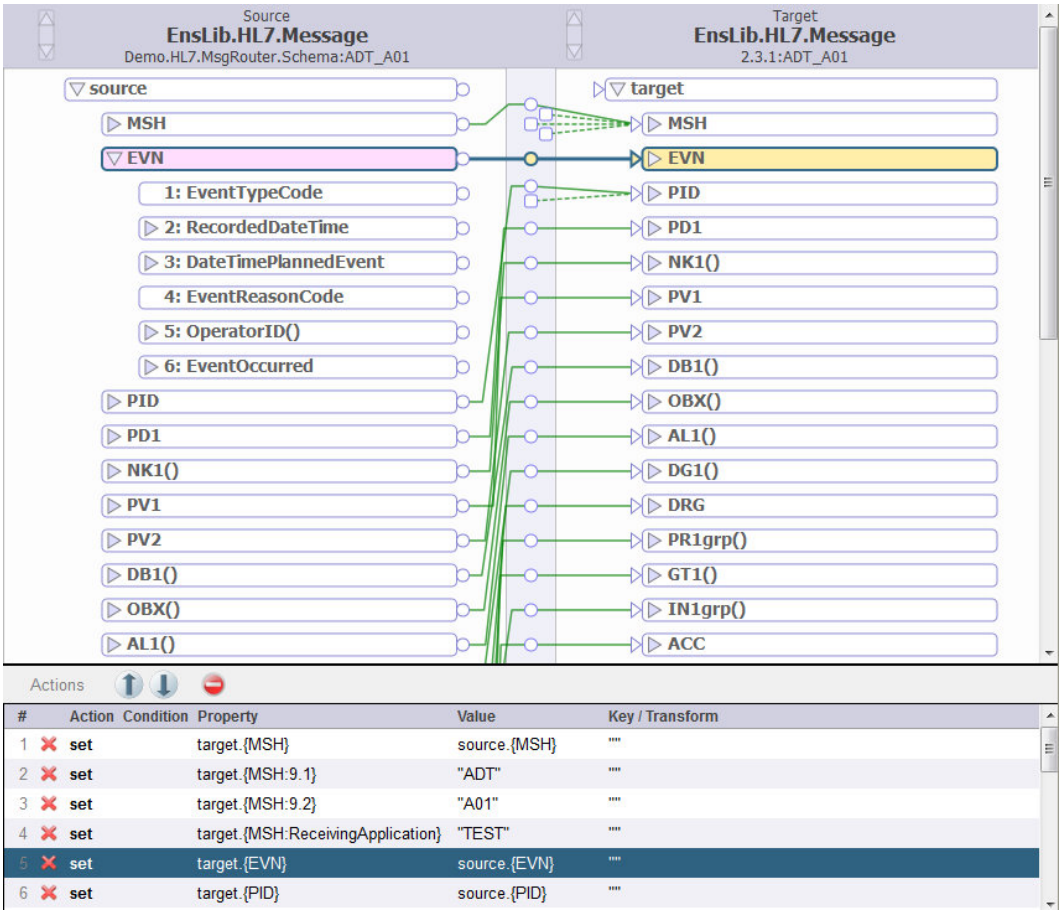
This **assign** action uses **set**.

4.4 Copying Values of All Sub-properties

If parent properties in the source and target are identical and the source and target have the same type, you can assign the values of all the sub-properties at once. In this case there is no need to expand the parent properties to reveal the sub-properties. Simply drag the cursor from the parent property on the source side to the parent property on the target side.

In the following DTL diagram, the single connector between the EVN property on the source side and the EVN property on the target side includes all of the following sub-properties of EVN:

- *EventTypeCode*
- *RecordedDateTime* and all of its sub-properties
- *DateTimePlannedEvent* and all of its sub-properties
- *EventReasonCode*
- *OperatorID*, all of its iterations, and all of its sub-properties
- *EventOccurred* and all of its sub-properties



This **assign** action uses **set**.

Note: If the source and target types are different, you cannot use this feature to assign subproperties, even if the structures appear to be parallel. For such a transformation, you must assign each leaf node of a structure independently and add a **for each** action to process iterations. See Adding a For Each Action for details on the **for each** action.

4.5 Assigning a Literal Value to a Target Property

To assign a literal value to a target property:

1. Select the target property.
2. Click **set** from the **Add Action** drop-down list. The **Action** tab for this operation displays.
3. Type a literal numeric or string value in the **Value** field:
   - A numeric literal is just a number. For example: 42.3
   - A string literal is a set of characters *enclosed by double quotes*. For example: `"ABD"`

     Note: This string cannot include XML reserved characters. For virtual documents, this string cannot include separator characters used by that virtual document format. For details, see Syntax Rules.
4. Click **Save**.

4.6 Using an Expression for the Value of a Target Property

A literal value, as described in the previous section, is a simple kind of expression. In some cases, you might want to use a more complex expression as the value of a target property. To do so, either:

- To create an expression that uses a function, click the search button  next to the **Value** field. This invokes the **Data Transform Function Wizard**, which is described in the following subsection.
- To create a more complex expression, type the expression into the **Value** field.

  See Valid Expressions. Make sure that the expression is valid in the scripting language you chose for the data transformation; see Specifying Transformation Details.

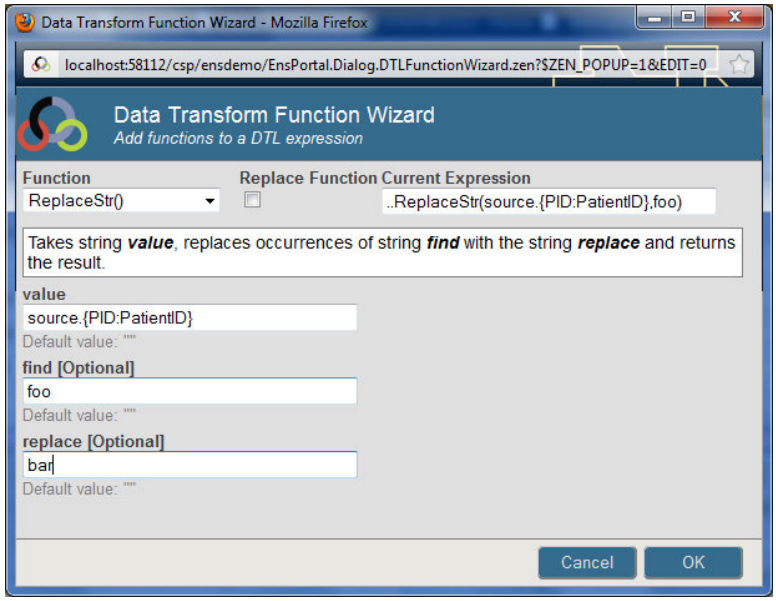
4.6.1 Using the Data Transform Function Wizard

To use the **Data Transform Function Wizard**:

1. Select a **Function** from the drop-down list.

   More fields display as needed to define the expression.

   If you select **Repeat Current Function** from the drop-down list, a copy of the current function is inserted as a parameter of the itself, which creates a recursive call to the function.
2. Edit the fields as needed. For instructions, see the context-sensitive help in the dialog.
3. Click **OK** to save your changes and exit the wizard.



4.7 Assigning a Value to a Collection Item

This section applies to the following kinds of collections:

- Collection properties in standard production messages.
- Repeating fields in XML virtual documents.

To change the value of an item from a collection:

1. Select the target list property or array property.
2. Click **set** from the **Add Action** drop-down list. The **Action** tab for this operation displays.
3. In the **Property** field, edit the value in parentheses so that it identifies the item to change.

   For array properties, use the key of the array item. For list properties, use the index of the list item. For repeating fields in virtual documents, use the index of the segment or field.

   For example, suppose that you originally see this:

   ```
   target.MyArrayProp.(1)
   ```

   Edit the field to contain this instead:

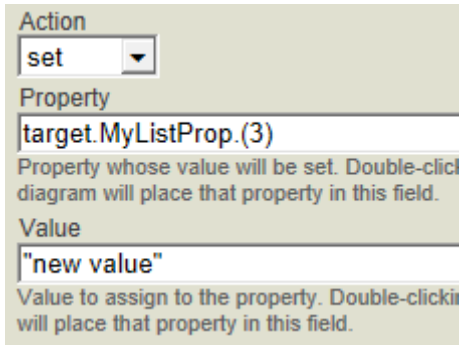
   ```
   target.MyArrayProp("key2")
   ```

4. Edit **Value** to contain a literal value or other valid expression.

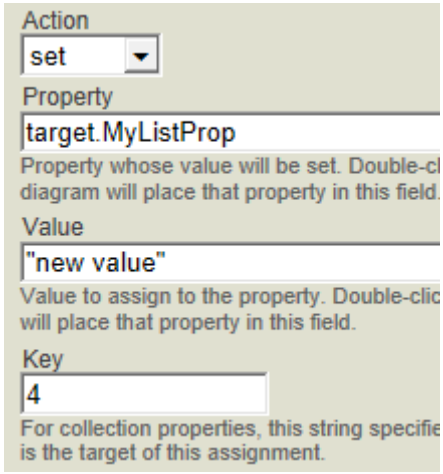
   See Valid Expressions. Make sure that the expression is valid in the scripting language you chose for the data transformation; see Specifying Transformation Details.

5. Click **Save**.

For example:



Or, edit the **Property** field to remove the trailing `.(1)` from the displayed value. Then use **key** to specify identify the item to change, as described in the next section. For example:



4.8 Inserting a List Item

This section applies to list properties (but not array properties) in standard production messages. You can also use this action with XML virtual documents; see Routing XML Virtual Documents in Productions.

To insert an item into a list:

1. Select the target list property or array property.
2. Click **insert** from the **Add Action** drop-down list. The **Action** tab for this operation displays.
3. In the **Property** field, remove the trailing `.(1)` from the displayed value.

   For example, suppose that you originally see this:

   ```
   target.MyListProp.(1)
   ```

   Edit the field to contain this instead:

   ```
   target.MyListProp
   ```

4. Edit **Value** to contain a literal value or other valid expression.

   See Valid Expressions. Make sure that the expression is valid in the scripting language you chose for the data transformation; see Specifying Transformation Details.

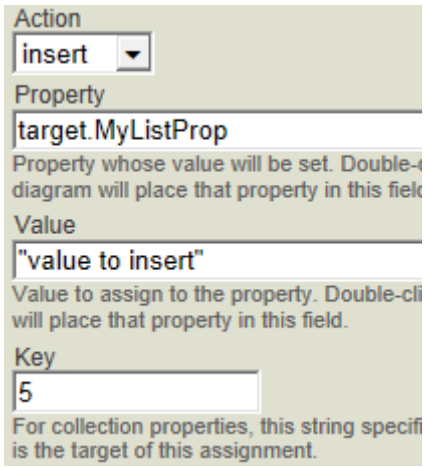
5. For **key**, identify the index position for the new item.

   For example:

   ```
   5
   ```

6. Click **Save**.

For example:



4.9 Appending a List Item

This section applies to list properties (but not array properties) in standard production messages. You can also use this action with XML virtual documents; see Routing XML Virtual Documents in Productions.

To insert an item into a list:

1. Select the target list property or array property.
2. Click **append** from the **Add Action** drop-down list. The **Action** tab for this operation displays.
3. In the **Property** field, remove the trailing `.(1)` from the displayed value.

   For example, suppose that you originally see this:

   ```
   target.MyListProp.(1)
   ```

   Edit the field to contain this instead:

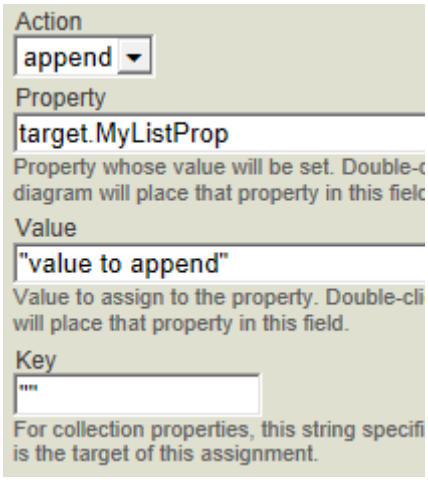
   ```
   target.MyListProp
   ```

4. Edit **Value** to contain a literal value or other valid expression.

   See Valid Expressions. Make sure that the expression is valid in the scripting language you chose for the data transformation; see Specifying Transformation Details.

5. Click **Save**.

For example:



4.10 Removing a Collection Item

This section applies to collection properties (lists and arrays) in standard production messages. You can also use this action with XML virtual documents; see Routing XML Virtual Documents in Productions.

To remove an item from a collection:

1. Select the target list property or array property.
2. Click **remove** from the **Add Action** drop-down list. The **Action** tab for this operation displays.
3. In the **Property** field, remove the trailing `.(1)` from the displayed value.

   For example, suppose that you originally see this:

   ```
   target.MyArrayProp.(1)
   ```

   Edit the field to contain this instead:

   ```
   target.MyArrayProp
   ```

4. For **key**, identify the item to remove.

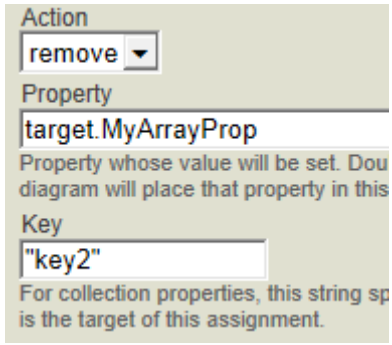
   For array properties, use the key of the array item. For list properties, use the index of the list item. For repeating fields in virtual documents, use the index of the segment or field.

   For example:

   ```
   "key2"
   ```

5. Click **Save**.

For example:



4.11 Clearing a Collection Property

This section applies to collection properties (lists and arrays) in standard production messages. You can also use this action with XML virtual documents; see Routing XML Virtual Documents in Productions.

To clear the contents of a collection:

1. Select the target list property or array property.
2. Click **clear** from the **Add Action** drop-down list. The **Action** tab for this operation displays.
3. In the **Property** field, remove the trailing `.(1)` from the displayed value.

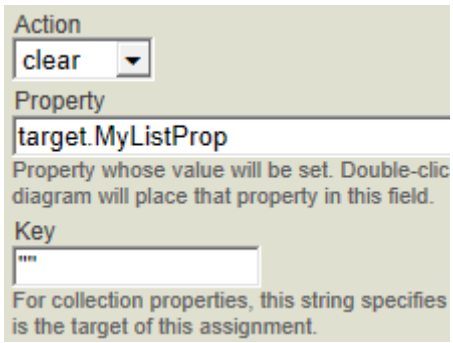
   For example, suppose that you originally see this:

   ```
   target.MyArrayProp.(1)
   ```

   Edit the field to contain this instead:

   ```
   target.MyArrayProp
   ```
4. Click **Save**.

For example:



5

Adding Other Actions

This topic provides details on adding other kinds of actions to a DTL transformation.

For information about working with groups of actions, see Working with Groups of Actions.

For information on adding **assign** actions, see Adding Assign Actions.

5.1 Adding an If Action

An **if** action executes other actions conditionally, depending on the value of an expression that you provide. InterSystems IRIS® represents each **if** action as a connector line in the DTL diagram.

To add an **if** action:

1. If the condition depends upon the value of a source property, click that property.
2. Select **if** from the **Add Action** drop-down list.

   On the **Action** tab, the **Condition** field automatically contains the name of the source property that you had selected.

   The area below the diagram contains three new rows. The Actions column displays the following labels for these rows:

   - **if** — This row marks the beginning of actions to perform if the condition is true.
   - **else** — This row marks the beginning of actions to perform if the condition is false.
   - **endf** — This row marks the end of the **if** action.

3. Edit the **Condition** field so that it contains an expression that evaluates to either true or false.

   For example:

   ```
   source.ABC = "XYZ"
   ```

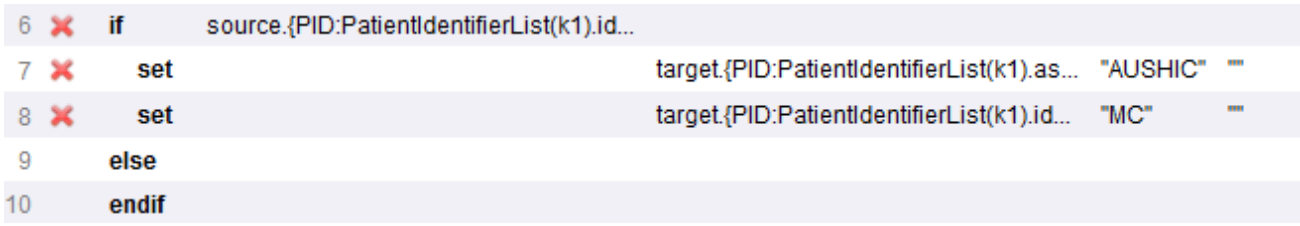
   Notes:

   - To create an expression that uses a function, click the search button  next to the **Value** field. This invokes the **Data Transform Function Wizard**, which is described earlier.
   - To create a more complex expression, type the expression into the **Value** field. See Valid Expressions. Make sure that the expression is valid in the scripting language you chose for the data transformation; see Specifying Transformation Details.

4. To add actions to perform when the condition is true:
   a. Click the **if** row.
   b. Select an item from the **Add Action** drop-down list.
   c. Edit the values in the **Action** tab as needed.
   d. Repeat as necessary.

   You can include **assign** actions, **if** actions, and **for each** actions.
5. To add actions to perform when the condition is false:
   a. Click the **else** row.
   b. Continue as described in the preceding item.

The details are then shown in the block below the DTL diagram. For example:



Note: It is not required to have any actions for the **if** branch or for the **else** branch. If there are no actions in either branch, the **if** action has no effect.

5.2 Adding a For Each Action

The **for each** action enables you to define a sequence actions that is executed iteratively, once for each member of one of the following:

- A collection property (for a standard message) .
- A repeating property (for a virtual document).
- A set of subdocuments in a document (for a virtual document).

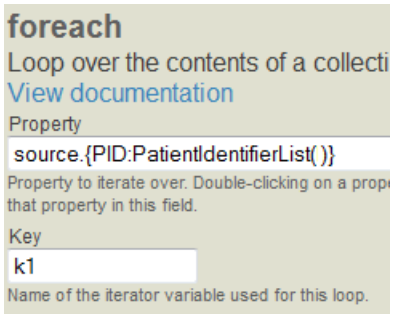
InterSystems IRIS represents each **for each** action as a connector line in the DTL diagram.

You can break out of a **for each** loop at any time by adding a **break** action within the loop.

To add a **for each** action:

1. Select a collection or repeating property in the source message.
2. Select **for each** from the **Add Action** drop-down list.

   On the **Action** tab, the **Property** field automatically contains the name of the selected source property, and the **Key** field automatically contains `k1`, as illustrated below:



For the **for each** action, the **Key** field specifies the name of an iterator variable.

The **Property** field should not include the iterator key within the parentheses. For example, the following is correct:

```
source.{PID:PatientIdentifierList( )}
```

The **for each** iterates through the `PatientIdentifierList` repeating fields, starting with the first one (numbered 1) and ending with the last one.

3. On the **Action** tab, the **Unload** check box controls whether to generate code to unload open objects or segments.

   If the **unload** is checked for a **for each** action, then code is generated in the Transform method to try to unload/unswizzle open object(s) or segment(s) for the property collection at the end of each loop. Unsaved virtual document segments are saved and finalized. If the property is the source object, the source object is usually already saved.

   You may still need to manually add actions to unload the target collection's objects or segments. For details on some strategies, see Unloading Target Collections.

   The unload of the **for each** property collection may be unnecessary – for example, for HL7, code generated using CopyValues does not instantiate the source segments.

4. To add actions to the **for each** block, click the **for each** action and then add the appropriate actions.

The details are then shown in the block below the DTL diagram. For example (partial):

#	Action	Condition	Property	Value	Key / Transform
1	**for each**		source.{PID:PatientIdentifierList( )}		k1
2	**if**	source.{PID:PatientIdentifierList(k1).id...			
3	**set**		target.{PID:PatientIdentifierList(k1).id...	"MR"	""
4	**else**				
5	**endif**				
6	**if**	source.{PID:PatientIdentifierList(k1).id...			

If the <foreach> applies to a collection property in a message, the sequence of activities is executed iteratively, once for every element that exists within the collection property. If the element is null, the sequence is not executed. The sequence is executed if the element has an empty value, that is, the separators are there but there is no value between them, but is not executed for a null value, that is, the message is terminated before the field is specified.

5.2.1 Shortcuts for the For Each Action

When you are working with virtual documents, InterSystems IRIS provides a shortcut notation that iterates through every instance of a repeating field within a document structure. This means you do not actually need to set up multiple nested **for each** loops to handle repeating fields; instead you create a single **assign** action using a virtual property path with empty parentheses within the curly bracket { } syntax. For information, see Curly Bracket {} Syntax.

Note: If the source and target types are different, you cannot use this shortcut for the **for each** action. Use an explicit **for each** action in these cases.

5.2.2 Unloading Target Collections

While the **Unload** option automatically removes objects from a source collection, you need to add custom code at the end of a **for each** action to remove objects from a target collection. In a simple example in which the target is a complex record, you could use the following code to save the current target record and then unload it:

```
Do target.Record16.GetAt(k1).%Save(0)
Do target.Record16.%UnSwizzleAt(k1)
```

In other scenarios, it might be better to avoid loading the target altogether in order to avoid issues where the target is not unloaded. For example, suppose you have an object that has a parent/child property with many children. Within the **for each** action, you have a subtransform combined with *prop*`SetObjectId(parentId))`, where *prop* is the name of the property.

In this example, the target is the batch object, the target class is `Demo.RecordMapBatch.Map.TrainDataOut.BatchOut` and the record class is `Demo.RecordMapBatch.Transform.Optimized.Record`

Before your **for each** loop, you need to create an empty target and assign its ID to a property `BatchOutID`:

```
<assign value='target.%Save()' property='tSC' action='set' />
<assign value='target.%Id()' property='BatchOutID' action='set' />
<assign value='target' property='' action='set' />
```

Then, in the **for each** loop, you can use code that directly impacts the target without having the target instantiated. For example:

```
<assign value='""' property='record' action='set' />
<subtransform class='Demo.RecordMapBatch.Transform.Optimized.Record' targetObj='record'
sourceObj='source.Records.(k1)' />

<comment>
<annotation>Assign record to target directly. </annotation>
</comment>
<assign value='record.%ParentBatchSetObjectId(BatchOutID)' property='tSC' action='set' />
<assign value='record.%Save()' property='tSC' action='set' />
```

Then, before the DTL ends, set the variable `target` back to the expected product of the DTL. For example:

```
<assign value='##class(Demo.RecordMapBatch.Map.TrainDataOut.BatchOut).%OpenId(BatchOutID)'
property='target' action='set' />
```

5.2.3 Avoiding <STORE> Errors with Large Messages

As you loop over segments in messages or object collections, they are brought into memory. If these objects consume all the memory assigned to the current process, you may get unexpected errors. You can avoid these errors in the source collection by using the **Unload** option in the Management Portal. For some strategies for removing objects in a target collection, see Unloading Target Collections.

As another strategy, if you are processing many segments in a **for each** loop, you can call the **commitSegmentByPath()** method on both the source and target as the last step in the loop. Similarly, for object collections, use the **%UnSwizzleAt()** method.

The method **commitCollectionOpenSegments()** loops through the runtimePath looking for open segments within the specified collection path and calls **commitSegmentByPath()** for each open segment. This method is available from the classes EnsLib.EDI.X12.Document, EnsLib.EDI.ASTM.Document, EnsLib.EDI.EDIFACT.Document, and EnsLib.HL7.Message.

If you cannot make code changes, a temporary workaround is to increase the amount of memory allocated for each process. You can change this by setting the *bbsiz* parameter on the **Advanced Memory Settings** page in the Management Portal. Note that this action requires a system restart, and you should consult with your system administrator before performing it.

5.3 Adding a Subtransform Action

A **subtransform** invokes another transformation (an ordinary transformation), often within a **for each** loop. Subtransformations are particularly useful with virtual documents, because EDI formats are typically based on a set of segments that are used in many message types. The ability to reuse a transformation within another transformation means that you can create a reusable library of segment transformations that you can call as needed, without duplicating code transformation.

InterSystems IRIS does not represent a **subtransform** action in the DTL diagram.

To add a **subtransform** action:

1. Select **subtransform** from the **Add Action** drop-down list.
2. On the **Action** tab, specify the following details:
   - **Transform Class** — Specifies the data transformation class to use. This can be either a DTL transformation or a custom transformation. For information on custom transformations, see Defining Custom Transformations. You must enter the class.
   - **Source Property** — Identifies the property being transformed. This may be an object property or a virtual document property path. Generally it is a property of the source message used by the transformation. You must enter the source property.
   - **Target Property** — Identifies the property into which the transformed value will be written. This may be an object property or a virtual document property path. Generally it is a property of the target message used by the transformation. You must enter the target property.
   - **Auxiliary Property**—Optionally, specifies a value that is passed to the subtransform. The subtransform accesses the value as the `aux` variable.
   - **Disabled**—Optionally, specifies that the subtransform is disabled.
   - **Description**—Optionally, specifies a text description of the subtransform.

Note: In the case of a **subtransform** with **Create** as `new` or `copy`, it is not necessary to have a pre-existing target object.

5.4 Adding a Trace Action

A **trace** action generates a trace message, which is helpful for diagnosis. If the Log Trace Events setting is enabled for the parent business host, this message is written to the Event Log. If the Foreground setting is enabled for the parent business host, the trace messages are *also* written to the Terminal window.

InterSystems IRIS does not represent a **trace** action in the DTL diagram.

To add a **trace** action:

1. Select `trace` from the **Add Action** drop-down list.
2. On the **Action** tab, specify the following:

- **Value** — Specify a literal value or other valid expression.

  See Valid Expressions. Make sure that the expression is valid in the scripting language you chose for the data transformation; see Specifying Transformation Details.

- **Description** — Specify an optional description.

The **trace** action generates trace message with User priority; the result is the same as using the `$$$TRACE` macro in ObjectScript.

5.5 Adding a Code Action

A **code** action enables you to execute one or more lines of user-written code within a DTL data transformation. This option enables you to perform special tasks that are difficult to express using the DTL elements. InterSystems IRIS does not represent a **code** action in the DTL diagram.

To add a **code** action:

1. Select **code** from the **Add Action** drop-down list.
2. On the **Action** tab, specify the following:

   - **Code** — Specify one or more lines of code in the scripting language specified for the transformation. For rules about expressions in this code, see Syntax Rules.

     InterSystems IRIS automatically wraps your code within a `CDATA` block. This means that you do not have to escape special XML characters such as the apostrophe (') or the ampersand (&),

     Also see the notes below.

   - **Description** — Specify an optional description.

Tip: To write custom code that you can debug easily, write the code within a class method or a routine so that it can be executed in the Terminal. Debug the code there. Then call the method or routine from within the code action of the DTL.

5.5.1 Guidelines for Using Custom Code in DTL

In order to ensure that execution of a data transformation can be suspended and restored, you should follow these guidelines when using a code action:

- The execution time should be short; custom code should not tie up the general execution of the data transformation.
- Do not allocate any system resources (such as taking out locks or opening devices) without releasing them within the same code action.
- If a code action starts a transaction, make sure that the same action ends the transactions in *all possible scenarios*; otherwise, the transaction can be left open indefinitely. This could prevent other processing or can cause significant downtime.

5.6 Adding an SQL Action

An **SQL** action enables you to execute an SQL SELECT statement from within the DTL transformation. InterSystems IRIS does not represent an **sql** action in the DTL diagram.

To add an **sql** action:

1. Select **sql** from the **Add Action** drop-down list.
2. On the **Action** tab, specify the following:
   - **SQL** — Specify a valid SQL SELECT statement.

     InterSystems IRIS automatically wraps your SQL within a `CDATA` block. This means that you do not have to escape special XML characters such as the apostrophe (') or the ampersand (&).

     Also see the notes below.
   - **Description** — Specify an optional description.

5.6.1 Guidelines for Using SQL in DTL

Be sure to use the following guidelines:

- Always use the fully qualified name of the table, including both the SQL schema name and table name, as in:

  `MyApp.PatientTable`

  Where `MyApp` is the SQL schema name and `PatientTable` is the table name.
- Any tables listed in the FROM clause must either be stored within the local InterSystems IRIS database or linked to an external relational database using the SQL Gateway.
- Within the INTO and WHERE clauses of the SQL query, you can refer to a property of the source or target object. To do so, place a colon (`:`) in front of the property name. For example:

  ```
  SELECT Name INTO :target.Name FROM MainFrame.EmployeeRecord WHERE SSN = :source.SSN AND City = :source.Home.City
  ```
- Only the first row returned by the query will be used. Make sure that the WHERE clause correctly specifies the desired row.

5.7 Adding a Switch Action

A **switch** action contains a sequence of one or more **case** actions and a **default** action. When a **switch** action is executed, it begins evaluating each **case** condition. When an expression evaluates to true, then the contents of the corresponding **case** block are executed; otherwise, the expression for the next **case** action is evaluated. As soon as one of the **case** actions is executed, the execution path of the transformation leaves the **switch** block without evaluating any other conditions. If no **case** condition is true, the contents of the **default** action are executed and then control leaves the **switch** block.

5.8 Adding a Case Action

Use the **case** action within a **switch** block to execute a block of actions when a condition is matched. When a **case** condition is met and the block of actions performed, the execution path of the transformation leaves the **switch** block without evaluating any other conditions.

To add a **case** action:

1. Select a **switch** action in the list below the diagram.
2. Select **case** from the **Add Action** drop-down list.
3. On the **Action** tab, add the condition. You can click the magnifying glass to add a function as part of the condition.
4. With the **case** action selected in the list below the diagram, use the **Add Action** drop-down to add the actions that will be executed if the condition evaluates to true.

5.9 Adding a Default Action

You cannot add a **default** block by using the **Add Action** drop-down list. Rather, the **default** action is automatically added to a **switch** block when you add the **switch** action. The actions contained in the **default** block are executed if none of the **case** conditions in the **switch** block are met. If you do not want anything to happen when none of the **case** conditions are met, simply leave the **default** block empty.

5.10 Adding a Break Action

Add a **break** action to a **for each** loop to leave the loop as soon as the **break** action is executed. After the **break** action is executed, the data transformation continues to process the action immediately following the **for each** loop.

If you add a **break** action outside of a **for each** loop, the data transformation terminates as soon as the **break** action is executed.

5.11 Adding a Comment Action

To help annotate the actions in a data transformation, you can add a comment that appears in the list of actions. After selecting **Add Action > Comment**, enter the comment in the **Description** text box on the **Action** tab.

6

Testing Data Transformations

After you compile a data transformation class, you can (and should) test it. This topic describes how to do so.

Note: This topic applies to both DTL transformations and custom transformations.

6.1 Using the Transformation Testing Page

The Management Portal provides the **Test Transform** wizard. You can access this from the following locations in the Management Portal:

- Click **Test** from the **Tools** tab in the Data Transformation Builder
- Select the transformation and click **Test** on the Data Transformation List page.

Initially the **Output Message** window is blank and the **Input Message** window contains a text skeleton in a format appropriate to the source message. To test:

1. If your DTL code references the properties of the aux, context, or process systems objects, enter values for these properties to see the results as if the data transformation was invoked with these objects instantiated. The table for entering values appears only if the DTL references the internal properties of aux, process, or context systems objects.
2. Edit the **Input Message** so that it contains appropriate data. What displays and what you enter in the input box depends on your source type and class:
    - For EDI messages, the window displays raw text; have some saved text files ready so that you can copy and paste text from these files into the **Input Message** box.
    - For regular production messages, the window displays an XML skeleton with an entry for each of the properties in the message object; type in a value for each property.
    - For record maps, complex record maps, and batch record maps, you can enter raw text or XML.
3. Click **Test**.
4. Review the results in the **Output Message** box.

The following shows an example:

6.2 Testing a Transformation Programmatically

To test a transformation programmatically, do the following in the Terminal (or write a routine or class method that contains these steps):

1. Create an instance of the source message class.
2. Set properties of that instance.
3. Invoke the **Transform()** class method of your transformation class. This method has the following signature:

   ```
   classmethod Transform(source As %RegisteredObject, ByRef target As %RegisteredObject) as %Status
   ```

   Where:

   - *source* is the source message.
   - *target* is the target message created by the transformation.

4. Examine the target message and see if it has been transformed as wanted. For an easy way to examine both messages in XML format, do the following:

   a. Create an instance of %XML.Writer.

   b. Optionally set the Indent property of that instance equal to 1.

      This adds line breaks to the output.

   c. Call the **RootObject()** method of the writer instance, passing the source message as the argument.

   d. Kill the writer instance.

   e. Repeat with the target message.

For example:

ObjectScript

```
//create an instance of the source message
set source=##class(DTLTest.Message).CreateOne()
set writer=##class(%XML.Writer).%New()
set writer.Indent=1 do writer.RootObject(source)
write !!
set sc=##class(DTLTest.Xform1).Transform(source,.target)
if $$$ISERR(sc)
 {do $system.Status.DisplayError(sc)}
set writer=##class(%XML.Writer).%New()
set writer.Indent=1
do writer.RootObject(target)
```