**Using the String Utilities service.**

This chapter discusses the following:

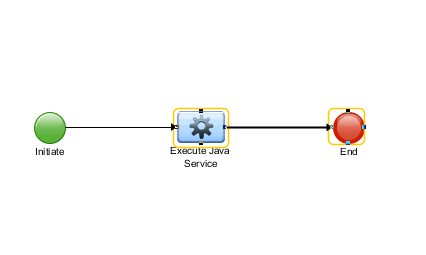
**• String Utilities**

**String Utilities**

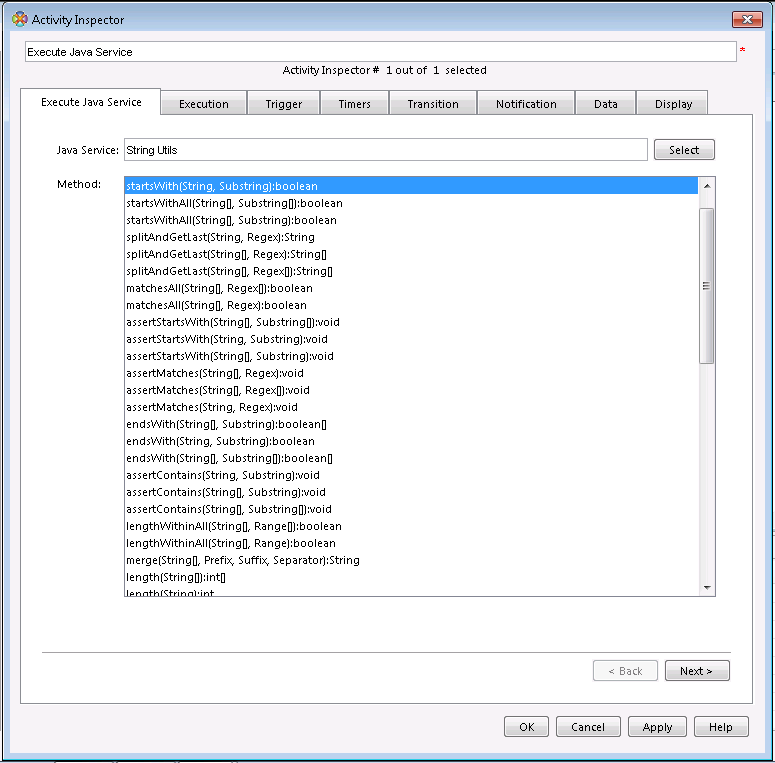
The StringUtils xCelerator provides a set of commonly used string operations, such as trim, substring, split, matches. This makes it possible for a developer to easily verify that a string conforms to a specific format, extract information from a string (such as a case number) and split a string into multiple substrings for further processing.

***Configuration***

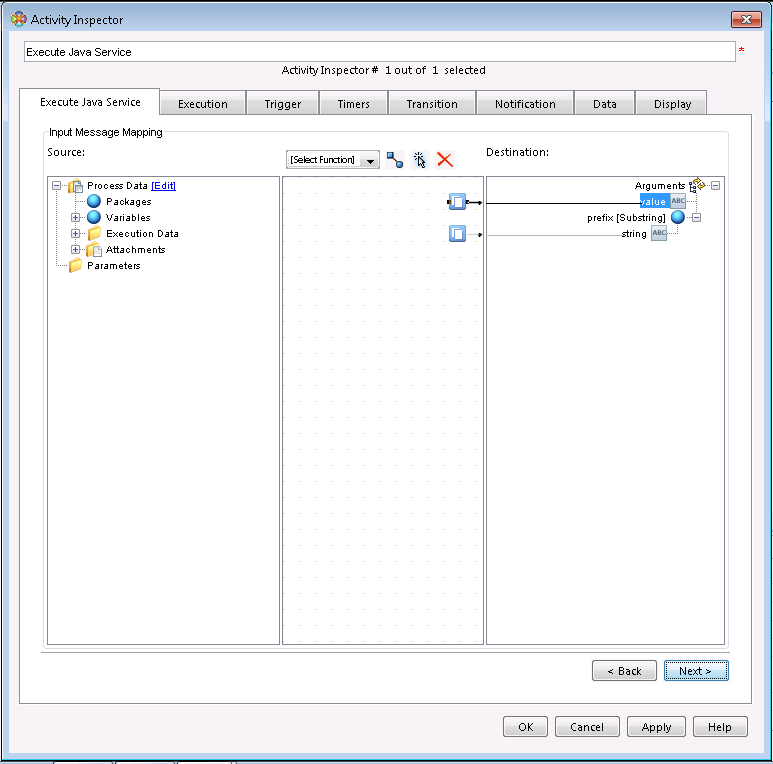
When using the String Utilities service within your process, you must select the service to be executed by the Java Execute Service activity. First, add the Java Execute Service to your workflow.



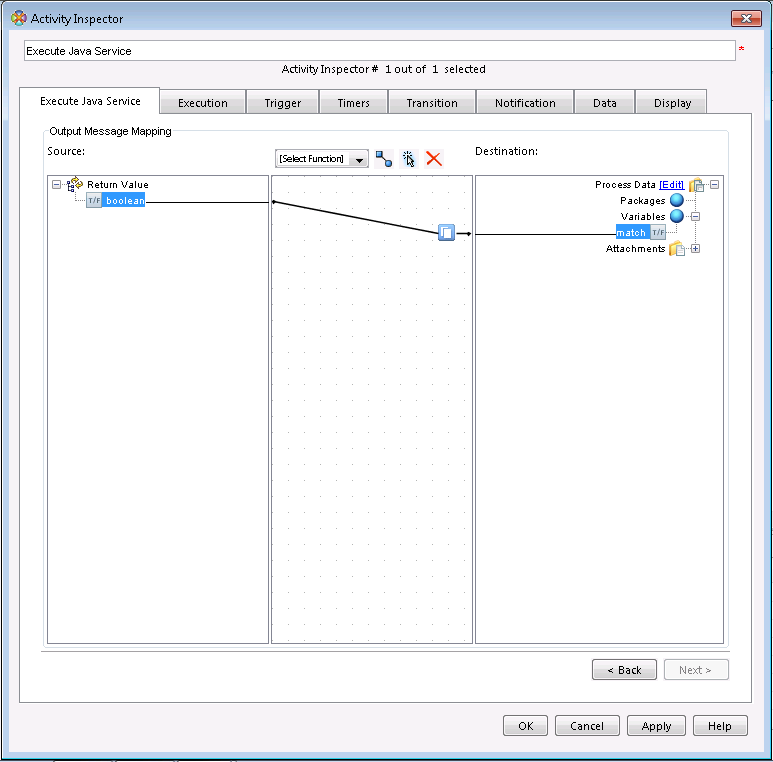
Double click and open the activity. Select the Java Service “String Utils”. Then select the function you wish to use.



The following example is with the startsWith function.



Map the Boolean result back into a value for storage.



***Overview of the Methods StringUtils Provides***

The StringUtils methods are grouped into the following groups:

• Assertions

• Contains

• EndsWith

• Format

• Length

• LengthWithin

• Matches

• Merge,

• Split

• StartsWith

• Substring

• Trim

Most of the methods exist in, at least 2 versions, one simple version and one, or more, aggregate or array versions. For example consider the contains methods.

1) boolean **contains**(String value, Substring substring);

2) boolean[] **contains**(String[] value, Substring substring);

3) boolean[] **contains**(String[] value, Substring[] substring);

1. The simple method, checks if value contains substring.

2. The first aggregate method, for each value check if they contain substring and returns an array of equal length to the value argument with the result of each contains check.

3. The second aggregate method, for each value check if they contain the substring at the corresponding index and returns an array of equal length to the value argument with the result of each contains check.

In addition for contains (and all methods that return a boolean value), there are two All methods which combines the result with short circuited logical and into one value (4, 5 below).

4) boolean **containsAll**(String[] value, Substring substring);

5) boolean **containsAll**(String[] value, Substring[] substring);

In general if a method takes multiple arrays they must be of equal length or an ArrayIndexOutOfBoundsException will be thrown that will halt your workflow.

***Assertions***

**Description**

The assertion methods verifies if the first parameter, the value, conforms to whatever value the function restricts it to in relation to the following parameters. If the value does not conform an exception is thrown and the workflow is halted. For example, the assertContains function checks if the substring is contained in the value, assertContains(“hamburger”,”ham”) would not halt the workflow while assertContains(“hamburger”,”bun”) would.

**Methods**

void **assertContains**(String value, Substring substring) throws DfException;

void **assertContains**(String value[], Substring substring) throws DfException;

void **assertContains**(String value[], Substring substring[]) throws DfException;

void **assertEndsWith**(String value, Substring suffix) throws DfException;

void **assertEndsWith**(String value[], Substring suffix) throws DfException;

void **assertEndsWith**(String value[], Substring suffix[]) throws DfException;

void **assertLengthWithin**(String value, Range range) throws DfException;

void **assertLengthWithin**(String value[], Range range) throws DfException;

void **assertLengthWithin**(String value[], Range[] range) throws DfException;

void **assertMatches**(String value, Regex pattern) throws DfException;

void **assertMatches**(String value[], Regex pattern) throws DfException;

void **assertMatches**(String value[], Regex pattern[]) throws DfException;

void **assertStartsWith**(String value, Substring prefix) throws DfException;

void **assertStartsWith**(String[] value, Substring prefix) throws DfException;

void **assertStartsWith**(String[] value, Substring[] prefix) throws DfException;

***Contains***

**Description**

The contains methods return boolean values indicating whether or not the first argument, the value(s), contains the following substring argument. For example contains(“hamburger”,”burg”) would return true while contains(“hamburger”,”bun”) would return false. The methods that end with All returns true if every string in the value argument contains the substring argument.

**Methods**

boolean **contains**(String value, Substring substring);

boolean[] **contains**(String[] value, Substring substring);

boolean[] **contains**(String[] value, Substring[] substring);

boolean **containsAll**(String[] value, Substring substring);

boolean **containsAll**(String[] value, Substring[] substring);

**EndsWith**

**Description**

The endsWith methods return boolean values indicating whether or not the first argument, the value(s), ends with the following substring argument. For example endsWith(“hamburger”,”burger”) would return true while endsWith(“hamburger”,”ham”) would return false. The methods that end with All returns true if every string in the value argument ends with the substring argument.

**Methods**

boolean **endsWith**(String value, Substring suffix);

boolean[] **endsWith**(String[] value, Substring suffix);

boolean[] **endsWith**(String[] value, Substring[] suffix);

boolean **endsWithAll**(String[] value, Substring suffix);

boolean **endsWithAll**(String[] value, Substring[] suffix);

**Format**

**Description**

The format method formats the arguments as specified by the msgFormat argument. This is implemented as new MessageFormat(msgFormat).format(arguments) so for details refer to java.text.MessageFormat. For example, format(“Hello {0}!”,new String[]{“John”}) would return the string “Hello John!”.

**Methods**

String **format**(String msgFormat, String[] arguments);

**Length**

**Description**

The length methods return integer values indicating the length of the first argument, the value(s).

For example length(“hello”) would return 5.

**Methods**

int **length**(String value);

int[] **length**(String[] values);

**LengthWithin**

**Description**

The lengthWithin methods return boolean values indicating whether or not the first argument, the value(s), are within (exclusive) the range. For example lengthWithin(“alfa”,new Range(0,10)); would return true while lengthWithin(“alfa”,new Range(4,10)) would return false.

**Methods**

boolean **lengthWithin**(String value, Range range);

boolean[] **lengthWithin**(String[] value, Range range);

boolean[] **lengthWithin**(String[] value, Range[] range);

boolean **lengthWithinAll**(String[] value, Range range);

boolean **lengthWithinAll**(String[] value, Range[] range);

**Matches**

**Description**

The matches methods return boolean values indicating whether or not the first argument, the value(s), matches the following regex argument. For example matches(“11 hamburgers”,new Regex(”.\*[0-9]\*.\*”)) would return true while matches(“hamburger”,new Regex(”.\*cheese.\*”) would return false. For details regarding the regular expression syntax refer to java.util.regex.Pattern.

**Methods**

boolean **matches**(String value, Regex pattern);

boolean[] **matches**(String[] value, Regex pattern);

boolean[] **matches**(String[] value, Regex[] pattern);

boolean **matchesAll**(String[] value, Regex pattern);

boolean **matchesAll**(String[] value, Regex[] pattern);

**Merge**

**Description**

The merge method merges an array of strings into one string, potentially with a prefix and suffix before and after all occurrences and a separator in between each occurrence. For example merge(new String[]{“a”,”b”,”c”}, new Prefix(“(“), new Suffix(“)”), new Separator(“,”)) would return the string “(a,b,c)”.

**Methods**

String **merge**(String[] values, Prefix before, Suffix after, Separator separator);

**Split**

**Description**

The split methods return an array of strings containing the individual parts of a string that was split. For example split(“a,b,c”,new Substring(“,”)) would return the array {“a”,”b”,”c”}. You can use both strings and regular expressions to do your split, for details regarding the syntax of the regular expression refer to the documentation for java.util.regex.Pattern. Note the splitAndGet\* convenience methods which in addition to splitting returns the first or last value of the parts resulting from the split.

**Methods**

String[] **split**(String value, Regex pattern);

String[] **split**(String value, Substring pattern);

String **splitAndGetFirst**(String value, Regex pattern);

String[] **splitAndGetFirst**(String value[], Regex pattern);

String[] **splitAndGetFirst**(String value[], Regex pattern[]);

String **splitAndGetLast**(String value, Regex pattern);

String[] **splitAndGetLast**(String value[], Regex pattern);

String[] **splitAndGetLast**(String value[], Regex pattern[]);

**StartsWith**

**Description**

The startsWith methods return boolean values indicating whether or not the first argument, the value(s), starts with the following substring argument. For example startsWith(“hamburger”,”ham”) would return true while startsWith(“hamburger”,”burger”) would return false. The methods that end with All returns true if every string in the value argument starts with the substring argument.

**Methods**

boolean **startsWith**(String value, Substring prefix);

boolean[] **startsWith**(String[] value, Substring prefix);

boolean[] **startsWith**(String[] value, Substring[] prefix);

boolean **startsWithAll**(String[] value, Substring prefix);

boolean **startsWithAll**(String[] value, Substring[] prefix);

**Substring**

**Description**

The substring methods return string that is a substring of the first argument, the value(s). The substring is indentified either by index (position) + length or by a regular expression group (by default the 1 group is used). For example substring(“hamburger”,2,2) would return the string “mb” while substring(“hamburger”,new Regex(“.\*m(.\*).\*”) would return the string “burger”.

**Methods**

String **substring**(String value, **int** index, **int** length);

String **substring**(String value, Regex pattern);

String[] **substring**(String value[], Regex pattern[]);

String **substring**(String value, Regex pattern, **int** group);

String[] **substring**(String value[], Regex pattern[], **int** group[]);

String[] **substring**(String[] value, **int** index, **int** length);

**Trim**

**Description**

The trim methods remove whitespace from the start and end of the string, for details refer to the java.lang.String#trim method. For example trim(“ hamburger ”) would return the string “hamburger”.

**Methods**

String **trim**(String value);

String[] **trim**(String[] value);