

# Tutorial: Developing transformations

Author: David Haasler

## Table of contents

1	Creating transformations for a »quotation«.....	2
2	Literal methods.....	3
2.1	Outputting object data.....	7
3	Binary methods.....	9
4	Creating transformations for the quotation collection.....	10
4.1	Number of quotations.....	11
4.2	Current time.....	11
4.3	Calculating the index.....	12
4.4	Outputting the respective quotation.....	13

## Requirements

For you to be able to work through this tutorial, you should have a **basic knowledge of XSLT**. If you would like to work through a tutorial on these topics first, we recommend the tutorial of w3schools.com.

- > [XSLT Tutorial von w3schools](#)
- > [Define Information Model](#)

## Description

The second part of the series shows how quickly and simply the recorded data can be utilised by means of XSL transformations.

Note: The processing logic is created in a transformation container in the module context of the onion.net Editor. A transformation is always assigned to a schema. It is understood here as an object-oriented method which, as is usual in programming, can be extended by means of derivations and parameterized by means of method signatures.

Incidentally: The processing logic is recorded in onion.net as information. This is one of numerous examples where onion.net has supported us with new solutions and extensions.

## Signs and symbols



Boxes marked with an arrow symbol and a green border contains instruction of what to do next.



This kind of boxes contains tips and tricks.

Source code is shown in blue boxes.

## 1 Creating transformations for a »quotation«

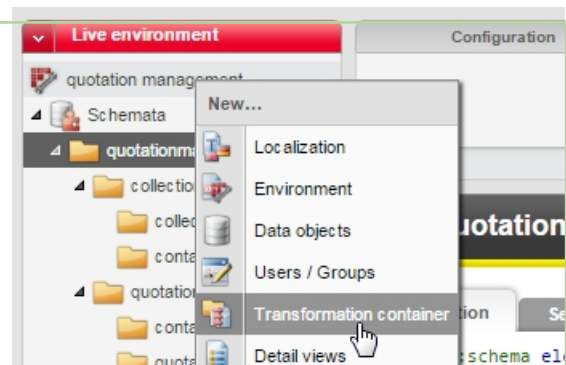
First of all, we create a transformation container below our quotation administration. All quotation transformations are to be located there.

To do this, you need to switch to the module administration of our module quotation management.



Right-click on the object **quotation management**. You now select *transformation container* in the context menu and confirm your action. You can leave the suggestion of **transformations** as the name for the container.

As an additional step, we need to define which data source the data for our transformations come from.



For this purpose, create a new data source with the name **onion.net** through right-clicking on **transformations**. Leave the type as *Onion* and return the object.

For purposes of clarity, related transformations should be grouped, which can be easily done using transformation groups.



Create a transformation group with the title **quotation management** below the data source.

We will next define which data type the transformation is to apply for.



To do this, right-click on the **quotation management** transformation group just created and select the item *Data type* under *New....* Select **quotation** here as the name, since the transformation is to apply for a quotation.

No connection to the schema is established by the name alone.



Click on *name* in order to display the field. The schema must now be inserted into this field.



**Tip:** Drag & drop in order to have the correct data types generated. This saves time and prevents typing errors.



Expand the schemas up to the schema **quotation** and drag & and drop the schema **quotationmanagement/quotations/quotation** into the *name* field of the data type just created. Then return this.

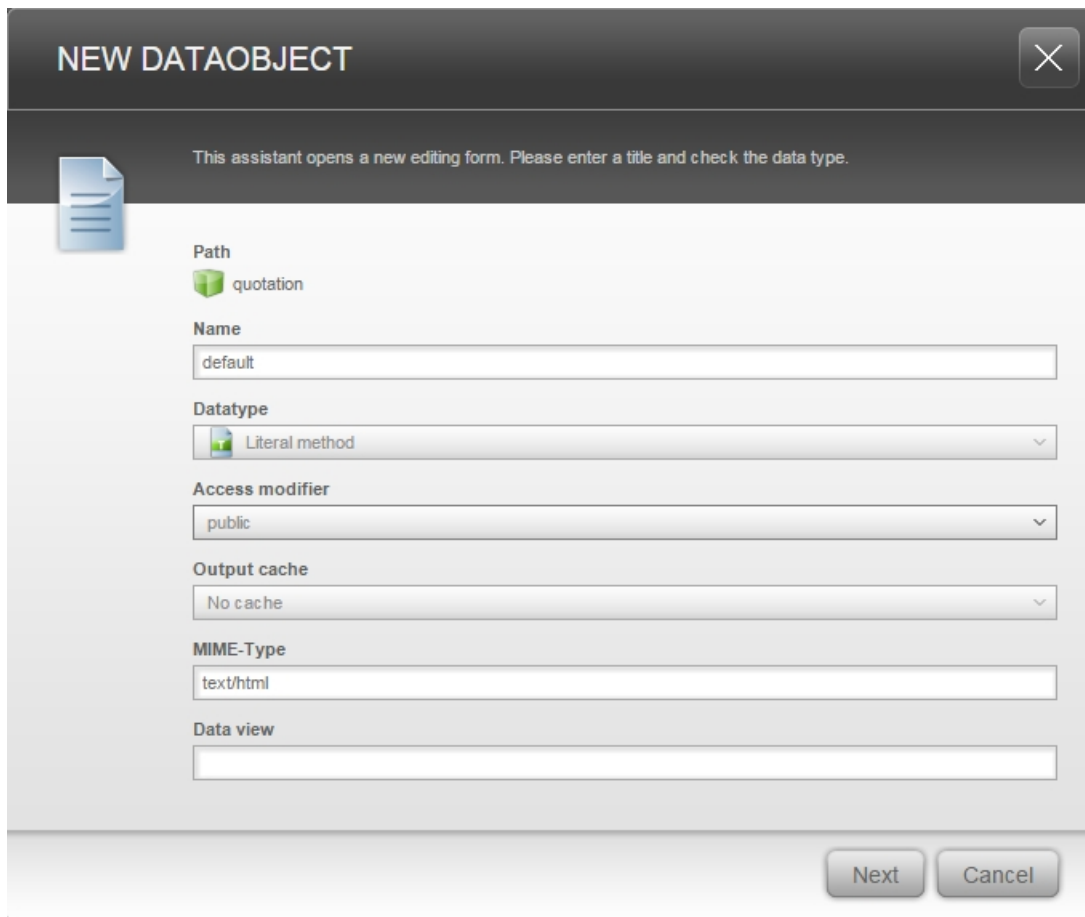
An XML schema is now clearly allocated to the transformation.

## 2 Literal methods

We will now create a literal method for our data type. These are able to generate text-based outputs.



To do this, right-click on the data type **quotation** and create a new literal method from the context menu. Give this method the name **default**.



**default** is considered a default method for the transformation system, meaning that this method is called if an object of this data type is to be displayed.

A number of settings for this method can be made in the dialogue. We must change the access protection in this case. The access protection controls from where the method may be called.

internal from other transformations

protected from .NET Code

public via public interfaces



For the **default** method, we select *public* as access modifier. We then confirm the dialogue by clicking on *Next*.

The tab **transformations** will now open. The actual transformation is now written into the featured code input field.



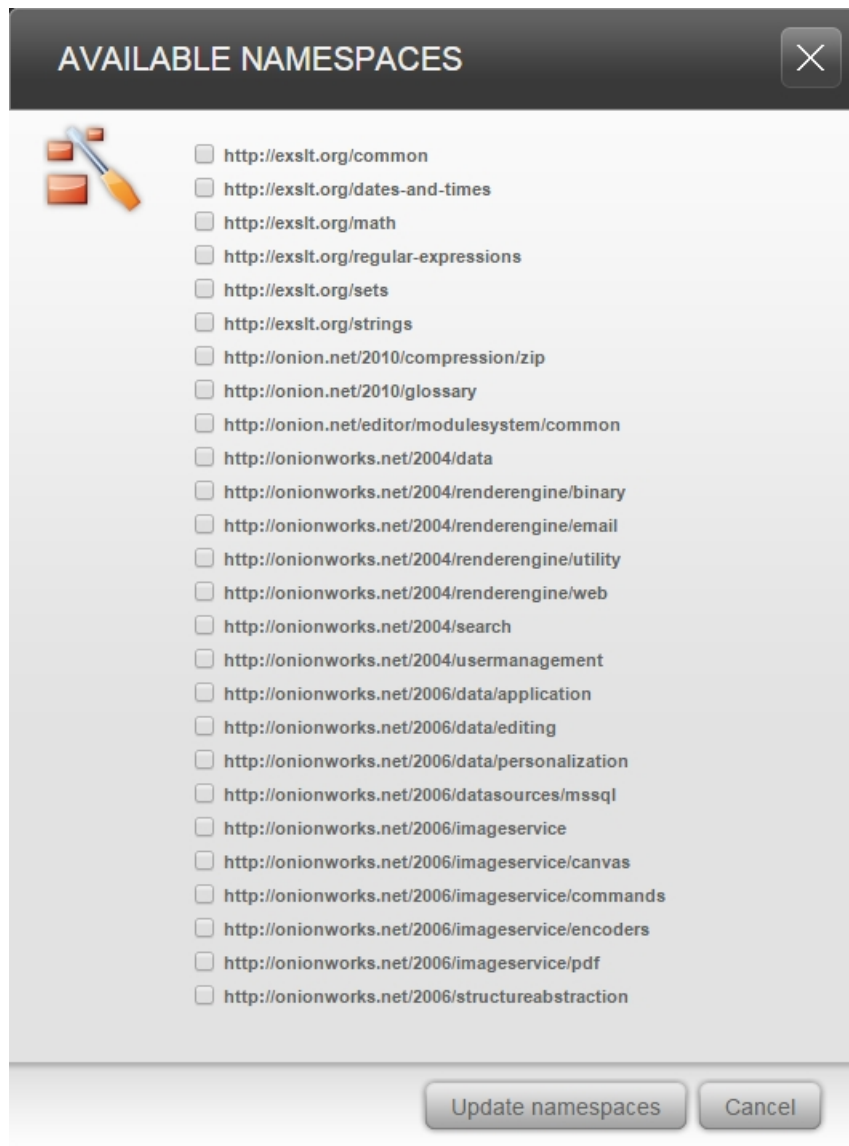
Check whether the cursor is in the code input field. Press **[CTRL] + [Space]**.

This key combination will open an assistant for completing the XSL transformation.

XSLT is an extendable transformation language. The standardized EXSLT extensions are found here as well as useful onion.net extensions containing for example methods for picture manipulation or PDF generation.



Click on *Update namespaces* in the assistant, without selecting a value.



The following default transformation will be created:

```
<xsl:stylesheet version="1.0">
  <xsl:output
    method=" "
    omit-xml-declaration="yes"
    indent="no"
  />
  <xsl:template match="/"></xsl:template>
</xsl:stylesheet>
```



In the `<xsl:output>` element, enter **xml** into the *method* attribute. You enter the desired output within the `<xsl:template>` element. We will make do with "Hello world!" for the time being. Now save the transformation.

The complete transformation now looks as follows:

```
<xsl:stylesheet version="1.0">
  <xsl:output
    method="xml"
    omit-xml-declaration="yes"
    indent="no"
  />
  <xsl:template match="/"> Hallo Welt! </xsl:template>
</xsl:stylesheet>
```

Before the template can be executed, the transformations of the module must be activated on the web server. This change can be carried out via the onion.net Editor.



Right-click on the transformations node and select *web server settings configure*. Select Preview in the dialogue that follows and confirm with *Apply*.



In order to now test the template, right-click on any quotation in the editorial system, e.g. **Quote Albert Einstein**.

Since the editor has now found a **default** method for our quotation, it offers us the function *Preview* in the context menu.

If you click on this, a dialogue will open with a link to the preview and the possibility to *open preview*. If you click on the button, a browser window with the text "Hello world!" opens, exactly as indicated in the transformation.





**Tip:** The pop-up blocker in your browser may possibly prevent the preview from being opened. In this case, configure your browser to allow pop-ups for the current page.

## 2.1 Outputting object data

The text "Hello world!" is now output for each object of the type **quotation**, i.e. for each quotation. We now replace this text with actual values from the respective quotations. As the first step, our transformation is to output the author.



Switch back to the literal method **default** of the data type **quotation**.



**Tip:** Unlike all previous objects, the literal method does not appear as a child element in the tree structure on the left-hand side, but is located in the list view on the top right (object structure window) when you select the appropriate data type. The reason for this is clarity, since further information about the method can be displayed at a glance in the list representation.

Each data object has various data views, each representing an aspect of the object. You can change these in the field *data view*. *no-data* is set by default. In our case, we are collecting our information from the data view *content* since we wish to access information from the content of the object.



Enter the value *content* in the *Configuration* tab under the data view.

We will now process the quotation in accordance with our XML schema. The transformation is to respond to the root node **quotation** and output the text content of the element **author**.



**Tip:** If you have not yet looked into XPath, now is a good time to read through a relevant tutorial. You will find a good tutorial on the pages of [w3schools.com](http://w3schools.com): [XPath Tutorial](http://w3schools.com)



In the `<xsl:template>` element, change the attribute *match* with the XPath expression that selects the root element **quotation**.

```
<xsl:template match="/quotation"></xsl:template>
```

The field with the name **author** is to now be output from quotation.



**Tip:** Some XSLT elements will now be introduced. There is a more detailed description of the individual elements on the w3schools site for example: [XSLT Elements Reference](#)

For this purpose, use the *value-of* element as follows:

```
<xsl:value-of select="author" />
```

Overall, the template block in the method then looks as follows:

```
<xsl:template match="/quotation">
  <xsl:value-of select="author" />
</xsl:template>
```



Now save the method and call the preview of a quotation.

The author of the respective quotation is now output instead of "Hello world!".

In the next step we will output the quotation. The quotation is a formatable text based on XHTML. We can copy this text including formatting instruction into the output.



Add the following lines to the template in the literal method, inserting them after the `<value-of>` element:

```
once said:
<xsl:copy-of select="quote/node()" />
```

If you now open the preview you will, as well as the output of the author, also see the text "once said:" followed by a quotation with formatting.

### 3 Binary methods

We would next like to display the picture. The system offers binary methods for this purpose.



Now create a new binary method below **quotation**. This is given the title **picture**. Under access protection, select *public*. We need the *content* as the data view.

The method we are now creating outputs the picture, precisely adapting it for the quotation, and is not a method that can generally be used for image output.

```
<xsl:stylesheet
  xmlns:b="http://onionworks.net/2004/renderengine/binary"
  xmlns:onion="http://onionworks.net/2004/data"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
>
  <xsl:output
    method="xml"
    omit-xml-declaration="yes"
    indent="no"
  />
  <xsl:template match="/quotation">
    <xsl:value-of select="b.write(image)" />
    <b:output mimeType="{image/@onion:mimeType}">
      <b:webResponse expires="60" />
    </b:output>
  </xsl:template>
</xsl:stylesheet>
```

Two further namespaces are used in this stylesheet (b and onion). These must be added to the stylesheet so that the appropriate places can be resolved.



For this purpose, position the cursor after *xsl: stylesheet* and press **[CTRL] + [Space]**. Select *http://onionworks.net/2004/data* and *http://onionworks.net/2004/renderengine/binary*. Update the namespaces and then return the method.

Now the binary method **picture** will be available to you in your data type also.

We then generate an HTML `img` tag once a picture has been maintained. For referencing our binary method we use the function `c.binaryUri()`.



**Tip:** The function `c.binaryUri()` is explained in more detail in the reference, as are other functions. You can open this here: `c.binaryUri`



Now go back into the literal method **default** and add the following entry to the template after the `<xsl:copy-of>` element:

```
<xsl:if test="image">
  
</xsl:if>
```

Once you have saved this change and reopened the preview, you will see, under the quotation, the picture maintained in the quotation.

## 4 Creating transformations for the quotation collection

We will next create a default method for our quotation collection.



To do this, create a new data type with the title **collection** below the transformation group **quotation management**.

Click on the data type *name* and drag the schema **collections/collection** from the schemas and into the field as described above. Then return the object.

Now create a literal method with the name **default** for the new data type **collection** by right-clicking on the object.

Just like the other methods, this literal method is given the access protection *public*. We also need the *content* of the object as a data view.



Fill out the fields *Access modifier* and *Data view* accordingly.

We will first of all take the default content without changes as the content of the transformation.

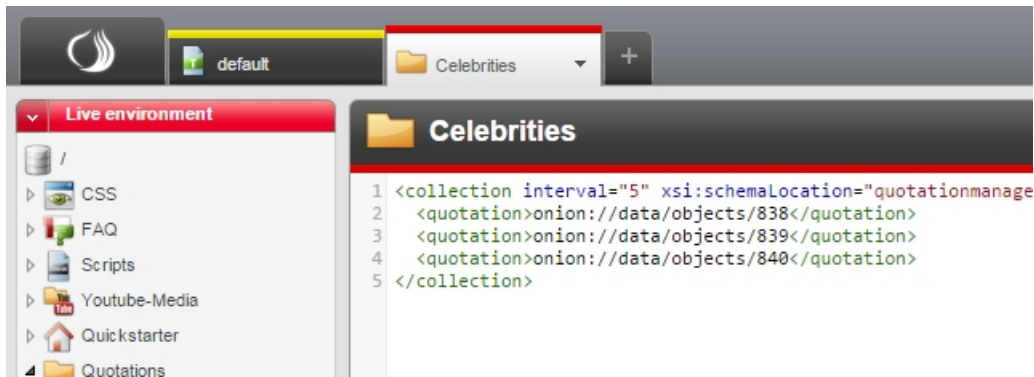


Create the default content with **[CTRL] + [Space]**. In `<xsl:output>` you merely change the attribute method to *xml*. Then save the method.

Let's have a look at the quotation collection as a reminder.



Below the container **quotation collections**, click on the collection **Celebrities**. The detail view will open on the right-hand side of the editor. Switch from the currently active tab *Form view* (at the bottom of the screen) to the tab *Xml raw data* on the far right.



Our new method is to deliver any quotation in seconds within the indicated interval. You can directly see the attribute **interval** in this XML view by looking at the collection element.



Now reopen the literal method of the collection. Just as with the quotation, we change the template element so that it selects the root element **collection**.

```
<xsl:template match="/collection"></xsl:template>
```

## 4.1 Number of quotations

We first output how many quotations are in the collection. We use the function *count()* for this purpose.



Assume the following as the content of the template:

```
Number of quotations:
<xsl:value-of select="count(quotation)" />
<br/>
```

Now take a look at the preview of the quotation collection **Celebrities**. The text "number of quotations" is now output: 3.

## 4.2 Current time

We now output the current time under this.



To do this, enter the following extra line into the content of the template:

```
Current time:
<xsl:value-of select="dt:time()" />
<br/>
```

Since we are using a date function, we need an extension of the namespace. This is done simply by completing the code.



Place the cursor directly after `<xsl:stylesheet` and press **[CTRL] + [Space]**. In the *available namespaces* assistant that now opens, select `http://exslt.org/dates-and-times` and click on *Update namespaces*. The new namespace will now be added for transformation. Save the transformation.

You can see that the current time is now also being output in the preview. If you press the refresh button of the browser, you will see that the time changes each time.

### 4.3 Calculating the index

We will next calculate which quotation must be displayed at the current time. In the first step we will switch on a second-by-second basis.



Extend the template as follows:

```
<xsl:variable name="index" select="floor(dt:seconds() mod count(quotation))
+1" />
```

To check, we will then output the index:

```
Current index:
<xsl:value-of select="$index" />
<br/>
```

We can now see in the preview that, when pressing on the browser button Refresh, a different index is displayed each second.

We now insert the maintained interval of the collection into the formula.



To do this, change the attribute *select* as follows when setting the variable *index*:

```
floor(dt:seconds() div @interval mod count(quotation)) +1
```

The preview check shows that the index changes every five seconds.

## 4.4 Outputting the respective quotation

Lastly, we call our **default** method of the xth quotation. This is done by the command `c.literalCall`.



Extend the template to include the function call:

```
<c.literalCall id="{quotation[$index]}" />
```

The complete default method now looks as follows:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dt="http://exslt.org/dates-and-times"
  xmlns:onion="http://onionworks.net/2004/data"
  version="1.0"
>
  <xsl:output
    method="xml"
    omit-xml-declaration="yes"
    indent="no"
  />
  <xsl:template match="/collection"> Number of quotations:
    <xsl:value-of select="count(quotation)" />
    <br/>
    Current time:
    <xsl:value-of select="dt:time()" />
    <br/>
    <xsl:variable name="index" select="floor(dt:seconds() div @interval mod
count(quotation)) +1" />
    Current index:
    <xsl:value-of select="$index" />
    <br/>
    <c.literalCall id="{quotation[$index]}" />
  </xsl:template>
</xsl:stylesheet>
```

**You've done it!**

In the next part we will customize the editor.