Merge HL7 Toolkit™
V. 5.8.0

SEGMENT MAP USER INTERFACE (UI)
SAMPLE APPLICATION GUIDE

Merge Healthcare Incorporated
900 Walnut Ridge Drive
Hartland WI 53029
USA
877.44.MERGE

The symbols glossary is provided electronically at http://www.merge.com/Support/Resources.aspx.

Manufacturer’s Address
Merge Healthcare Incorporated
900 Walnut Ridge Drive
Hartland, WI 53029

For assistance, please contact Merge Healthcare Customer Support:

- In North America, call toll free 1-800-668-7990, then select option 2
- International, call Merge Healthcare (in Canada) +1-905-672-7990, then select option 2
- Email MDTsupport@merge.com or MC3Support@ca.ibm.com

Part	Date	Revision	Description
COM-3443	December 2018	1.0	Updated bi-annually

Contents

Introduction

The Segment Map User Interface (UI) Sample Application is a graphical tool that allows users to quickly create mapping specifications between HL7 messages.

The mapping specification can be created as scripts following a specific grammar and saved to a text file. The main toolkit provides interfaces to load this script file and perform actual mapping from an input HL7 message instance to an output HL7 message instance.

This guide provides the information required to understand the functionality of this tool. It is intended to be used by service engineers who want to:

- create a Segment Map script to normalize all the received messages at a specific site before next operation
- process HL7 Messages stored on media and save them.

Files

This tool supports loading HL7 messages and segment map scripts from files. Both operations are optional. If no file is loaded, a segment map script can still be created; however, if there is a message instance loaded from the file, the operations are not abstract any more. Immediate mapping effects will be displayed and provide a more clear view for the user.

Message File

Input Files

This tool supports loading one and only one HL7 encoded message from a file. If there are multiple messages stored in the file, only the first one can be read successfully.

The start of the message is determined by the presence of an MSH segment, and the end of a message is reached if it meets either of the following conditions:

- The end of the file.
- The end of a line with presence of an MSH segment in the next line.

A line is defined as a sequence of characters followed by a carriage return ("\r"), a line feed ("\n") or a carriage return immediately followed by a line feed ("\r\n").

Note: Be careful when editing message files using text editors because they may “secretly” add some control characters that will interfere with the parsing of the message.

Script File

This tool supports loading pre-exist segment maps from a script file following a certain format. Usually, each line of the file represents a mapping specification

except ending with the "\" sign which is a concatenating sign indicating a statement across multiple lines.

If a line starts with the "#" sign, this mapping specification is commented out and doesn't have any effect.

Conventions

Throughout this guide, the following conventions and visual cues are used:

Words shown in large, boldface text, such as **Exit**, indicate application functions that you can click with the mouse (e.g., buttons, menu items).

Words shown in upper case, such as FIRST, indicate segment map statement keywords.

Sample Margin Note

Margin notes (in the left margin) are used to highlight important points or sections of the document.

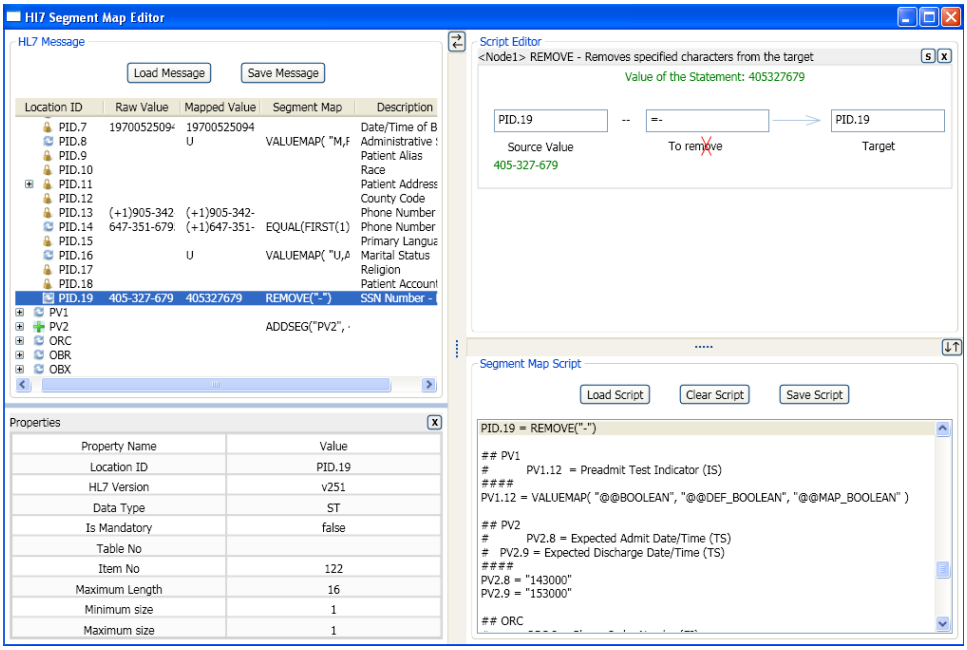
Note: Notes are used to indicate information which may be helpful or of special interest to the reader.

The Map Window

The main Segment Map UI consists of four interconnected views as can be seen from the image below:

- Message display window
- Properties window
- Script display window
- Script editing window

The Main Window



Of the views above, the message display window together with properties window makes up the message detail window and the properties window is closable; the script editing window together with the script display window make up the script detail window.

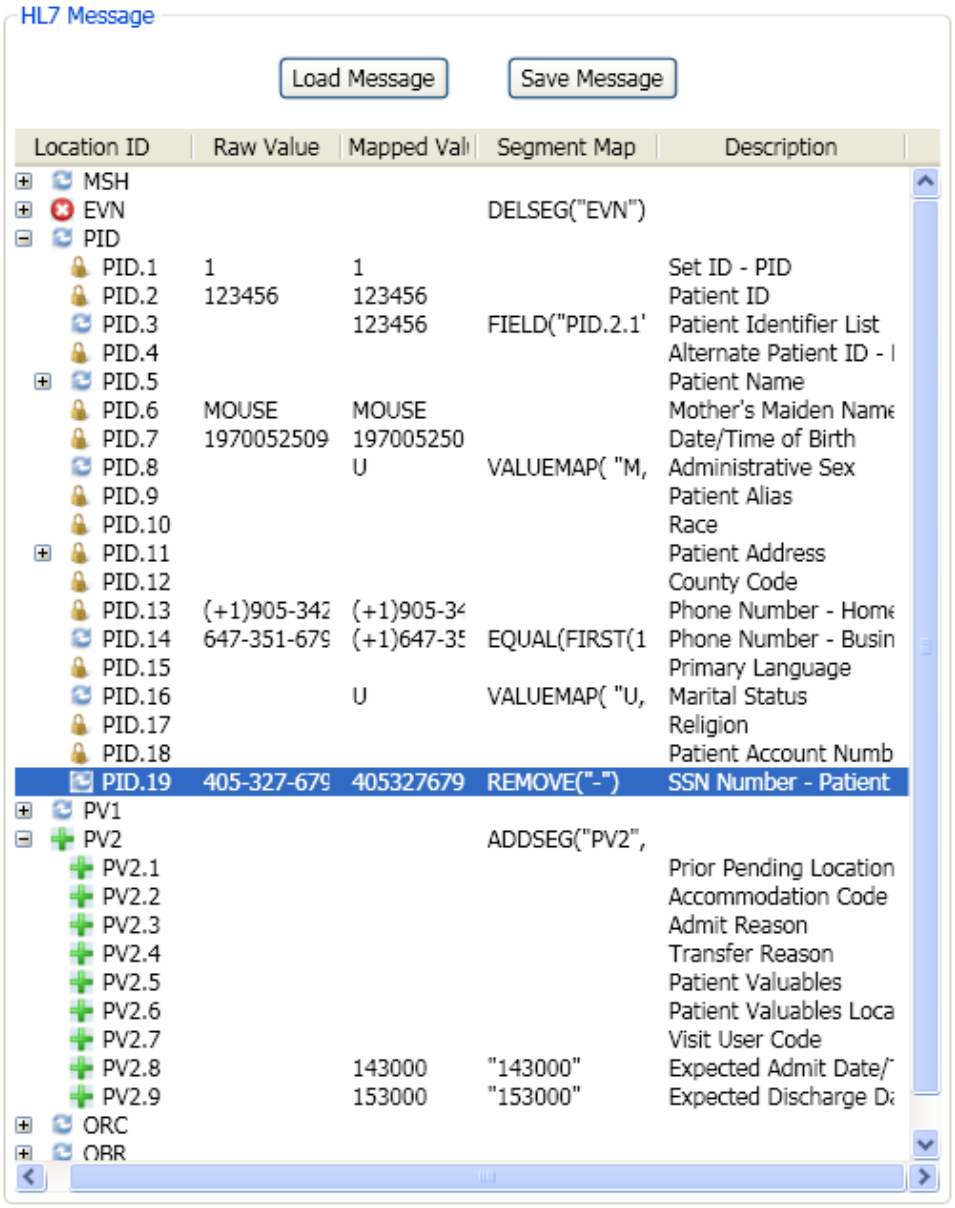
The spaces of the above views can be redistributed by moving the splitter cursor between them. Some views can also be relocated under a limit of range. For example, the message detail and script detail can be swapped by click the button between them, so does the script display pane and the script editing pane by clicking the button.

These views are fully described in the following section.

Message Display Window

The message display window is used to display both the raw HL7 message and the mapped one. Therefore it's a comparative structured list view with 4 additional columns: the raw value, the mapped value, the mapping script, and the field/component/subcomponent description besides the location identifier of each message element.

Message Display Window



Since it is a comparative view, to display the value change of each message element before and after mapping, four icons are used.

- The red x icon indicates that the segment this message element belongs to is deleted.
- The green plus icon indicates that the segment this message element belongs to is added.
- The yellow lock icon indicates that all the sub-elements of this message element remain unchanged during the mapping.

- The blue refresh icon indicates that the value of this message element is changed during the mapping.

A Message element is uniquely identified by its location in the message following a certain format:

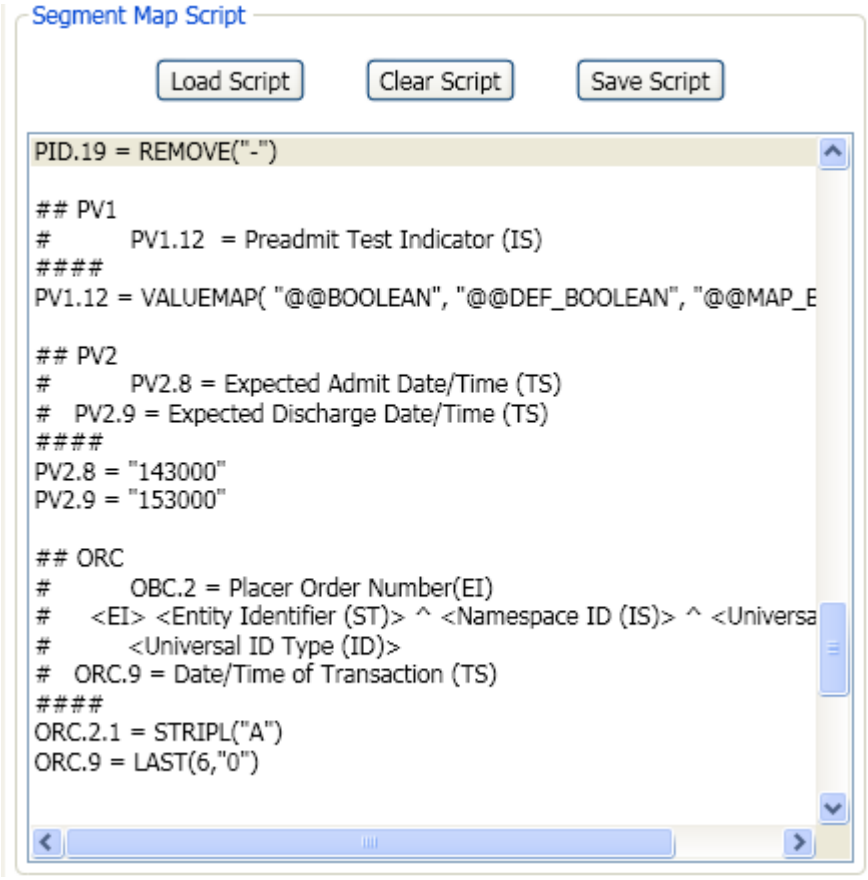
`"SegmentId.X.Y.Z'.`

Of which, the `SegmentId` could be the segment name if this message contains only one of this segment type or a segment name with index if this is a repeated segment in the message; `X`, `Y`, `Z` indicates the sequence number of the element in each layer of container element. For example, the only MSH segment in a message is identified as `MSH`; the family name component of the patient name field of the only PID segment is `PID.5.1`; the set ID field of the second OBR segment in the message is `OBR[1].1`.

Note: To avoid redundancy, the value of a message element is displayed only if it's atomic which means that it doesn't contain any sub items. For example, the value column of field `PID.5` is shown as empty because it has two components `PID.5.1` and `PID.5.2`.

If the script editing view is empty, the script display window will be synchronized with the message display view. As showed below, when browsing the `PID.19` message element, related segment map will be highlighted and scrolled into view.

Script Display Window



Properties Window

When browsing the message, the properties of each selected element will be displayed in the properties window if it's set to be visible. This window shares spaces with the message display window and set to be just under it. It can be closed clicking the [x] button on the top right. To make it visible, right click on a message element and select **Properties** from the context menu.

Properties Window

Properties

Property Name	Value
Location ID	PID.5
HL7 Version	v251
Data Type	XPN
Is Mandatory	true
Table No	
Item No	108
Maximum Length	250
Minimum size	1
Maximum size	14

Script Editing Window

This window is used to edit segment map graphically. As shown below, each statement is encapsulated in a rectangle box and described using natural language. To make the denotation convenient, the rectangle box is called a node and identified with the string Node plus an index number which is automatically assigned by the system and displayed in the header of the rectangle box. The nested statements are referenced in its parent statement by its node ID.

Text boxes or radio buttons in each node represent the parameters of the statement and need to be filled in or selected by the user to complete the mapping specification.

For example:

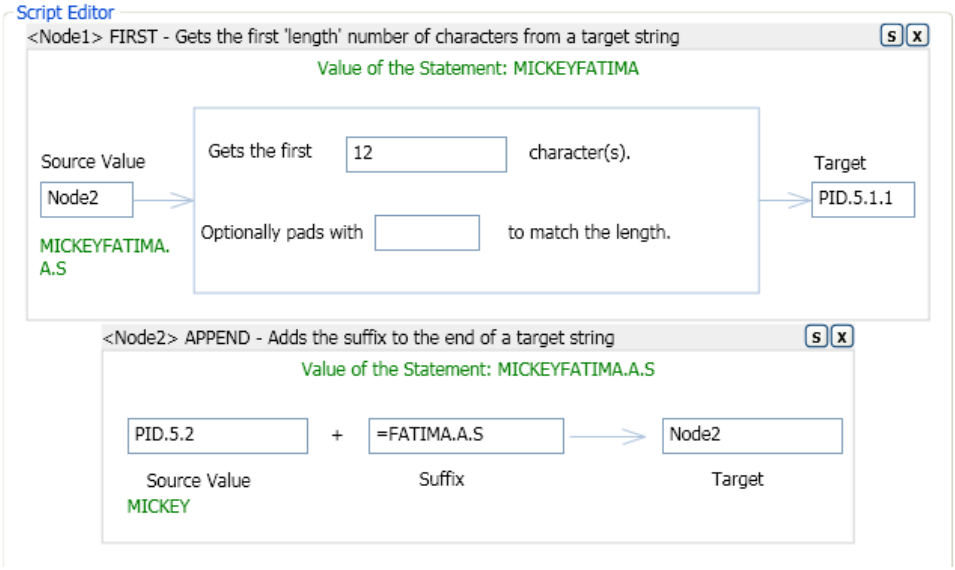
```
PID.5.1.1 = FIRST(APPEND(FIELD("PID.5.2"),"FATIMA.A.S"),12)
```

This segment map contains two atomic statements which are identified as Node1 and Node2.

Node2 manipulates the component `PID.5.2` and appends a string `FATIMA.A.S` to the end of it.

Node1 manipulates Node" and assign the first 12 characters of it to the subcomponent `PID.5.1.1`.

Script Editing Window



If a parameter accepts a node ID or field ID, the blanks can be filled in by dragging it from the message display window or from a nested statement box. For example, the "Source Value" of Node1 is "Node2", you can drag it from the Node2 box; the "Target" of Node1 is "PID.5.1.1", you can drag this field from the Message display window.

Usually, the last parameter of the field manipulation functions is the destination of the statement which could be a field ID or a node ID. If it's a field ID, this is a root statement from which a mapping specification can be generated and saved to the

script display window by clicking the S button; and if it's a node ID which must be equal to the ID of this statement, this statement is a nested one and can only be used as parameters by the other statements. The button X is used to remove a statement from this window.

This UI provides a dynamic view for the user. The values of the input field and the whole statement change immediately in accordance with the context which consists of the message loaded, the script loaded, and the parameters.

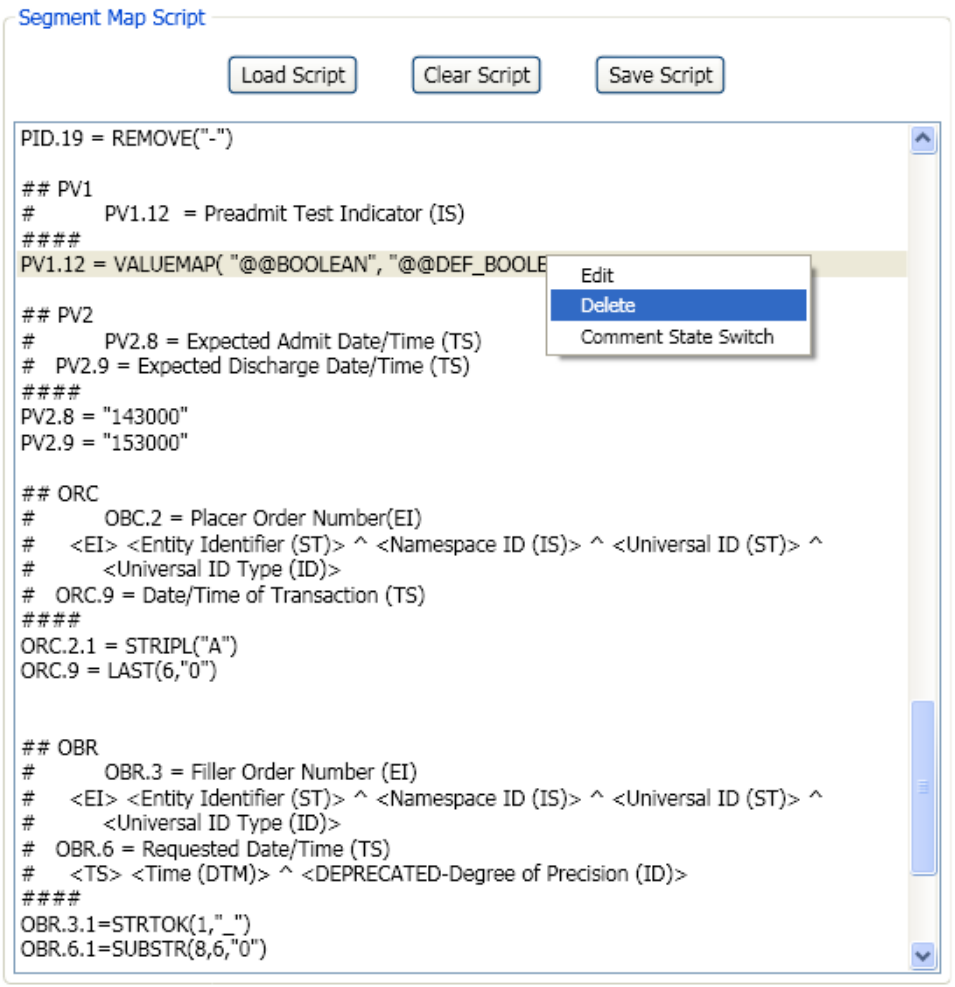
Script Display Window

This window is used to display the map specifications in the style of script which is the only acceptable format by the map engine.

For each segment map, there are three operations supported as shown below.

- **Edit** — for displaying this segment map graphically in the script editing window. Any change in that window can be saved to update this map specification.
- **Delete** — used to remove the selected segment map from the script.
- **Comment State Switch** — for changing the comment state of this segment map by removing the comment tag from the beginning of this line if it has one or adding a comment tag to the beginning of this line if it doesn't.

Script Display Window



This window also contains three buttons. **Load Script** is used to load a script file; **Clear Script** will remove all the segment maps from the window; **Save Script** can be used to save all the contents in this window to a file.

Segment Maps

The segment map specifications supported can be divided into three categories:

- Macro definition
- Segment operations (add or delete segments)
- Field manipulations (sub string, append, replace etc…)

This UI removes the efforts to understand the grammar of the segment map script language by creating a graphical wizard for each type of statement where the syntax is described using natural language. The users just need to follow the

wizard and supply several parameters and click a button to generate an error-free segment map.

Hl7 Parsing Script

This section gives detailed description of the HL7 parsing script language.

Note: Parameter of type Stmt could be any of the field manipulation statements below or a string. Parameters within square brackets are optional.

Macro Definition

Syntax:

```
@@MACRONAME = MACROVALUE
```

The macro is identified by starting "@@" sign.

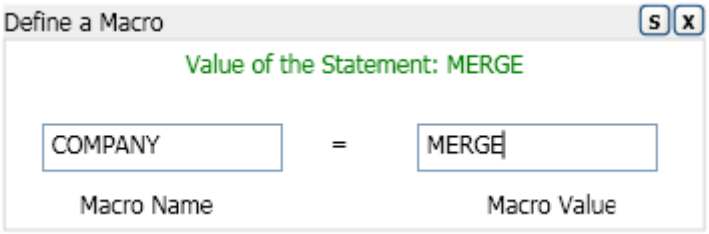
Description:

Define a Macro.

Example:

```
@@COMPANY = MERGE
```

Graphical Representation:



ADDSEG

Syntax:

```
ADDSEG("segmentType",messsageIndex)
```

Description:

This command allows you to add a new segment at the specified place.

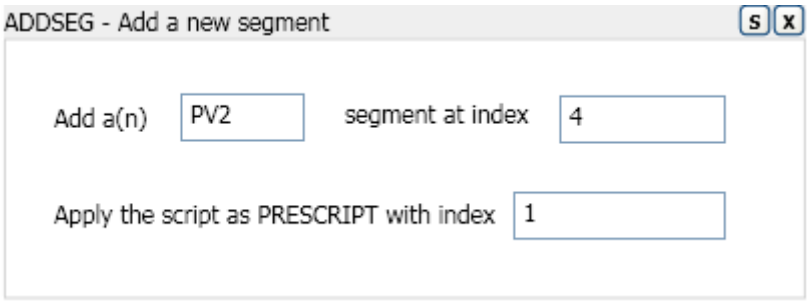
Adding segments is confined to be PRESCRIPT which indicates that this command happens before any field manipulation. The index number of the PRESCRIPT specifies the execution order of it. The smaller is the number, the more advanced that script is applied.

Because adding segments will change the index numbers of the segments that follow the added segment, you should add segments in reverse order (i.e. Highest-numbered segments first).

Example:

```
PRESCRIPT1 = ADDSEG("PV2", 4)
```

Graphical Representation:



DELSEG

Syntax:

```
DELSEG("segmentId")

DELSEG(messsageIndex)

DELSEG(messsageIndex1,messsageIndex2)
```

Description:

This command allows you to delete (a) segment(s) in any of the following four modes.

1. Specify a segment type. This will delete all instances of the specified segment type. For example, `DELSEG("EVN")`
2. Specify a segment type and an index number. This will delete the specified instance of the specified segment type. For example, `DELSEG ("OBX [0]")` will delete the first OBX segment in the message.
3. Specify a 0-based index number. This will delete the specified segment. For example, `DELSEG(1)` will delete the second segment in the message.
4. Specify a range of index numbers. This will delete all segments inclusive of the index numbers. For example, `DELSEG(2,4)` will delete the 3rd, 4th, 5th segments in the message.

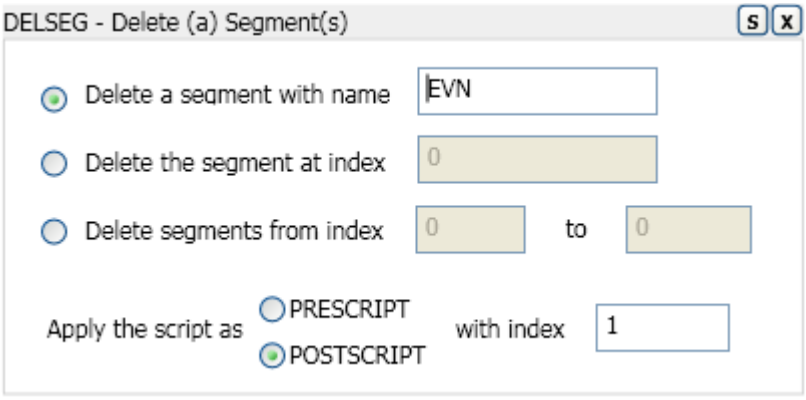
`DELSEG` can be used as a PRESCRIPT or a POSTSCRIPT which indicates that this command happens after all the field manipulation. The index number of the PRESCRIPT or POSTSCRIPT specifies the execution order of it. The smaller is the number, the more advanced that script is applied.

Because deleting segments will change the index numbers of the segments that follow the deleted segment, you should delete segments in reverse order (i.e. Highest-numbered segments first).

Example:

```
POSTSCRIPT1=DELSEG ("EVN")
```

Graphical Representation:



FIELD

Syntax:

```
FIELD (String fieldLocation)
```

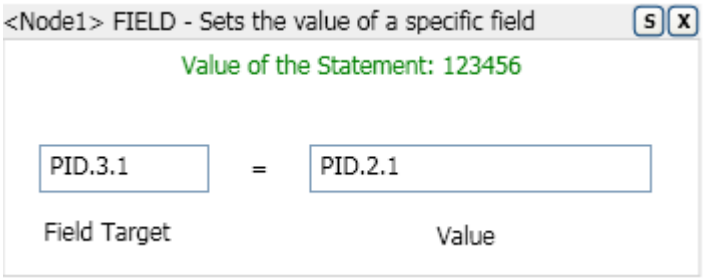
Description:

Copy the value of a specified field.

Example:

```
PID.3.1 = FIELD ("PID.2.1")
```

Graphical Representation:



EQUAL

Syntax:

```
EQUAL(Stmt() a, Stmt() b, Stmt() c [,Stmt() d])
```

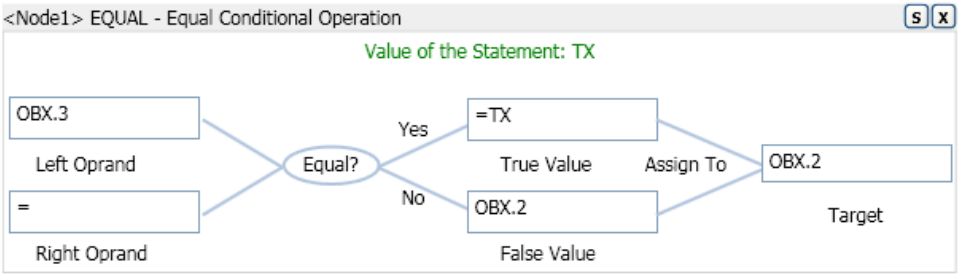
Description:

This function compares the results of statement “a” and statement “b”. If the comparison is true, then the result of statement “c” is returned. If the comparison is false, then the result of statement “d” is returned, unless the optional statement “d” does not exist. In this case, the original field will remain unchanged.

Example:

```
OBX.2=EQUAL(FIELD("OBX.3"), "", "TX")
```

Graphical Representation:



FIRST

Syntax:

```
FIRST([Stmt() value,] Integer length, [String padding])
```

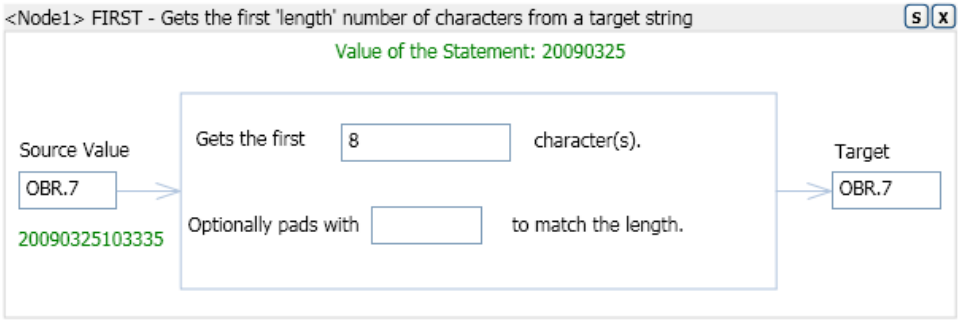
Description:

This function returns the first number (indicated by `length`) of characters of the target field, unless the optional `value` is included. If `value` is provided, then the first number of characters from `value` is returned. Padding is also an optional parameter. If the length of the target field or `value` (if it is included) is less than the `integer length`, it is padded with the `string padding`.

Example:

```
OBR.7=FIRST(8)
```

Graphical Representation:



LAST

Syntax:

```
LAST([Stmt() value,] Integer length, [String padding])
```

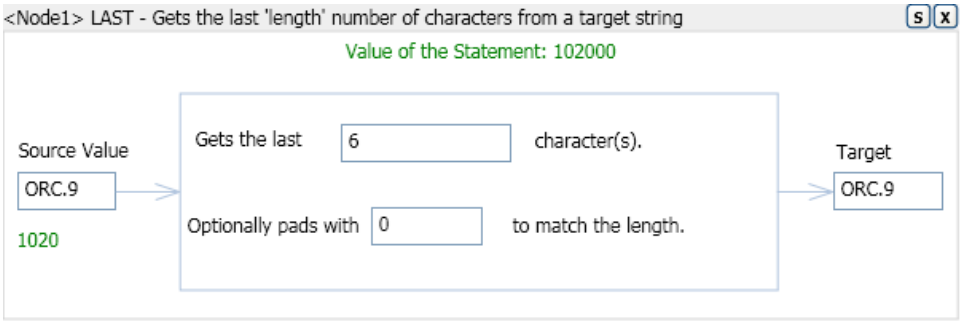
Description:

This function returns the last number (indicated by `length`) of characters of the target field, unless the optional `value` is included. If 'value' is provided, then the last number of characters from `value` is returned. Padding is also an optional parameter. If the length of the target field or `value` (if it is included) is less than the `integer length`, it is padded with the `string padding`.

Example:

```
ORC.9 = LAST(6,"0")
```

Graphical Representation:



PREPEND

Syntax:

```
PREPEND([Stmt () value,] Stmt() prefix)
```

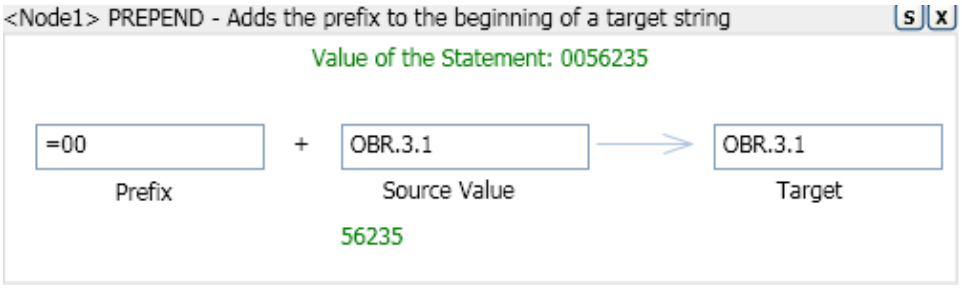
Description:

This function returns the concatenation of `prefix` and the target field. If `value` is provided, then the function returns the concatenation of `prefix` and `value` instead.

Example:

```
OBR.3.1=PREPEND("00")
```

Graphical Representation:



APPEND

Syntax:

```
APPEND([Stmt () value,] Stmt() suffix)
```

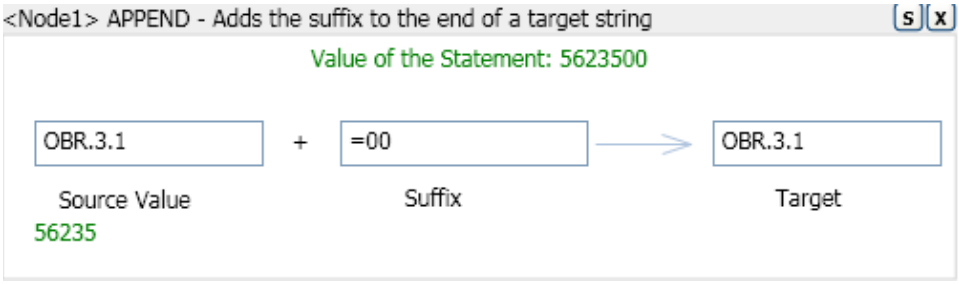
Description:

This function returns the concatenation of the target field and `suffix`. If `value` is provided, then the function returns the concatenation of `value` and `suffix` instead.

Example:

```
OBR.3.1=APPEND(“00”)
```

Graphical Representation:



REPLACE

Syntax:

```
REPLACE([Stmt () value,] Stmt () original, Stmt () new)
```

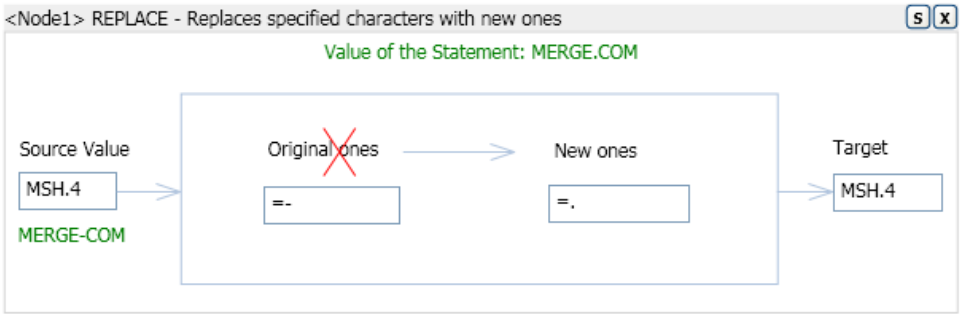
Description:

This function replaces all occurrences of `original` with new in the target field or `value`, if it is provided. The original string is a regular expression, which means that you should avoid using any of the following special characters: [\^$.|?*+()

Example:

```
MSH.4 = REPLACE("-",".")
```

Graphical Representation:



REMOVE

Syntax:

```
REMOVE([Stmt () value,] Stmt () toRemove)
```

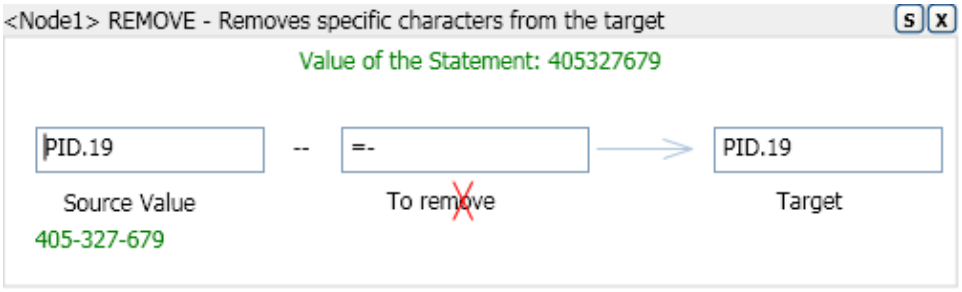
Description:

This function removes all occurrences of `toRemove` from the target field or from `value` if provided.

Example:

```
PID.19 = REMOVE("-")
```

Graphical Representation:



STRIPL

Syntax:

```
STRIPL([Stmt () value,] Stmt () lead)
```

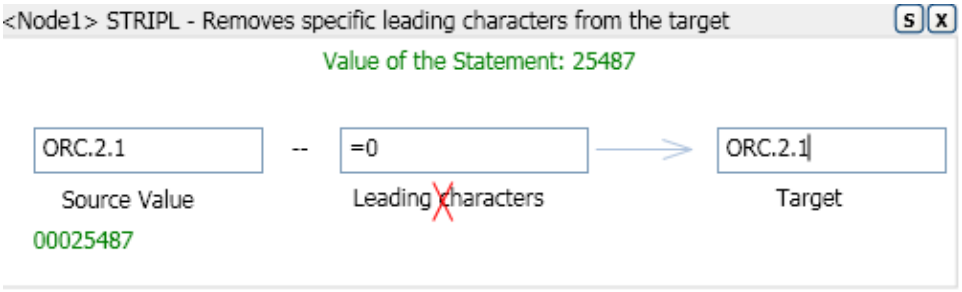
Description:

This function removes the leading occurrences of `lead` from the target field or `value` if provided.

Example:

```
ORC.2.1 = STRIPL("0")
```

Graphical Representation:



STRIPT

Syntax:

```
STRIPT([Stmt () value,] Stmt () trail)
```

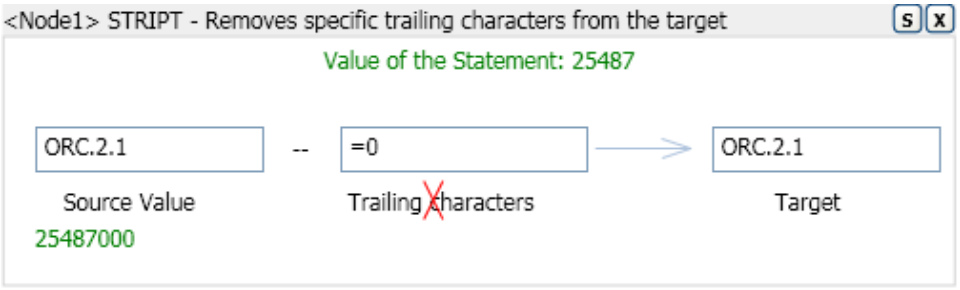
Description:

This function removes the trailing occurrences of `trail` from the target field or `value` if provided.

Example:

```
ORC.2.1 = STRIPT("0")
```

Graphical Representation:



SUBSTR

Syntax:

```
SUBSTR([Stmt () value,] Integer offset, Integer length [,Stmt () pad])
```

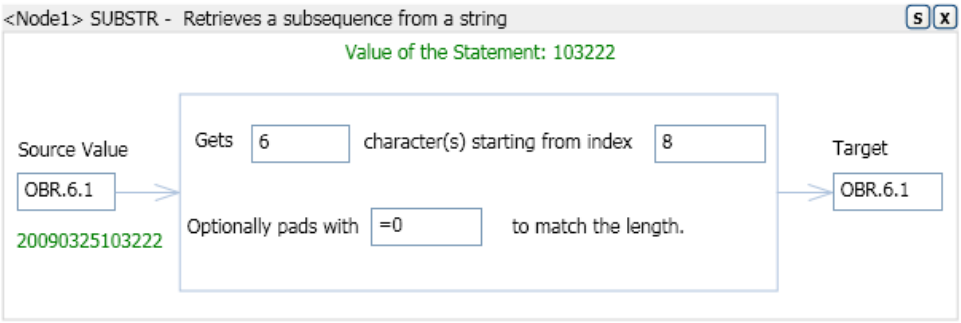
Description:

This function returns `length` characters, starting at `offset`, of the target field or `value`, appending the `pad` string to the string, so that the returned string is always `length` characters long

Example:

```
OBR.6.1=SUBSTR(8,6,"0")
```

Graphical Representation:



STRTOK

Syntax:

```
STRTOK([Stmt () value,] Integer index [,String sep])
```

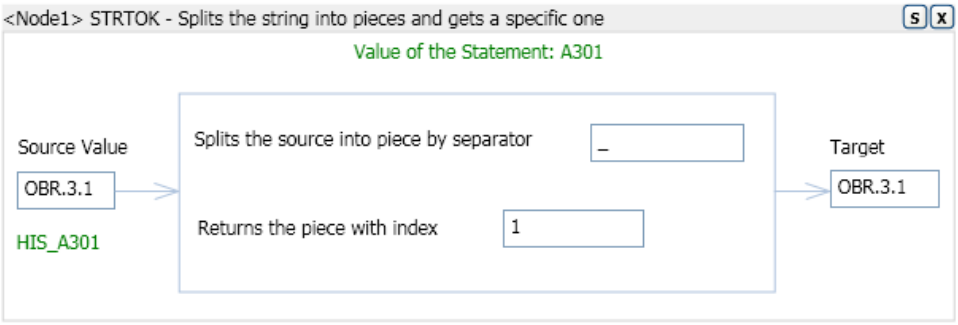
Description:

This function takes the target string or `value`, which is separated into indexed sections by the defined `sep` (separator) character, and returns the value for the section indicated by the index value. The index starts at 0.

Example:

```
OBR.3.1=STRTOK(1,"_")
```

Graphical Representation:



VALUEMAP

Syntax:

```
VALUEMAP(String acceptableValues, String defaultValue [,String
valueMappings])
```

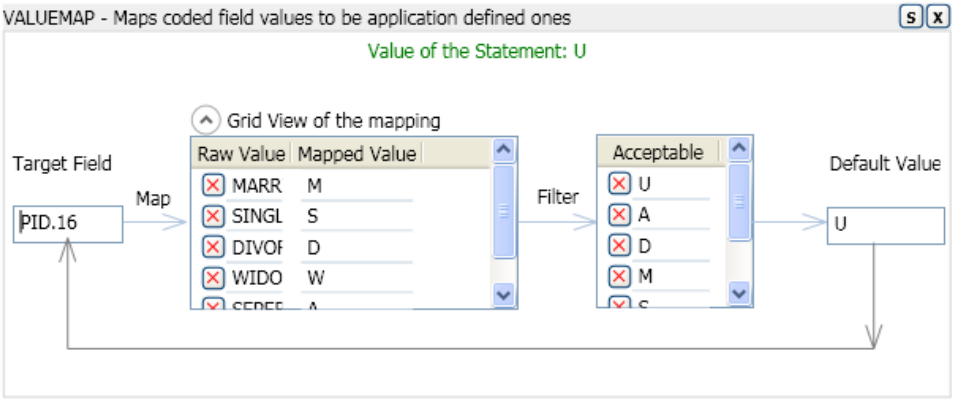
Description:

This function maps coded field values to application defined values. `acceptableValues` is the comma separated list of mapped output values. `defaultValue` is the value to be used in case a mapping cannot be found. `valueMappings` is a comma separated list of raw/mapped value pairs in the form `rawValue:mappedValue`.

Example:

```
PID.16 = VALUEMAP( "U,A,D,M,S,W", "U",
"MARRIED:M,SINGLE:S,DIVORCED:D,WIDOWED:W,SEPERATED:A" )
```

Graphical Representation:



Adding a Segment Map

Adding a segment map in the script editing window

Use the following procedure to add a new segment map in the script editing window.

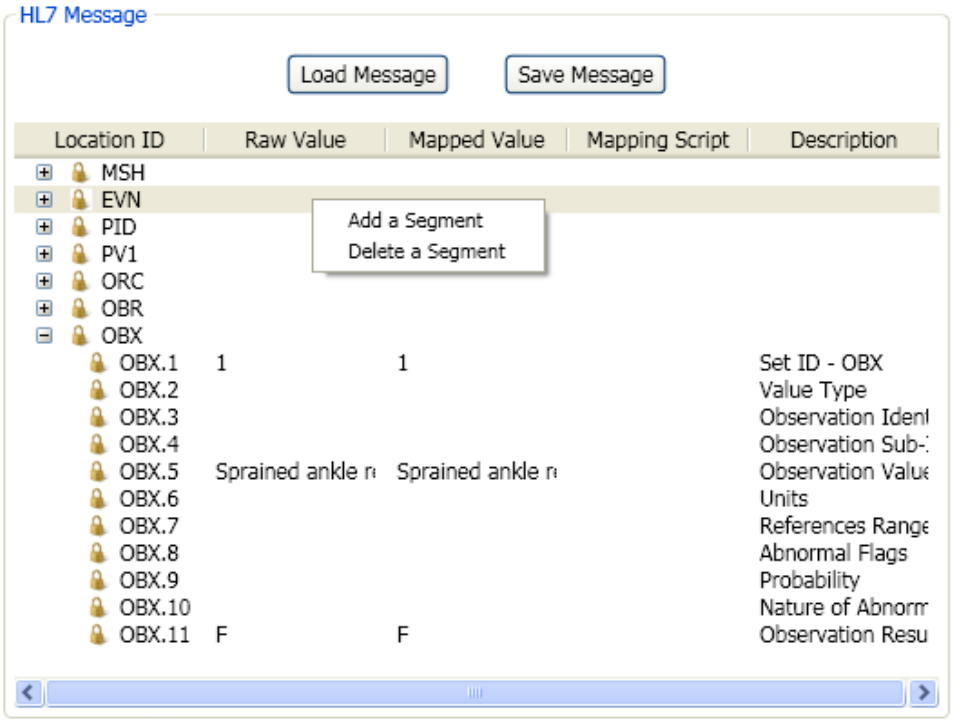
1. Right click in the script editing window.
2. Select **Clear Edit Area** from the popup menu to clear this area if there is any graphical element in this window.
3. Enter into the sub menu of **Add Mapping Statement** and select the statement type to be added. The statement generating wizard encapsulated in a rectangle box will appear in this window.
4. Supply the parameters for the statement following corresponding hints.
5. Repeat Step 3 and Step 4 to add nested statements.
6. Select the **Save Edit Area** menu item or click the S button in the root statement to save it as a script in the script display window.

Shortcuts of adding segment maps

There are shortcuts for adding several mostly used types of segment maps from the message display widow.

1. Copy a field to another one. When browsing the message tree list view, dragging one message element and dropping it at another one will generate a FIELD operation in the script editing window with all the parameters filled.
2. Add a segment. As shown in the image below. Right-click on the segment and a menu pops up. Select **Add a Segment** to add an ADDSEG statement in the script editing area.
3. Delete a segment. As shown in the image below. Right-click on the segment and a menu pops up. Select **Delete a Segment** to add a DELSEG statement in the script editing area.

In the graphical representation generated through shortcuts, the parameters are pre-filled according to the context that triggers the operation. These parameters may not be exactly what you want and you can modify them in the script editing window and then click the S button to save them.



Editing a Segment Map

Use the following procedure to edit a segment map.

1. Right-click on the segment map you want to edit in the script display window.
2. Select **Edit** from the popup menu and the segment map will be displayed graphically in the script editing window.
3. Modify the parameters of all the statements. Add or delete statements in the script edit window to get expected mapping specification.
4. Click the S button at the right top of the root statement to save the changes to the script display window.

Deleting a Segment Map

There are two kinds of delete operations:

- Remove the segment map from the script display window.
- Comment out the segment map.

Regarding the second operation, this segment map is still shown in the script display window with additional comment tag "#" at the beginning of it.

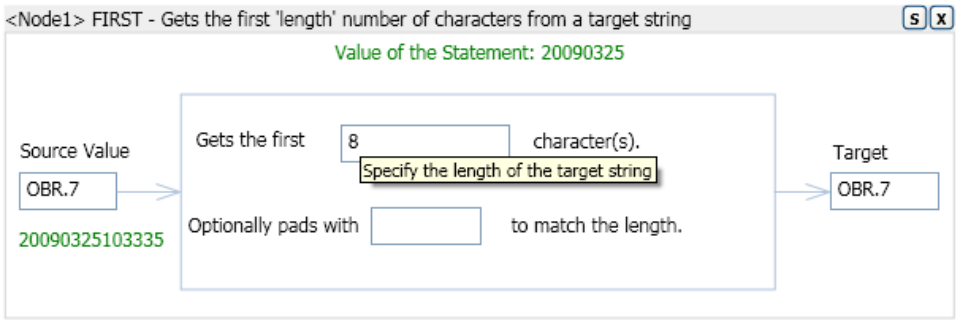
Use the following procedure to delete a Segment Map

1. Right-click on the segment map you want to delete in the script display window
2. Select **Delete** to remove it completely or **Comment State Switch** to comment it out.

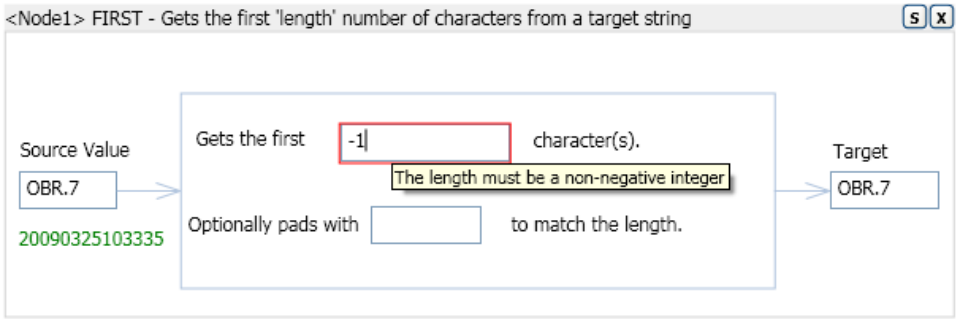
Input Hint and Error Checking

In the graphical representation of the statements described above, there are hints and basic error checking implemented for each parameters. When the cursor is on the text box, a hint will be displayed indicating acceptable input for it. And if the input by the user is unacceptable, the text box is bordered with red lines and the hint for it gives the reason why the validation fails.

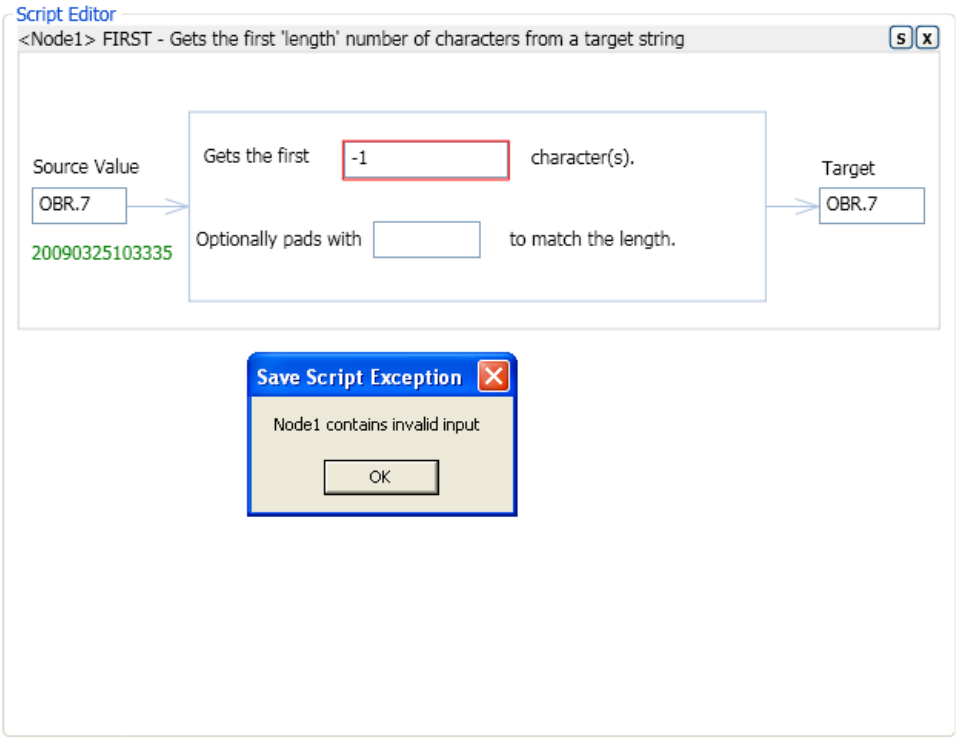
For example: in the FIRST statement, when the cursor is on the text box for the 'length' input. A message is displayed "Specify the length of the target string".



If the user input "-1" for this parameter, the text box will be in error state and the message changes to be "The length must be a non-negative integer".



Trying to save a node with invalid input will cause an error message box popped up.



Some errors are not checked until you try to save the script as shown in the image below. There is no statement identified by Node2, and trying to save this statement will cause an error “Unresolved Node ID: Node2”.

To create a correct statement, make sure to follow the hints for each input parameter or the error messages displayed.