WHITE PAPER

dev>scope

Written by

Sandro Pereira

Using Liquid filters in API Management to transform messages

{ We can say that Liquid filters are like functions in C# code or Functoids in BizTalk Server maps. Filters change the output of a Liquid object or variable. }

Table of Contents

Introduction

Liquid is an open-source template language created by **Shopify** and written in Ruby. It is the backbone of Shopify themes and is used to load dynamic content on storefronts.

Azure API Management uses the Liquid templating language (**DotLiquid**) to transform the body of a request or response. This can be effective if you need to completely reshape the format of your message. That can be accomplished by using the set-body policy inside inbound, backend, outbound or on-error policies.

For **example**:

```
<inbound>
    <base />
    …
    <set-body template="liquid">
    {
        "Header": {
            "OrigSystem": "API Management",
            "DateRequest": "{{body.MsgDate}}"
        },
        "InputParameters": {
            "MyObject": {
                "Reference": "{{body.ExtId}}",
                "Type": "{{body.ObjType}}",
                "Id": "{{body.Id}}"
            }
        }
    }
    </set-body>
</inbound>
```

DotLiquid is a deviation from Shopify's original liquid template language, it is a .Net port of the popular Shopify's Liquid templating language. It is a separate project that aims to retain the same template syntax as the original, while using .NET coding conventions where possible. However, they are not entire the same, in some cases they have different behaviors.

Liquid uses a combination of **objects**, **tags**, and **filters** inside template files to display dynamic content.

Where:

- **Objects** contain the content that Liquid displays on a page. Objects and variables are displayed when enclosed in double curly braces: {{ and }}.

```
Hi my name is {{ author.name }}
```

- **Tags** create the logic and control flow for templates. The curly brace percentage delimiters {% and %} and the text that they surround do not produce any visible output when the template is rendered. This lets you assign variables and create conditions or loops without showing any of the Liquid logic on the page.

```
{% if author %}
 Hi my name is {{ author.name }}!
{% endif %}
```

- Filters change the output of a Liquid object or variable. They are used within double curly braces {{ }} and variable assignment, and are separated by a pipe character |.

```
{{ "Sandro" | Append: " Pereira" }}
```

On this whitepaper we will focus on the **DotLiquid Filters** you can use on API Management to transform the body of a request or response.

What is a liquid filter?

As mentioned above, **Filters** change the output of a Liquid object or variable. They are used within double curly braces {{ }} and variable assignment, and are separated by a pipe character |. They are like functions in C# code or Functoids in BizTalk Server maps.

If we apply for example the **Append** filter like:

```
{% assign myvariable = "Sandro" %}
{{ myvariable | Append: " Pereira" }}
```

The end result will be: **Sandro Pereira**

We can also apply multiple **filters** on one output, and are applied from left to right. Like a PowerShell pipeline. A PowerShell pipeline is a series of commands connected by pipeline operators (|). Each pipeline operator sends the results of the preceding command to the next command. Filters use the same concept. For **example**:

```
"Name": "{{ "Sandro Pereira" | Split: " " | Last }}"
```

The end result here will be: **"Name": "Pereira"**

How to apply transformations inside api management policies?

The **set-body** policy can be configured to use the Liquid templating language to transform the body of a request or response. This can be effective if you need to completely reshape the format of your message.

The implementation of Liquid used in the set-body policy is configured in 'C# mode'. This is particularly important when doing things such as filtering. As an **example**, using a date filter requires the use of Pascal casing and C# date formatting e.g.:

```
{{ body.foo.startDateTime| Date: "yyyyMMddTHH:mm:ssZ" }}
```

In order to correctly bind to an XML body using the Liquid template, use a set-header policy to set Content-Type to either application/xml, text/xml (or any type ending with +xml); for a JSON body, it must be application/json, text/json (or any type ending with +json).

Supported Liquid filters

One of the biggest differences between **DotLiquid** and **Shopify's Liquid** is that DotLiquid requires Pascal casing for Liquid filter names, for example: **AtLeast** (DotLiquid) instead of **at_least** (Shopify).

The following DotLiquid filters are supported in the **set-body** policy.

Abs

This filter returns the absolute value of a number. The filter It will also work on a string that only contains a number. Otherwise returns NaN. For **example:**

```
{% assign myvariable = "17" %}
{{ myvariable | Abs }}
```

Result is **17**

```
{% assign myvariable = "17" %}
{{ myvariable | Abs }}
```

Result is **17**

```
{% assign myvariable = "-4" %} {{ myvariable | Abs }}
```

Result is **-4**

```
{% assign myvariable = -4 %} {{ myvariable | Abs }}
```

Result is **-4**

```
{% assign myvariable = "21.15" %} {{ myvariable | Abs }}
```

Result is **21.15**

```
{% assign myvariable = "21,15" %} {{ myvariable | Abs }}
```

Result is **NaN**

```
{% assign myvariable = "Test" %} {{ myvariable | Abs }}
```

Result is **NaN**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
  {
      "OutputMessage": {
```

```
        "InternalId": "{{body.Id}}",
        "ExternalRef": "{{body.ExternalId}}",
        "Age": "{{body.Age | Abs }}"
        "StaticValue": "Sandro"
    }
}
</set-body>
```

Append

This filter adds a specified string to the end of another string. To do that you need to:

```
{% assign myvariable = "Sandro" %}
{{ myvariable | Append: " Pereira" }}
```

Result is **Sandro Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{
    "OutputMessage": {
        "InternalId": "{{body.Id}}",
        "ExternalRef": "{{body.ExternalId}}",
        "filename": "{{body.id | Append: '.xml' }}",
        "StaticValue": "Sandro"
    }
}
</set-body>
```

AtLeast

This filter limits a number to a minimum value. To do that you need to:

```
{% 4 | AtLeast: 5 }}
```

Result is **5**

```
{% 4 | AtLeast: 3 }}
```

Result is **4**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{
    "OutputMessage": {
        "InternalId": "{{body.Id}}",
        "ExternalRef": "{{body.ExternalId}}",
        "Evaluation": "{{body.Eval | AtLeast: 1 }}",
        "StaticValue": "Sandro"
    }
}
</set-body>
```

AtMost

This filter it is the inverse of AtLeast, it limits a number to a maximum value.

To do that you need to:

```
{% 4 | AtMost: 5 }}
```

Result is **4**

```
{% 4 | AtMost: 3 }}
```

Result is **3**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{
    "OutputMessage": {
        "InternalId": "{{body.Id}}",
        "ExternalRef": "{{body.ExternalId}}",
        "Evaluation": "{{body.Eval | AtMost: 9 }}",
        "StaticValue": "Sandro"
    }
}
</set-body>
```

Capitalize

This filter makes the first character of a string capitalized and converts the remaining characters to lowercase. To do that you need to:

```
{% assign myvariable = "Sandro PEREIRA" %}
{{ myvariable | Capitalize }}
```

Result is **Sandro Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{
    "OutputMessage": {
        "InternalId": "{{body.Id}}",
        "ExternalRef": "{{body.ExternalId}}",
        "Name": "{{body.Name | Capitalize }}"
    }
}
</set-body>
```

Ceil

This filter rounds the input up to the nearest whole number. Liquid tries to convert the input to a number before the filter is applied. To do that you need to:

```
{% assign myvariable = 1.4 %}
{{ myvariable | Ceil }}
```

Result is **2**

Despite that is the goal of this filter I couldn't put it working properly on a set-body policy in API Management. If I apply the same operation inside APIM I get 1.4 as result.

Compact

This filter Removes any nil values from an array. To do that you need to:

```
{% assign all_categories = site.pages | Map: "category" | Compact %}
```

```
{% for item in all_categories %}
- {{ item }}
{% endfor %}
```

Based on **shopify documentation**, in this example, assume the object site.pages contains all the metadata for a website. Using assign with the map filter creates a variable that contains only the values of the category properties of everything in the site.pages object. If we perform a **Map filter** the result will then be:

```
- Business
- Celebrities
- Lifestyle
- Sports
- Technology
```

Combined with the **Compact filter** the result then should be like:

```
- Business
- Celebrities
- Lifestyle
- Sports
- Technology
```

On the time I was writing this document I couldn't find a practical sample to set on a **set-body** policy or any other policy in API Management.

Concat

This filter concatenates (joins together) multiple arrays. The resulting array contains all the items from the input arrays. To do that you need to:

```
{% assign fruits = "apples, oranges, peaches" | Split: ", " %}
{% assign vegetables = "carrots, turnips, potatoes" | Split: ", " %}

{% assign everything = fruits | Concat: vegetables %}

{% for item in everything %}
- {{ item }}
{% endfor %}
```

According to the official documentation, the result should be:

```
- apples
- oranges
- peaches
- carrots
- turnips
- potatoes
```

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {% assign fruits = "apples, oranges, peaches" | Split: ", " %}
    {% assign vegetables = "carrots, turnips, potatoes" | Split: ", " %}

    {% assign everything = fruits | Concat: vegetables %}
    {
        "OutputMessage": [
        {% for item in everything %}
            "Item": "{{ item }}"
        {% endfor %}
        }
    }
</set-body>
```

However, it seems that it shouldn't work properly in DotLiquid, since on the time I write this document I'm always getting the following output:

```
- apples
- oranges
- peaches
```

Currency

This filter converts the input object into a formatted currency as specified by the context culture, or languageTag parameter (if provided). To do that you need to:

```
{% assign myvariable = "2" %}
{{ myvariable | Currency }}
```

Result is **$2,00**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ 2 | Currency }}",
        }
    }
</set-body>
```

We can override language for rendering, for example **'fr-FR'**

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ 5 | Currency:'fr-FR' }}",
        }
    }
</set-body>
```

The result will be: 5,00 €

Date

This filter formats a date using a .NET date format string. Opposite to the date filter in shopify that uses **strftime** as the format of the timestamp, DotLiquid use a .Net date format. To do that you need to:

```
{% assign myvariable = "2022-11-02 14:39" %}
{{ myvariable | Date: "dd-MM-yyyy HH:mm:ss" }}
```

Result is **02-11-2022 14:39:00**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{
    "OutputMessage": {
        "InternalId": "{{body.Date | Date: " dd-MM-yyyy HH:mm:ss " }}"
        "StaticValue": "Sandro"
    }
}
</set-body>
```

Default

This filter sets a default value for any variable with no assigned value. default will show its value if the input is nil, false, or empty. To do that you need to:

```
{% assign myvariable = "" %}
{{ myvariable | Default: "Sandro" }}
```

Result is **Sandro**

```
{% assign myvariable = "Pereira" %}
{{ myvariable | Default: "Sandro" }}
```

Result is **Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{
    "OutputMessage": {
        "SystemName": "{{body.System | Default: "APIM" }}"
    }
}
</set-body>
```

DividedBy

This filter divides a number by another number. The result is rounded down to the nearest integer (that is, the floor) if the divisor is an integer. To do that you need to:

```
{% assign myvariable = 10 %}
{{ myvariable | DividedBy: 2 }}
```

Result is **5**

```
{% assign myvariable = 10 %}
{{ myvariable | DividedBy: 3 }}
```

Result is **3**

```
{% assign myvariable = 10 %}
{{ myvariable | DividedBy: 3.2 }}
```

Result is **3.12499995343387**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Name": "{{ 10 | DividedBy: 2 }}"
        }
    }
</set-body>
```

Notice: the filter doesn't accept string has input.

Downcase

This filter makes each character in a string lowercase. It has no effect on strings which are already all lowercase. To do that you need to:

```
{% assign myvariable = "Sandro Pereira" %}
{{ myvariable | Downcase }}
```

Result is **sandro pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ "Sandro Pereira" | Downcase }}",
        }
    }
</set-body>
```

Escape

This filter Escapes HTML chars in a string by replacing characters with escape sequences (so that the string can be used in a URL, for example). It doesn't change strings that don't have anything to escape. To do that you need to:

```
{% assign myvariable = "Sandro > Pereira 'Hello'?" %}
{{ myvariable | Escape }}
```

Result is **Sandro > Pereira 'Hello'?**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ "Sandro &gt; Pereira 'Hello'?" | Escape }}",
        }
    }
</set-body>
```

EscapeOnce

This filter escapes HTML chars in a string without changing existing escaped entities. It doesn't change strings that don't have anything to escape. To do that you need to:

```
{% assign myvariable = "Sandro &lt; Pereira &" %}
{{ myvariable | EscapeOnce }}
```

Result is **Sandro < Pereira &**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ item.Name | EscapeOnce }}",
        }
    }
</set-body>
```

First

This filter returns the first item of an array. To do that you need to:

```
{% assign myvariable = "Sandro Pereira" | Split: " " %}
{{ myvariable | First }}
```

Result is **Sandro**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": [
            "Name": "{{ "Sandro Pereira" | Split: " " | First }}"
        ]
    }
</set-body>
```

Floor

This filter rounds the input down to the nearest whole number. Liquid tries to convert the input to a number before the filter is applied. To do that you need to:

```
{% assign myvariable = 1.4 %}
{{ myvariable | Floor }}
```

Result is **1**

Despite that is the goal of this filter I couldn't put it working properly on a set-body policy in API Management. If I apply the same operation inside APIM I get 1.4 as result.

H

This filter escape html chars. This is exactly the same as Escape filter – it is an alias to the Escape filter. It doesn't change strings that don't have anything to escape. To do that you need to:

```
{% assign myvariable = "Sandro 'A' Pereira" %}
{{ myvariable | H }}
```

Result is **Sandro 'A' Pereira**

Simple sample on a set-body policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ "Sandro 'A' Pereira" | H }}",
        }
    }
</set-body>
```

Join

This filter joins elements of the array into a single string using the argument as a separator. To do that you need to:

```
{% assign beatles = "John, Paul, George, Ringo" | Split: ", " %}
{{ beatles | Join: " and " }}
```

Result is **John and Paul and George and Ringo**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {% assign beatles = "John, Paul, George, Ringo" | Split: ", " %}
    {
        "OutputMessage": [
            "Name": "{{ beatles | Join: " and " }}"
        ]
    }
</set-body>
```

Last

This filter returns the last item of an array. To do that you need to:

```
{% assign myvariable = "Sandro Pereira" | Split: " " %}
{{ myvariable | Last }}
```

Result is **Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": [
            "Name": "{{ "Sandro Pereira" | Split: " " | Last }}"
        ]
    }
</set-body>
```

Lstrip

This filter Removes all whitespace (tabs, spaces, and newlines) from the left side of a string. It does not affect spaces between words. To do that you need to:

```
{% assign myvariable = "    Sandro Pereira     !" %}
{{ myvariable | Lstrip }}
```

Result is **Sandro Pereira   !**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Items": "{{ "      Sandro Pereira     !" | Lstrip }}"
        }
    }
</set-body>
```

Map

This filter creates an array of values by extracting the values of a named property from another object. To do that you need to:

```
{% assign all_categories = site.pages | Map: "category" %}
```

```
{% for item in all_categories %}
- {{ item }}
{% endfor %}
```

Based on **shopify documentation**, in this example, assume the object site.pages contains all the metadata for a website. Using assign with the map filter creates a variable that contains only the values of the category properties of everything in the site.pages object. The result will then be:

```
- Business
- Celebrities
- Lifestyle
- Sports
- Technology
```

On the time I was writing this document I couldn't find a practical sample to set on a **set-body** policy or any other policy in API Management.

Minus

This filter subtracts a number from another number. To do that you need to:

```
{% assign myvariable = 4 %}
{{ myvariable | Minus: 3 }}
```

Result is **1**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Name": "{{ 4 | Minus: 3 }}"
        }
    }
</set-body>
```

If you provide negative values on the Minus input this operation will be converted to an addition. For **example**:

```
{% assign myvariable = 4 %}
{{ myvariable | Minus: -3 }}
```

Result will be **7**.

Notice that, opposite to the Plus filter the Minus do not allow that the input to be a string.

Modulo

This filter returns the remainder of a division operation. To do that you need to:

```
{% assign myvariable = 3 %}
{{ myvariable | Modulo: 2 }}
```

Result is **1**

```
{% assign myvariable = 183.357 %}
{{ myvariable | Modulo: 12 }}
```

Result is **3.357**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Name": "{{ 3 | Modulo: 2 }}"
        }
    }
</set-body>
```

Notice: the filter doesn't accept string has input.

NewlineToBr

This filter inserts an HTML line break (
) in front of each newline (\n) in a string. To do that you need to:

```
{% capture string_with_newlines %}Sandro
```

```
Pereira{% endcapture %}
{{ string_with_newlines | NewlineToBr }}
```

Result is:

**Sandro
**

Pereira

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {% capture string_with_newlines %}Sandro
    Pereira{% endcapture %}
    {
        "OutputMessage": {
            "Items": "{{ string_with_newlines | NewlineToBr }}"
        }
    }
</set-body>
```

Plus

This filter adds a number to another number. If you provide a string this will converts as a concatenation operation. To do that you need to:

```
{% assign myvariable = 4 %}
{{ myvariable | Plus: 4 }}
```

Result is **8**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Name": "{{ 4 | Plus: 4 }}"
        }
    }
</set-body>
```

Once, again if you provide a string as input that will perform a concatenation. For **example**:

```
{% assign myvariable = "4" %}
{{ myvariable | Plus: 4 }}
```

Result will be **44**

Prepend

This filter adds the specified string to the beginning of another string. To do that you need to:

```
{% assign myvariable = "Sandro Pereira" %}
{{ myvariable | Prepend: "person: " }}
```

Result is person: **Sandro Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{
    "OutputMessage": {
        "InternalId": "{{body.Id | Prepend: "entity: " }}"
    }
}
</set-body>
```

Remove

This filter removes every occurrence of the specified substring from a string. To do that you need to:

```
{% assign myvariable = "Sandro \Pereira\" %}
{{ myvariable | Remove: "\" }}
```

Result is **Sandro Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
                "Name": "{{ result.Address | Remove: "\" }}"
        }
    }
</set-body>
```

RemoveFirst

This filter removes only the first occurrence of the specified substring from a string. To do that y ou need to:

```
{% assign myvariable = "Sandro !Pereira!" %}
{{ myvariable | RemoveFirst: "!" }}
```

Result is **Sandro Pereira!**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
                "Name": "{{ result.Address | RemoveFirst: "!" }}"
        }
    }
</set-body>
```

Replace

This filter replaces every occurrence of the first argument in a string with the second argument. To do that you need to:

```
{% assign myvariable = "Sandro \ Pereira" %}
{{ myvariable | Replace: '\\', '\\' }}
```

Result is **Sandro \\ Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
```

```
    {
        "OutputMessage": {
                "Name": "{{ result.Address | Replace: '\\', '\\' }}"
        }
    }
</set-body>
```

ReplaceFirst

This filter replaces only the first occurrence of the first argument in a string with the second argument. To do that you need to:

```
{% assign myvariable = "Sandro !Pereira!" %}
{{ myvariable | ReplaceFirst: '!', '' }}
```

Result is **Sandro Pereira!**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
                "Name": "{{ result.Address | ReplaceFirst: '!', '' }}"
        }
    }
</set-body>
```

Reverse

This filter reverses the order of the items in an array. reverse cannot reverse a string. To do that you need to:

```
{% assign my_array = "John, Paul, George, Ringo" | Split: ", " %}
{{ my_array | Reverse }}
```

Result is: **JohnPaulGeorgeRingo**

```
{% assign my_array = "John, Paul, George, Ringo" | Split: ", " %}
{{ my_array | Reverse | Join: " and " }}
```

Result is: John and Paul and George and Ringo

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {% assign my_array = "John, Paul, George, Ringo" | Split: ", " %}
    {
        "OutputMessage": {
            "Items": "{{ my_array | Reverse | Join: " and "  }}"
        }
    }
</set-body>
```

Round

This filter rounds a number to the nearest integer or, if a number is passed as an argument, to that number of decimal places. To do that you need to:

```
{% assign myvariable = 1.2 %}
{{ myvariable | Round }}
```

Result is **1**

```
{% assign myvariable = 1.2 %}
{{ myvariable | Round: 2 }}
```

Result is **1.2**

```
{% assign myvariable = 1.212 %}
{{ myvariable | Round: 2 }}
```

Result is **1.21**

```
{% assign myvariable = 1,312 %}
{{ myvariable | Round: 2 }}
```

Result is **1312**

```
{% assign myvariable = "1,312" %}
{{ myvariable | Round: 2 }}
```

Result is **1.31**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{
    "OutputMessage": {
        "UnitPrice": "{{body.Price | Round: 2 }}"    }
}
</set-body>
```

Rstrip

This filter Strip all trailing (from the right side) whitespace (tabs, spaces, and newlines) from input. It does not affect spaces between words. To do that you need to:

```
{% assign myvariable = "     Sandro Pereira      " %}
{{ myvariable | Rstrip }}
```

Result is **" Sandro Pereira"**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": [
            "Name": "{{ "    Sandro Pereira    " | Rstrip }}"
        ]
    }
</set-body>
```

Size

This filter returns the size of an array or of a string. To do that you need to:

```
{% assign myvariable = "Sandro" %}
{{ myvariable | Size }}
```

Result is: **6**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
```

```
        "OutputMessage": {
            "Money": "{{ "Sandro" | Size }}",
        }
    }
</set-body>
```

If we work with arrays, then for **example**:

```
<set-body template="liquid">
    {% assign my_array = "apples, oranges, peaches, plums" | Split: ", " %}
    {
        "OutputMessage": {
            "Money": "{{ my_array | Size }}",
        }
    }
</set-body>
```

The result will be **4**.

Slice

This filter returns a substring of one character or series of array items beginning at the index specified by the first argument. An optional second argument specifies the length of the substring or number of array items to be returned. To do that you need to:

```
{% assign myvariable = "Sandro Pereira" %}
{{ myvariable | Slice: 0 }}
```

Result is **S**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {% assign my_array = "apples, oranges, peaches, plums" | Split: ", " %}
    {
        "OutputMessage": {
            "Money": "{{ "Sandro Pereira" | Slice: 0 }}",
        }
    }
</set-body>
```

If Slice is set to 1 that the result will be **a**.

We can also add a second parameter to specify the length of the substring, let's say:

```
{{ myvariable | Slice: 0, 5 }}
```

The result then will be: **Sandro**

Negative values on the first input will count back from the end of the string/array.

```
{{ myvariable | Slice: -3 }}
```

The result then will be: **i**

Sort

This filter sorts items in an array. Opposite to the default **sort** filter in shopify this **is not** a sort in case-sensitive order. DotLiquid Sort filter in case-insensitive to do that you need to:

```
{% assign beatles = "John, Paul, George, Ringo" | Split: ", " | Sort %}
{% for member in beatles %}
  {{ member }}
{% endfor %}
```

Result is:

- **George**
- **John**
- **Paul**
- **Ringo**

Simple sample on a **set-body** policy:

```
<set-body template="liquid">
    {% assign beatles = "John, Paul, George, Ringo" | Split: ", " | Sort %}

    {{ beatles}}
    {
        "OutputMessage": [
            {% for member in beatles %}
```

```
                "Member": "{{ member }}"{% if forloop.last != true %},{% endif %}
            {% endfor %}
        ]
    }
</set-body>
```

Split

This filter split input string into an array of substrings separated by given pattern. To do that you need to:

```
{% assign beatles = "John, Paul, George, Ringo" | Split: ", " %}
{% for member in beatles %}
  {{ member }}
{% endfor %}
```

Result would be:

```
- John
- Paul
- George
- Ringo
```

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {% assign beatles = "John, Paul, George, Ringo" | Split: ", " %}
    {
        "OutputMessage": [
        {% for member in beatles %}
            "Member": "{{ member }}"{% if forloop.last != true %},{% endif %}
        {% endfor %}
        ]
    }
</set-body>
```

Strip

This filter removes all whitespace (tabs, spaces, and newlines) from both the left and right sides of a string. It does not affect spaces between words. To do that you need to:

```
{% assign myvariable = "      Sandro Pereira      " %}
{{ myvariable | Strip }}
```

Result is **Sandro Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": [
            "Name": "{{ "    Sandro Pereira    " | Strip }}"
        ]
    }
</set-body>
```

StripHtml

This filter removes any **HTML tags** from a string. To do that you need to:

```
{% assign myvariable = "Sandro <strong>Pereira</strong>" %}
{{ myvariable | StripHtml }}
```

Result is **Sandro Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": [
            "Name": "{{ "Sandro <strong>Pereira</strong>" | StripHtml }}"
        ]
    }
</set-body>
```

StripNewlines

This filter removes any newline characters (line breaks) from a string. To do that you need to:

```
{% capture string_with_newlines %}Sandro
Pereira{% endcapture %}
{{ string_with_newlines | StripNewlines }}
```

Result is **Sandro Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{% capture string_with_newlines %}Sandro
Pereira{% endcapture %}
    {
        "OutputMessage": [
            "Name": "{{ string_with_newlines  | StripNewlines }}"
        ]
    }
</set-body>
```

Notice: it does not remove
. For example, if the input is:

```
{% capture string_with_newlines %}Sandro<br /> Pereira{% endcapture %}
{{ string_with_newlines | StripNewlines }}
```

The result will still be: **Sandro
 Pereira**

Times

Do not get fooled by the name of this filter, it is not to perform operation in datetime or time (clock), this is a mathematic operation. This filter multiplies a number by another number. To do that you need to:

```
{% assign myvariable = 4 %}
{{ myvariable | Times: 2 }}
```

Result is **8**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
{
    "OutputMessage": {
        "TotalValue": "{{body.Quantity | Times: body.UnitPrice }}"
    }
}
</set-body>
```

DotLiquid Times filter also allows that the input to be a string. On that case it will repeat the input the certain amount of times you specify on the multiple input

```
{% assign myvariable = "Sandro" %}
{{ myvariable | Times: 2 }}
```

Result is **SandroSandro**

Truncate

This filter shortens a string down (truncates) to the number of characters passed as an argument. If the specified number of characters is less than the length of the string, an ellipsis (...) is appended to the string and is included in the character count. To do that you need to:

```
{% assign myvariable = "Sandro Pereira" %}
{{ myvariable | Truncate: 6 }}
```

Result is **Sandro...**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ "Sandro Pereira" | Truncate: 40 }}",
        }
    }
</set-body>
```

If we set for **example Truncate**: 40 then the result would be only **Sandro Pereira**

Optional we can setup a suffix to append when string is truncated. The defaults are the ellipsis(...). For **example**:

```
{% assign myvariable = "Sandro Pereira" %}
{{ myvariable | Truncate: 7, "+" }}
```

The result then will be **Sandro+**

TruncateWords

This filter shortens a string down to the number of words passed as an argument. If the specified number of words is less than the number of words in the string, an ellipsis (...) is appended to the string. To do that you need to:

```
{% assign myvariable = "Sandro Augusto Sousa Pereira" %}
{{ myvariable | TruncateWords: 2 }}
```

Result is **Sandro Augusto...**

Simple sample on **a set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ "Sandro Augusto Sousa Pereira" | TruncateWords: 2 }}",
        }
    }
</set-body>
```

If we set for example **Truncate**: 40 then the result would be only **Sandro Augusto Sousa Pereira**

Optional we can setup a suffix to append when string is truncated. The defaults is the ellipsis(...). For **example**:

```
{% assign myvariable = "Sandro Pereira" %}
{{ myvariable | Truncate: 2, "---" }}
```

The result then will be **Sandro Augusto---**

Uniq

This filter removes any duplicate items in an array. To do that you need to:

```
{% assign my_array = "Sandro, Pereira, Sandro, Pereira" | Split: ", " %}
{{ my_array | Uniq | Join: ", " }}
```

Result is **Sandro, Pereira**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {% assign my_array = "Sandro, Pereira, Sandro, Pereira" | Split: ", " %}
    {
        "OutputMessage": {
            "Items": "{{ my_array | Uniq | Join: ", " }}"
        }
    }
</set-body>
```

Upcase

This filter makes each character in a string uppercase. It has no effect on strings which are already all uppercase. To do that you need to:

```
{% assign myvariable = "Sandro Pereira" %}
{{ myvariable | Upcase }}
```

Result is **SANDRO PEREIRA**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ "Sandro Pereira" | Upcase }}",
        }
    }
</set-body>
```

UrlDecode

This filter decodes a string that has been encoded as a URL or by UrlEncode filter. To do that you need to:

```
{% assign myvariable = "sandro.pereira%40devscope.net" %}
{{ myvariable | UrlDecode }}
```

Result is sandro.pereira@devscope.net

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{context.Request.OriginalUrl.Query.Name | UrlDecode }}",
        }
    }
</set-body>
```

UrlEncode

This filter converts any URL-unsafe characters in a string into percent-encoded characters. To do that you need to:

```
{% assign myvariable = "sandro.pereira@devscope.net" %}
{{ myvariable | UrlEncode }}
```

Result is **sandro.pereira%40devscope.net**

Simple sample on a **set-body** policy

```
<set-body template="liquid">
    {
        "OutputMessage": {
            "Money": "{{ "sandro.pereira@devscope.net" | UrlEncode }}",
        }
    }
</set-body>
```

For me, this filter makes more sense to use not inside bodies but when we are extracting a value from a message and add it to a URL to do for example a Get HTTP call.

Another example is, if the value is **sandro:pereira/net** the result of the UrlEncode filter will be sandro%3Apereira%2Fnet

Where

This filter creates an array including only the objects with a given property value, or any truthy value by default. To do that you need to:

```
All products:
{% for product in products %}
- {{ product.title }}
{% endfor %}

{% assign kitchen_products = products | where: "type", "kitchen" %}

Kitchen products:
{% for product in kitchen_products %}
- {{ product.title }}
{% endfor %}
```

Based on **shopify documentation**, in this example, assume you have a list of products, and you want to show your kitchen products separately. Using where, you can create an array containing only the products that have a "type" of "kitchen". The result will then be:

```
All products:
        - Vacuum
        - Spatula
        - Television
        - Garlic press
Kitchen products:
        - Spatula
        - Garlic press
```

On the time I was writing this document I couldn't find a practical sample to set on a **set-body** policy or any other policy in API Management.

About the Author

Written by **Sandro Pereira**

Azure MVP & MCTS BizTalk

sandro.pereira@devscope.net

sandro-pereira@live.com.pt

twitter.com/sandro_asp

linkedin.com/in/sandropereira

github.com/sandroasp

Sandro Pereira lives in Portugal and is currently the Head of Enterprise Integration at **DevScope**. In the past years, he has been working on implementing Integration scenarios both on-premises and cloud for various clients, each with different scenarios from a technical point of view, size, and criticality, using Microsoft Azure (API Management, Logic Apps, Service Bus, Event Hubs, PowerApps, Power Automate, ...), Microsoft BizTalk Server and different technologies like AS2, EDI, RosettaNet, SAP, TIBCO and many others.

Sandro is very active in the BizTalk community as **blogger**, member and moderator on the MSDN BizTalk Server Forums, TechNet Wiki author, Code Gallery and GitHub contributor, member of several online communities, guest author at BizTalk360 and Serverless360, public speaker and technical reviewer of several BizTalk and Azure books and whitepapers, all focused on Integration. He is also the author of the book **BizTalk Mapping Patterns & Best Practices**.

He has been awarded the Microsoft Most Valuable Professional (MVP) since January 2011, for his contributions to the world-wide **BizTalk Server community** and **Microsoft Azure com**. He currently holds MCTS: BizTalk Server 2006 and BizTalk Server 2010 certifications.

About DevScope

Rua de Passos Manuel 223, 3º
4000-385 Porto
T. 22 375 1350
info@devscope.net
devscope.net

DevScope specializes in giving organizations the tools and knowledge they need to be competitive. We are one of the most distinguished Microsoft partners in Portugal, one of the few companies in the country with two advanced specializations and 9 Gold certifications in different technological areas. We are pioneers and always work with the latest technology to provide our customers and partners with the most advanced solutions. For almost 20 years, we have developed and implemented solutions in and outside Portugal in the most varied areas of activity, from retail to health, through real estate or the public sector, producing lasting results.

Solutions

- Power Platform
- Portals – Office 365 & SharePoint
- Web and App Development
- AI Machine Learning
- Business Intelligent
- Enterprise Integration
- Cloud & DevOps
- Training & Education

Products

- PowerBI Tiles Pro
- PowerBI Robots
- PowerBI Portal
- PowerBI Data Portal
- SmartDocumentor Cloud

Microsoft Partner
Advanced Specialization: Analytics
Advanced Specialization: Low Code
Gold Application Development
Gold Application Integration
Gold Cloud Platform
Gold Cloud Productivity
Gold Collaboration and Content
Gold Data Analytics
Gold Datacenter
Gold DevOps

dev>scope

Rua de Passos Manuel 223, 3º
4000-385 Porto | T. 22 375 1350
info@devscope.net

devscope.net