

# Tutorial:

## Object-oriented extension of data types

version: 1

Author: David Haasler

# Table of contents

1	Object-oriented extension of data types.....	2
1.1	Creating a derived schema.....	2
1.2	Creating a new object.....	3
1.3	Adapting the editor.....	3
1.4	Adapting the transformation.....	5

## Requirements

No more prerequisites are needed for these tutorials than for the preceding tutorials.

- >
- >
- >
- > [Structural integrity and child schemas](#)
- > [Creating and distributing modules](#)
- > [Extension of data model and data consistency](#)
- > [Restriction of simple data types](#)

## Description

In this tutorial, it is shown how derivations can be created from schemas and thus the advantages of object-oriented modelling used.

## 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 Object-oriented extension of data types

We will begin with changing the data type for the quotation in such a way that the ComplexType is not available inline but as a designated data type.



Go to the schema “quotation”. Extend the element with the name “quotation” to include the attribute “type=’quotation’” and make it a self-closing element. Delete the now superfluous closing tag `</xs:element>`. Then add an attribute “name” with the value “quotation” to the complexType element.

```
<xs:element name="quotation" type="quotation" />
<xs:complexType name="quotation">
  <xs:sequence>
    <xs:element name="author" type="xs:string" />
    <xs:element name="quote" type="xhtml:Flow" />
    <xs:element
      name="image"
      type="xlink:binaryReference"
      minOccurs="0"
    />
  </xs:sequence>
</xs:complexType>
```

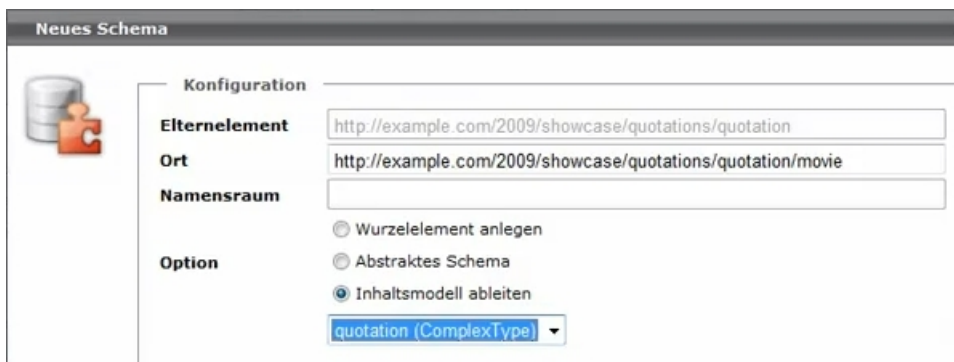


Save the schema.

### 1.1 Creating a derived schema

We will now create a new schema, in which the ComplexType “quotation” is derived.

Create a new schema “movie” below “quotations/quotation”. Now choose “derive content model” as an option. Select the ComplexType “quotation” from the drop-down box.



The derived schema receives an additional attribute.



Add the following attribute within the extension element and then save the schema:

```
<xs:extension base="quotation">
  <xs:attribute
    name="movie"
    type="xs:string"
    use="required"
  />
</xs:extension>
```



Go to the detail view and create a new object of the type “movie” below “quotations”.

Since the new data type is derived from the schema “quotation”, it may be created in the folder “quotations”. As you can see, a further input field “movie” is now displayed in the editor.

## 1.2 Creating a new object



Now create the following quotation and return it:

**Titel: Filmzitat Terminator**

movie: Terminator

Autor: Terminator (Arnold Schwarzenegger)

Zitat: I'll be back.



Now click on the object “quotations”.

As you can see, our newly created object is only displayed in the tree view, but not in the list view.

## 1.3 Adapting the editor

We will now adapt the editor for the new data type.



To do this, go to the editor module “quotation administration” and edit the object “icon”. Where the icons are defined for the schema “quotation”, activate the checkbox “include derivations”.

The tick gives the derived data type the same icon as the quotation.

After refreshing the editor (with F5), you will see that the new quotation “Film quotation Terminator” in the tree view has received the same icon as the objects of the type “quotation”.



Now return the object “icons” and edit the object structure window.

So that we can also see the film quotations in the list view of the editor, we can set the attribute “handleDerivations”.



You add the attribute in the “childType” element:

```
<childType
  schemaLocation="http://example.com/2009/showcase/quotations/quotation"
  structureInvisible="false"
  handleDerivations="true"
/>
```

If you now click on your quotation container “quotations”, you will see the new quotation also in the list view.

We will now translate the remaining elements. We also do this in the editor module “quotation administration” in the object “Ressources/de”.



Add a new “item” element in the “type” element. Use “http://example.com/2009/showcase/quotations/quotation/movie” as the “name” attribute. “Film quotation” is to be output as the text.

```
<item
  name="http://example.com/2009/showcase/quotations/quotation/movie">Filmzitat</item>
```

In addition, the new attribute “movie” is to be given a translation.



In the namespace element “editor”, add a new “type” element for the schema “quotations/quotation/movie”. In the new item-element take “Attribute:movie#Label” as the name-attribute. Take “Film” as the text:

```
<type name="http://example.com/2009/showcase/quotations/quotation/movie">
  <item name="Attribute:movie#Label">Film</item>
</type>
```

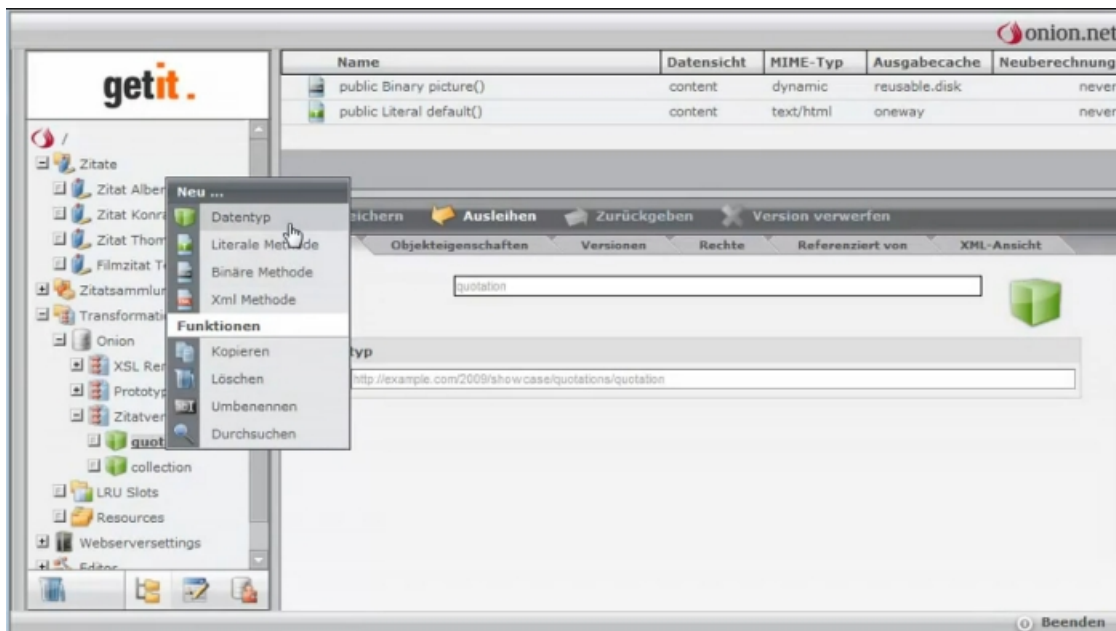
Return the object and go to your film quotation. You will now see the output “Film” for the attribute “movie”.

## 1.4 Adapting the transformation

We must now distinguish in the transformation whether a normal or a film quotation is concerned. The film is not yet considered of course.



Go to your transformation group “quotation administration” and to the data type “quotation” there. Create a new data type “movie” underneath. Drag the schema “quotations/quotation/movie” into the field “name”.



In the transformation for the data type “movie”, we specify the output for the film in the literal method “default” to start with.



To do this, create a new literal method “default”. Select “public” as the access protection and “content” as the data view. Write „<xml:stylesheet/>“ as the content of the method and use the code completion to create the default content without selcting a further namespace. Enter the value “xml” into the attribute method only in the output-element.

Extend the template-element as follows:

```
<xsl:template match="/quotation"> Zitat aus dem Film:
  <xsl:value-of select="@movie" />
  <br/>
</xsl:template>
```

Now look at a preview of the object “film quotation Terminator”. You will now see the output we just wrote into the literal method. We now extend the “default” method in such a way that the remaining data of a quotation can be transformed by the superordinate method.

For this purpose, the superordinate literal method must be imported. This is done directly at the first point in the stylesheet via the import-element:

```
<xsl:import href="../../default" />
```

So that the imported transformation is also actually applied, it must be applied concretely. For this purpose, insert an apply-imports-element within the template-element:

```
<xsl:template match="/quotation"> Zitat aus dem Film:
  <xsl:value-of select="@movie" />
  <br/>
  <xsl:apply-imports/>
</xsl:template>
```

If you now re-open the preview of the film quotation, you will now see the remaining outputs of the superordinate transformation also. To compare, open the preview of another quotation of the type “quotation”. As always, the output is without the addition “quotation from the film” here.

Object orientation makes complex simple; using existing logic instead of copying it, deriving models instead of reinventing them. This means maintenance efforts are minimised.