

This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. The article is believed to be still valid for the original topic scope.



This section contains descriptions of each available expression type.

## Contents

- [1 <item>](#)
- [2 <list>](#)
- [3 <str>](#)
- [4 <strcat>](#)
- [5 <substr>](#)
- [6 <var>](#)
- [7 <get>](#)
- [8 <foreach>](#)
- [9 <iterator>](#)
- [10 <regex>](#)
- [11](#)  
[<regex\\_match>](#)
- [12](#)  
[<regex\\_match\\_start>](#)
- [13](#)  
[<regex\\_match\\_end>](#)
  - ◆ [13.1](#)  
[<xpath>](#)
- [14 <choice>](#)
- [15 <to\\_date>](#)
- [16 <to\\_hash>](#)
- [17](#)  
[<entity\\_decode>](#)
- [18 <untag>](#)
- [19 <pipeline>](#)
- [20 <tee>](#)
- [21 Example](#)  
[filter](#)
- [22 See also](#)

## <item>

The item expression is used to produce named or nameless string or integer values that are sent to the client. An item expression always forms an implied tee branch.

*The value definition can be*

- empty,
- a constant string,

## Filter\_expressions\_reference

- a string containing state variable references, or
- one or more child expressions.

If no value is defined, then the item expression acts as a forward declared item expression. Another expression can later return a non-target result to the item expression that catches the result and sets it as the value of the item.

If there are more than one child expressions, the child expressions form an implied pipeline expression.

A non-target result returned from the child expressions is set as the value of the item. Any target result returned from the child expressions is returned to the parent expression.

### *The result of the item expression can be*

- null,
- a nameless item representing a nameless string or integer value, or
- a named item representing a named string or integer value.

### **The item expression can have the following optional attributes:**

name	Specifies the name of the item. Can be any expression producing a string.
id	Specifies the id of the item. Must be a constant string. Needed if the item has no constant value name, but you want to refer to the item from another expression.
to	Specifies the id or name of the expression (typically a list expression) to which the non-null result from this expression is returned.

## <list>

The list expression is used to produce named or nameless lists of items and lists that are to be returned to the client. A list expression always forms an implied tee branch.

### *The value definition can be*

- empty, or
- one or more child expressions.

If no value is defined, then the list expression acts as a forward declared list expression. Some other expressions can later return target results to the list expression that catches the results and adds them to the value list.

If there are more than one child expressions, the child expressions form an implied pipeline expression.

Any target result returned from the child expressions is added to the value list. Any non-target result returned from the child expressions is caught and ignored.

### *The result of the list expression can be*

- null, or

<item>

## Filter\_expressions\_reference

- a nameless list of items and lists, or
- a named list of items and lists.

**The *list expression* can have following optional attributes:**

name	Specifies the name of the list. Can be any expression producing a string.
id	Specifies the id of the list. Must be a constant string. Needed if the list has no constant value name, but you want to refer to the list from another expression.
to	Specifies the id or name of the expression (typically a list expression) to which a non-null result from this expression is returned.

### <str>

The str expression is used to produce strings when normal text values cannot be used due to XML syntax restrictions.

**The value definition can be**

- a constant string, or
- a string containing state variable references.

**The result of the str expression can be**

- null, or
- the resulting string.

### <strcat>

The strcat expression is used to form strings by concatenating substrings.

**The value definition can be**

- one or more child expressions.

The child expressions are evaluated one by one and the non-target results, interpreted as strings, are concatenated to form the result of this expression. Note that the child expressions do not form a pipeline, but each child will receive the same input.

Any non-target result from the child expressions is appended to the result string. Any target result from the child expressions is caught and ignored.

**The result of the strcat expression can be**

- null, or
- the concatenated string.

**The *strcat expression* can have the following optional attributes:**

<list>

to	Specifies the id or name of the expression to which a non-null result is returned.
----	--

## <substr>

The substr expression is used to extract a substring from a source string.

By default, the input to the filter is used as the source string. A different source string can be specified with the source attribute or child expression.

The start and end indices are specified with the start and end attributes or child expressions.

*The result of the substr expression can be*

- null, or
- the extracted substring.

**The substr expression can have the following optional attributes:**

source	Specifies the source string. Can be constant string or an expression returning a string.
start	Specifies the start index of the substring. Can be a constant or an expression returning an integer or a string having integer value. If not specified, the value 0 is used as the start index. The start index must be $\geq 0$ and $<$ end index and $<$ source string length.
end	Specifies the end index of the substring. Can be a constant or an expression returning an integer or a string having integer value. If not specified, the length of the source string is used as the end index. The end index must be $\geq 0$ and $>$ start index and $\leq$ source string length.
to	Specifies the id or name of the expression to which a non-null result is returned.

## <var>

The var expression is used to define a state variable that can store values during filter processing. The id attribute of the var expression identifies the variable. The value can be set from another expression by targeting the result with the to attribute. The value can be get with the get expression.

*The value definition can be*

- empty,
- a constant string,
- a string containing state variable references, or
- one or more child expressions.

The value definition defines the initial value of the variable. If no value is defined, then, when evaluated, the var expression stores the current input value.

If there are more than one child expressions, the child expressions form an implied pipeline expression.

A non-target result returned from the child expressions is set as the initial value of the variable. Any target

result returned from the child expressions is returned to the parent expression.

*The result of the var expression is*

- the input given to the expression.

**The var expression must have the following attribute:**

id	id - Specifies the id of the variable. Must be a constant string. Needed to refer to the variable from another expression.
----	--

## <get>

The get expression is used to get and return the value of a state variable or the current input value.

The id attribute of the get expression identifies the state variable to get. If no id is specified, then the get expression gets and returns the current input value.

*The get expression has no value definition.*

*The result of the get expression is*

- the value of the state variable specified with the id attribute, or
- the current input value if no id is specified.

**The get expression can have the following optional attribute:**

id	Specifies the id of the state variable to get.
----	--

## <foreach>

The foreach expression is used to loop through a set of values and to evaluate a set of expressions for each value.

*The value definition can be*

- one or more child expressions.

This first child expression is the values-expression, that is, it defines the set of values to loop through.

The rest of the child expressions, if any, are the do-expressions, that is, the set of expressions to evaluate for each value.

If the do-expressions contain more than one expression, the expressions form an implied pipeline expression.

By default, the foreach expression does not have any result. A result can, however, be set, by returning a result to the foreach expression from the descendant expressions by using the to or replacement\_to attribute (of a regex expression).

## Filter\_expressions\_reference

Any uncaught result returned from the do-expressions is returned to the parent expression of the foreach expression.

**The *foreach* expression can have the following optional attributes:**

id	Specifies the id of the foreach expression. Must be a constant string. Needed if you want to set a result of the foreach expression by returning a result to it.
index	Specifies a symbolic name for the foreach-loop index variable. Must be a constant string. The index can be then referred to from within strings by using the syntax <code>\${symbolic-name}</code> .

***The result of the foreach expression can be***

- none, or
- the one returned to the foreach expression from a descendant expression.

## <iterator>

The iterator expression is used to get an iterator over the current input value and to iterate over the values and to evaluate a set of expressions for each value.

The *iterator expression* can have two child expressions: *next* and *after\_last*.

The *next* expression, if defined, is evaluated for each iterated non-null value; each value is given as input to the *next* expression.

The *after\_last* expression, if defined, is evaluated after the last iteration if the last iterated value exists and is non-null. The last iterated value is given as input to the *after\_last* expression.

**The iterator expression does not have any result.**

Any uncaught target result returned from the *next* or *after\_last* expressions is returned to the parent expression of the iterator expression.

## <regex>

The regex expression is used to create a regular expression matcher that can be used to extract data from the input string and apply replacements to the input string.

***The value definition can be***

- a constant string defining the regular expression pattern, or
- one or more child expressions that produce a string defining the regular expression pattern.

If there are more than one child expressions, the child expressions form an implied pipeline expression.

The implementation uses the `java.util.regex` regular expression syntax and engine. Note that the regular expression flags can be defined in the regular expression pattern string (see `java.util.regex.Pattern`

<foreach>

documentation).

***The result can be***

- a regular expression matcher if at least one match is found from the input by applying the regular expression pattern, or
- null, if no match is found.

A regex expression is usually put as the values-expression of a foreach expression. Then, on each round, the returned matcher tries to find the next match and if found, the matcher is given to the do-expressions as input.

The actual matching strings are extracted and returned by using the regex\_match expression with the matcher as input.

The matcher can also be used to produce replacements to the input string by using the attributes replacement and replace\_to.

**The regex expression can have the following optional attributes:**

replacement	Specifies the id of the foreach expression. Must be a constant string. Needed if you want to set a result of the foreach expression by returning a result to it.
replace_to	Specifies the target expression to which the input string with the replacements applied is returned.
count	Specifies the maximum number of matches to find and process. Accepts values "1", "2", etc. for integer values, and any other value for "all". Default is "all".

## <regex\_match>

The regex\_match expression is used to extract the current match or a catching group from the current match of the input regular expression matcher.

The regex\_match expression does not have any value definition.

**The regex\_match expression must have the following attribute:**

group	Defines the catching group to return. Value "0" means that the whole match should be returned.
-------	--

**The following attributes are optional:**

to	Specifies the id or name of the expression to which a non-null result from this expression is returned.
----	---

***The result of the regex\_match expression can be***

- the specified catching group as string,
- the whole match as string, or
- null if no such group or match exists.

## <regex\_match\_start>

The `regex_match_start` expression is used to get the start index of a match or a catching group from the current match of the input regular expression matcher.

The `regex_match_start` expression does not have any value definition.

**The `regex_match_start` expression must have the following attribute:**

group	Defines the catching group. Value "0" means that the whole match.
-------	---

**The following attributes are optional:**

to	Specifies the id or name of the expression to which a non-null result from this expression is returned.
----	---

*The result of the `regex_match_start` expression can be*

- the start index of the specified catching group,
- the start index of the whole match, or
- null if no such group or match exists.

## <regex\_match\_end>

The `regex_match_end` expression is used to get the end index of a match or a catching group from the current match of the input regular expression matcher.

**The `regex_match_end` expression does not have any value definition.**

**The `regex_match_end` expression must have the following attribute:**

group	Defines the catching group. Value "0" means that the whole match.
-------	---

**The following attributes are optional:**

to	Specifies the id or name of the expression to which a non-null result from this expression is returned.
----	---

*The result of the `regex_match_end` expression can be*

- the end index of the specified catching group,
- the end index of the whole match, or
- null if no such group or match exists.

## <xpath>

The `xpath` expression is used to extract data from XML text and XML nodeset inputs by using the XPath expression language.

*The value definition can be*

<regex\_match\_start>

## Filter\_expressions\_reference

- a constant string defining the XPath expression, or
- one or more child expressions that return a string defining the XPath expression.

### *The result can be*

- null, if the XPath expression does not select any nodes, or
- a node set containing the XML nodes selected by the XPath expression.

The xpath expression can be used as the values-expression of the foreach expression. Then, on each round of the foreach loop, the next node in the result node set is given as an input to the do-expressions of the foreach expression.

If the result nodeset contains just either Text or Attribute XML nodes, then the nodeset can be used as a string value. The nodeset is then interpreted as a string composed by appending the text values of the nodes.

## <choice>

The choice expression is used to choose one result among the results from several alternative child expressions. The choice expression chooses the result from the first child expression that produces a non-null result.

### *The value definition can be*

- one or more child expressions.

The child expressions are evaluated one by one in their definition order until a child expression is found that returns a non-null result. Then the evaluation stops and the rest of the child expressions are not evaluated. The non-null result will become the result of the choice expression.

### *The result of the choice expression is*

- the result from the first child expression that returns a non-null result, or
- null, if none of the child expressions return a non-null result.

### **The choice expression can have the following optional attributes:**

to	Specifies the id or name of the expression to which a non-null result from this expression is returned.
----	---

## <to\_date>

The to\_date expression is used to convert the input string value representing a date and time value to a millisecond or second timestamp value or to another format date and time string value. Custom in and out date and time formats can be specified with attributes if the default values are not sufficient. The to\_date expression uses the java.text.SimpleDateFormat syntax.

*The to\_date expression does not have any value definition.*

### *The result is*

<xpath>

## Filter\_expressions\_reference

- an integer holding the millisecond or second value timestamp, or
- a string containing the date and time string value of the desired format.

**The *to\_date* expression can have the following optional attributes:**

type	Specifies the desired output type. Valid values are <ul style="list-style-type: none"><li>• ?ms? for millisecond value integer timestamp. This is the default.</li><li>• ?sec? for second value integer timestamp.</li><li>• ?str? for formatted date and time string.</li></ul>
in_format	Specifies the format of the input string. If not defined, then a number of predefined date and time formats are tried.
out_format	Specifies the format of the output string when the type attribute has value ?str?. If not specified, the default out format is "yyyy-MM-dd'THH:mm:ssZ"

## <to\_hash>

The *to\_hash* expression is used to compute and return a hash value of the input string by computing a digest value of the input string. The hash value is returned as a string containing the hex integer corresponding to the hash value.

**The default message digest used is ?MD5?. A different message digest can be defined with the optional attribute:**

type	Specifies the message digest used to compute the hash.
------	--

*The result of the to\_hash expression is*

- the computed hex integer hash value as string.

## <entity\_decode>

The *entity\_decode* expression is used to decode any [HTML] entity encoded characters in the input string and to return the decoded string.

*The result of the entity\_decode expression is*

- the input string with the encoded characters decoded.

## <untag>

The *untag* expression is used to strip any HTML/XML/? tags from the input string and to return the untagged string.

*The result of the untag expression is*

- the input string with the tags stripped.

## <pipeline>

The pipeline expression is used to form an explicit pipeline of expressions.

*The value definition is*

- one or more child expressions.

A non-target result returned from a child expression is given as an input to the next child expression in the pipeline. Any target result returned from descendant expressions is returned to the parent expression of the pipeline expression.

*The result of the pipeline expression is*

- the result returned from the last expression of the pipeline.

## <tee>

The tee expression is used to form an explicit tee branch.

*The value definition is*

- one or more child expressions.

If there are more than one child expressions, the child expressions form an implied pipeline expression.

A result returned from the child expressions is returned to the parent expression of the tee expression.

*The result of the tee expression is*

- the input given to the expression.

## Example filter

This example filter attempts to parse rss feed, extracting also

```
<img>
```

links to a separate list.

```
<filter id="feeditems">  
  <list>  
    <foreach>
```

```
<untag>
```

## Filter\_expressions\_reference

```
<regex><![CDATA[(?s)<item>(.*?)</item>]]></regex>
<regex_match group="1"/>
<list>
  <item name="title">
    <regex><![CDATA[(?s)<title>(.*?)</title>]]></regex>
    <regex_match to="title" group="1"/>
  </item>
  <item name="created">
    <regex><![CDATA[(?s)<pubDate>(.*?)</pubDate>]]></regex>
    <regex_match to="created" group="1"/>
  </item>
  <choice>
    <regex>
      <![CDATA[
        (?s)<description>.*?<![CDATA\[ (.*?) \]]>.*?</description>
      ]]>
    </regex>
    <regex><![CDATA[(?s)<description>(.*?)</description>]]></regex>
  </choice>
  <regex_match group="1"/>
  <list name="images"/>
  <foreach id="loop" index="i">
    <regex replace_to="loop" replacement="%%IMAGE${i}%%">
      <![CDATA[<img[ ]+src=\"?([^\"]*)\" [\" \"].*?>]]>
    </regex>
    <item to="images" name="%%IMAGE${i}%%">
      <regex_match group="1"/>
    </item>
  </foreach>
  <item name="content">
    <foreach>
      <regex replacement=" "><![CDATA[(?s)<[a-zA-Z/].*?>]]></regex>
    </foreach>
  </item>
</list>
</foreach>
</list>
</filter>
```

The above uses regex expressions to parse the input XML. It is an exercise on how to convert the filter to utilize the xpath expressions.

## See also

- [Getting content with Services](#)
- [Available content fetching services](#)
  - ◆ [Syndication service](#)
  - ◆ [Webfeed service](#)
  - ◆ [HTTP service](#)
- [Fetcher details](#)
  - ◆ [Feed formats](#)
  - ◆ [HTTP authentication](#)
- [Advanced filters](#)
  - ◆ **Filter expressions reference**