Tutorial On Generating Variants Using XSLT

Table of Contents

1. Overview

This tutorial demonstrates how to generate a variant using XSLT transformations on the example of a simple shop project. The products sold in the shop, i.e. laptops, consist of the same components in different variants. According to the chosen product, an order form shall be generated listing the components, the price of each component, and the total sum of the order. This order form shall be plain HTML that is generated using an XSLT script executed in an XSLT transformation.

These are the steps to perform for realizing this shop using **pure::variants**.

1. A new **pure::variants** project for the shop has to be created.
2. The products of the shop, i.e. the laptop variants, have to be modelled in a feature model.
3. The components of the products have to be modelled in a family model.
4. An XSLT script has to be written for generating the HTML order form.
5. Finally an XSLT transformation has to be set up using the XSLT script to generate the order form.

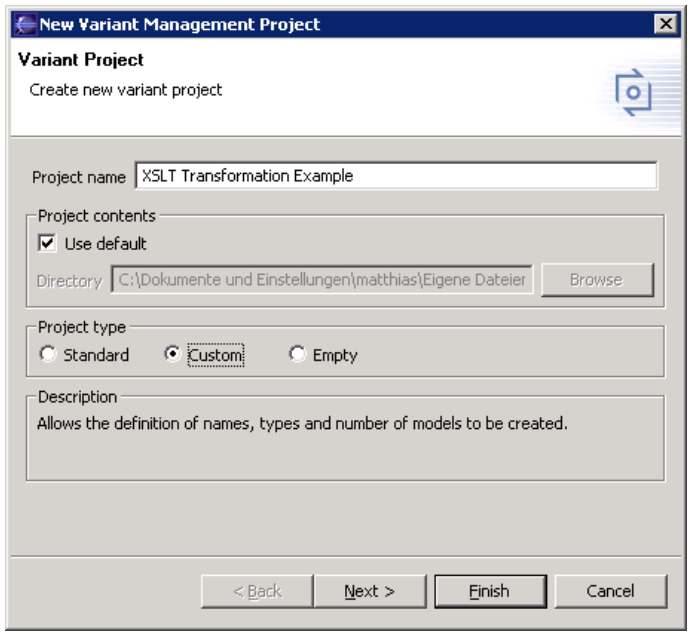
2. About this tutorial

The reader of this tutorial is expected to have basic knowledge about **pure::variants**. Please consult the **pure::variants** introductory material before reading this tutorial. This tutorial is available in online help or in printable PDF format here.

3. Setting up the pure::variants project

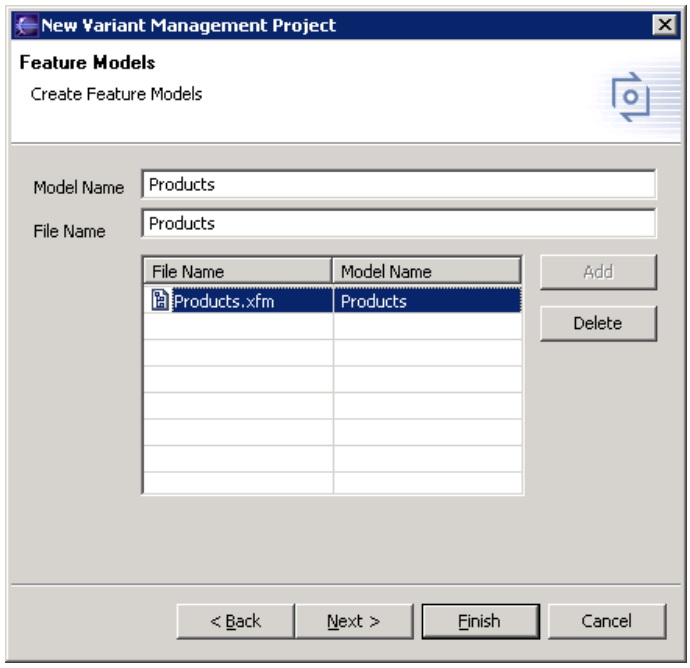
The first step to realize the shop is to create a new **pure::variants** project. Switch to the *Variant Management* perspective and choose *New -> Variant Project* from the context menu of the *Variant Projects* view. Enter "XSLT Transformation Example" as project name, choose *Custom* project type, and click *Next* two times.

Figure 1. The new project wizard



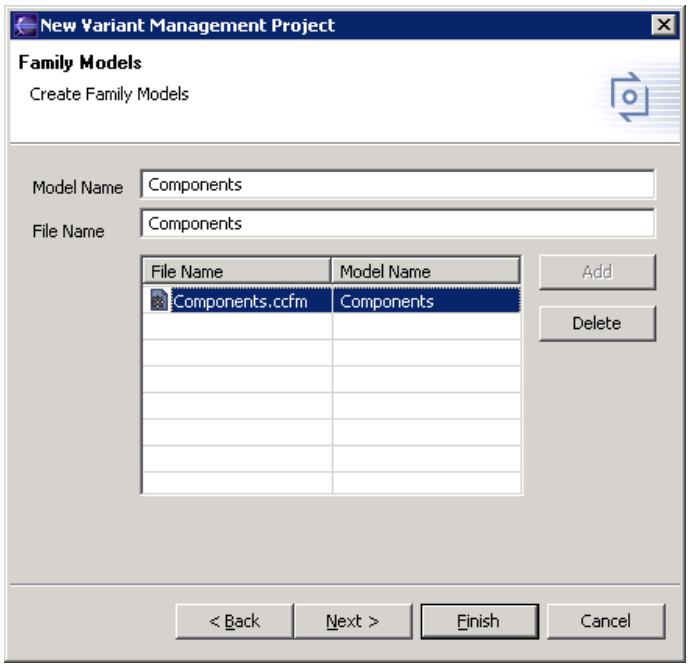
On the *Feature Models* page enter "Products" as model name and click *Add*. This adds a new feature model with the name "Products" to the project.

Figure 2. Adding a feature model



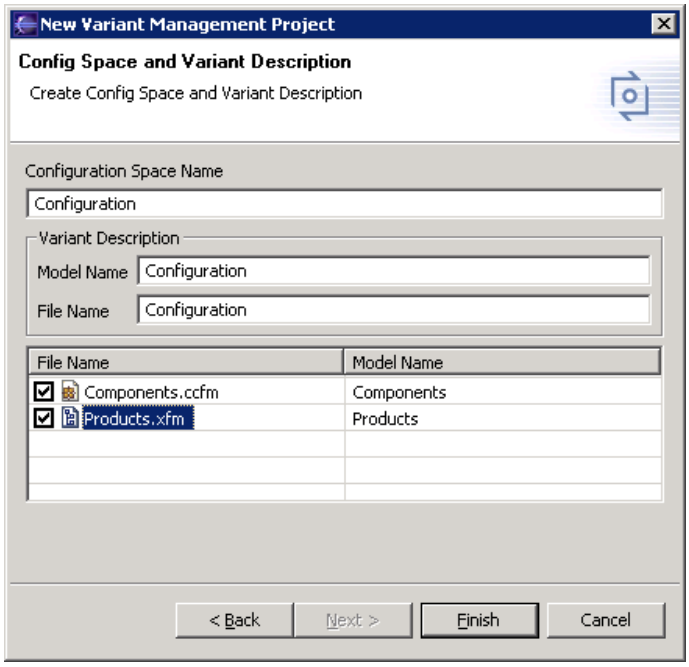
Click *Next* and enter "Components" as family model name and click *Add*. This adds a new family model with the name "Components" to the project.

Figure 3. Adding a family model



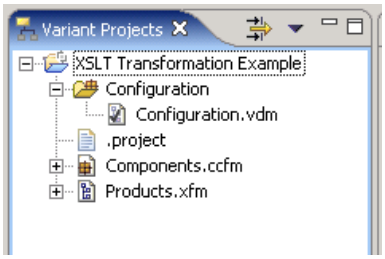
Click *Next*. On the *Config Space* page enter "Configuration" for both the configuration space name and the variant description name. Ensure that "Products.xfm" and "Components.ccfm" are selected. This adds a configuration space and a variant description model to the project.

Figure 4. Adding the configuration space and variant description model



After clicking on *Finish*, the basic project structure is created including the models and the configuration space.

Figure 5. The resulting project structure



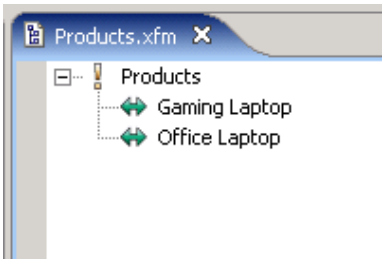
4. Setting up the feature model

The next step is to create the feature model listing the products of the shop. To keep this example short only two laptop variants are available, i.e. a compact office laptop and a high end gaming laptop.

Open the feature model `Products.xfm` by double-clicking on it in the *Variant Projects* view. Right click on the root feature of the model and select *New -> Generic Feature* from the context menu. In the *New Feature* wizard that is opened enter "Gaming Laptop" as the visible name and "Gaming" as the unique name. Select *Alternative* variation type to make this feature member of an alternative feature group from which only one feature can be selected in a variant. After clicking *Finish* the new feature is created. Perform the same steps to create a second feature with the unique name "Office" and the visible name "Office Laptop".

This is all to do for setting up the feature model (see Figure 6, “The Products feature model”).

Figure 6. The Products feature model



5. Setting up the family model

After setting up the feature model, listing the products of the shop, the next step is now to create the family model describing the components of the products, i.e. the two laptop variants. For simplicity reason only three laptop components are modelled: the hard disc, the display, and the memory.

To model the hard disc component, open the family model `Components.ccfm` by double-clicking on it in the *Variant Projects* view. Right click on the root element of the

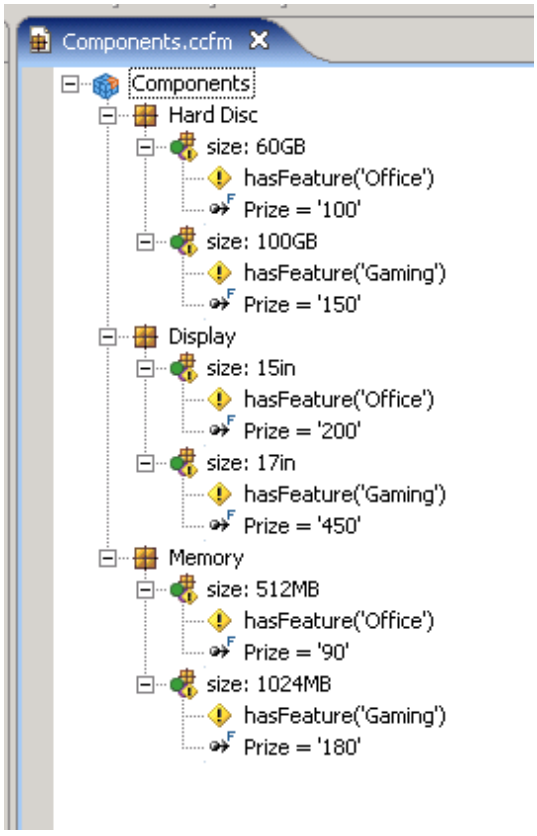
model and choose *New->Component* from the context menu. In the wizard that is opened enter "Hard Disc" as the visible name and click *Finish*. A new component with the name "Hard Disc" is created.

For the two laptop variants two different sized hard discs are available, i.e. 60GB and 100GB. Right click on the new component *Hard Disc* and choose *New->Generic Element* from the context menu. Enter "60GB" as the visible name and "size" as element type in the wizard that is opened. Switch to the *Restrictions* page of the wizard. Click on *Add* to add a new restriction. Enter "hasFeature('Office')" as restriction expression. This restriction effects that only office laptops will be sold with a 60GB hard disc. Switch to the *Attributes* page of the wizard and click on button *Add*. Enter "Prize" as name of the attribute, select "ps:integer" as attribute type, and enter "100" as attribute value. This means that the 60GB hard disc costs 100 EUR.

For the hard disc of the gaming laptop, copy the element *60GB* by right-clicking on it and choose *Copy* from the context menu. Right-click on the element *Hard Disc* and choose *Paste* from the context menu. A copy of the element *60GB* is inserted below the element *Hard Disc*. Double-click on this element and change its visible name to *100GB*. Switch to the *Attributes* page of the dialog and change the value of attribute *Prize* to "150". Finally switch to the *Restrictions* page and change the restriction expression to "hasFeature('Gaming')".

This is all to do for modelling the different hard disc sizes for the two laptop variants. Now perform the same steps to add the two remaining components *Display* and *Memory*. See Figure 7, “The three components Hard Disc, Display and Memory” for the sizes and prizes of the displays and memory chips.

Figure 7. The three components Hard Disc, Display and Memory



After that the family model is nearly finished. For the generation of the order form, the total sum of the components of the chosen laptop variant is needed. For that purpose create a new attribute on the element *Components* by right-clicking on it and choosing *New->Attribute* from the context menu. Enter "Total" as name of the new attribute and set the attribute type to "ps:integer". Now click into the *Value* field of the new attribute and there on button "...". In the dialog that is opened select "Calculation" as kind of the attribute value and enter the following text into the input field.

```
getContext(EID),
sumSelectedSubtreeAttributes(EID,'Prize',Sum),
Value is 500+Sum
```

This code calculates the total sum of the components of the chosen laptop variant. First the values of all attributes with the name "Prize" on the elements of the family model are summarized. Depending on the selected laptop variant only the components of the office resp. gaming laptop are in the variant, and thus only the prizes of these components are summarized. Since the base prize of the laptop shall be 500 EUR, the total sum of the selected laptop is the prize of the components plus 500 EUR.

6. Setting up the XSLT script

The generation of the order form shall be realized using a simple XSLT transformation. For this purpose a corresponding XSLT script is needed that is executed by the XSLT transformation. This XSLT script shall produce a simple HTML page with a title, the list of components for the chosen laptop variant, the size and price of each component, and the total sum of the order.

Create a new file in the root directory of the project by right-clicking on the name of the project in the *Variant Projects* view and choose *New->File* from the context menu. Enter "genhtml.xsl" as the name of the file in the dialog that is opened and click *Finish*. The new file is created and opened. Enter the following text and save the file.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:cm="http://www.pure-systems.com/consul/model"
  xmlns:pv="http://www.pure-systems.com/purevariants"
  extension-element-prefixes="pv">

  <!-- generate indented html output -->
  <xsl:output method="html" indent="yes"/>

  <!-- build element id map -->
  <xsl:key name="element-by-id" match="cm:elements/cm:element"
use="@cm:id"/>

  <!-- select all family models -->
  <xsl:variable name="model" select="//cm:consulmodel[@cm:type='ps:ccm']"/>

  <!-- begin html generation -->
  <xsl:template match="/">
    <html>
      <head/>
      <body>
        <xsl:if test="pv:hasFeature('Office')">
          <h1>Office Laptop</h1><hr/>
        </xsl:if>
        <xsl:if test="pv:hasFeature('Gaming')">
          <h1>Gaming Laptop</h1><hr/>
        </xsl:if>
        <xsl:for-each
select="key('element-by-id',$model/cm:elements/@cm:rootid)">
          <xsl:call-template name="list-components"/>
        </xsl:for-each>
        <hr/>
        <p><b><u>
          Total (+500 EUR base) =
```

```
          <xsl:value-of select="$model//cm:property[@cm:name='Total']"/> EUR
        </u></b></p>
      </body>
    </html>
  </xsl:template>

  <!-- list the components of the product -->
  <xsl:template name="list-components">
    <!-- list the current component -->
    <xsl:call-template name="list-component"/>
    <!-- iterate child elements -->
    <xsl:for-each
select="cm:relations[@cm:class='ps:children']/cm:relation/cm:target">
      <xsl:for-each select="key('element-by-id',substring-after(.,'/'))">
        <!-- traverse subtree of child element -->
        <xsl:call-template name="list-components"/>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

  <!-- generate html for a component -->
  <xsl:template name="list-component">
    <xsl:if test="@cm:type='ps:component'">
      <h2><i><xsl:value-of select="cm:vname"/></i></h2>
    </xsl:if>
    <xsl:if test="@cm:type='size'">
      <p>
        <xsl:value-of select="cm:vname"/> :
        <b><xsl:value-of select=".//cm:property[@cm:name='Prize']"/></b>
        EUR
      </p>
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

For a better understanding of how this XSLT script works, a short description of the parts of the script is given in the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:cm="http://www.pure-systems.com/consul/model"
  xmlns:pv="http://www.pure-systems.com/purevariants"
  extension-element-prefixes="pv">
```

Each XSLT script is surrounded by a `stylesheet` tag. Here it can be specified which XSLT extensions are used in the script. For the order form the **pure::variants** XSLT extensions are used.

```
  <!-- generate indented html output -->
  <xsl:output method="html" indent="yes"/>
```

This line specifies that the output of the script is HTML. Additionally indentation of the generated HTML is enabled.

```
  <!-- build element id map -->
  <xsl:key name="element-by-id" match="cm:elements/cm:element"
use="@cm:id"/>
```

This line builds an unqiue identifier to element map. The map will be used later in the script to access model elements by its unique identifier while traversing the models.

```
  <!-- select all family models -->
  <xsl:variable name="model" select="//cm:consulmodel[@cm:type='ps:ccm']"/>
```

This line defines a variable named `model` containing all concrete family models of the variant that is transformed, i.e. the concrete variant of model *Components* in this case. The

concrete family model contains all the information needed for the order form, i.e. the laptop components, the sizes and prizes, and the calculated total sum of the order.

```
  <!-- begin html generation -->
  <xsl:template match="/">
    <html>
      <head/>
      <body>
        <xsl:if test="pv:hasFeature('Office')">
          <h1>Office Laptop</h1><hr/>
        </xsl:if>
        <xsl:if test="pv:hasFeature('Gaming')">
          <h1>Gaming Laptop</h1><hr/>
        </xsl:if>
        <xsl:for-each
select="key('element-by-id',$model/cm:elements/@cm:rootid)">
          <xsl:call-template name="list-components"/>
        </xsl:for-each>
        <hr/>
        <p><b><u>
          Total (+500 EUR base) =
          <xsl:value-of select="$model//cm:property[@cm:name='Total']"/> EUR
        </u></b></p>
      </body>
    </html>
  </xsl:template>
```

This script part is the starting point of the order form generation. The `template` tag is used to find and process a specific position in the input XML document[1]. In this case the root node of the input XML document is matched. Here the basic HTML structure of the order form is generated. Depending on whether feature *Office* or *Gaming* is selected in the variant description model, and thus the order form for an office or gaming laptop is to be generated, the title of the HTML page is set to "Office Laptop" or "Gaming Laptop". To find out which feature is selected, the **pure::variants** XSLT extension function `hasFeature` is used.

After the title of the page is generated the components of the laptop are listed. Starting at the root element of the family model, using variable `model`, the model is traversed by calling the template `list-components` described below.

Finally the total sum of the order is printed simply by printing the calculated value of the attribute *Total*.

```
  <!-- list the components of the product -->
  <xsl:template name="list-components">
    <!-- list the current component -->
    <xsl:call-template name="list-component"/>
    <!-- iterate child elements -->
    <xsl:for-each
select="cm:relations[@cm:class='ps:children']/cm:relation/cm:target">
      <xsl:for-each select="key('element-by-id',substring-after(.,'/'))">
        <!-- traverse subtree of child element -->
        <xsl:call-template name="list-components"/>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>
```

This script part traverses the family model to print the list of components by iterating the children of the current model element and calling itself recursively for each child element. Here the previously created map is used to access the child elements of an element by its unique identifier. It is not necessary to use the map for this task, but it speeds up the script.

While traversing the model, for each model element the template `list-component` is called.

```
  <!-- generate html for a component -->
```

[1]The XML representation of the concrete model variants.

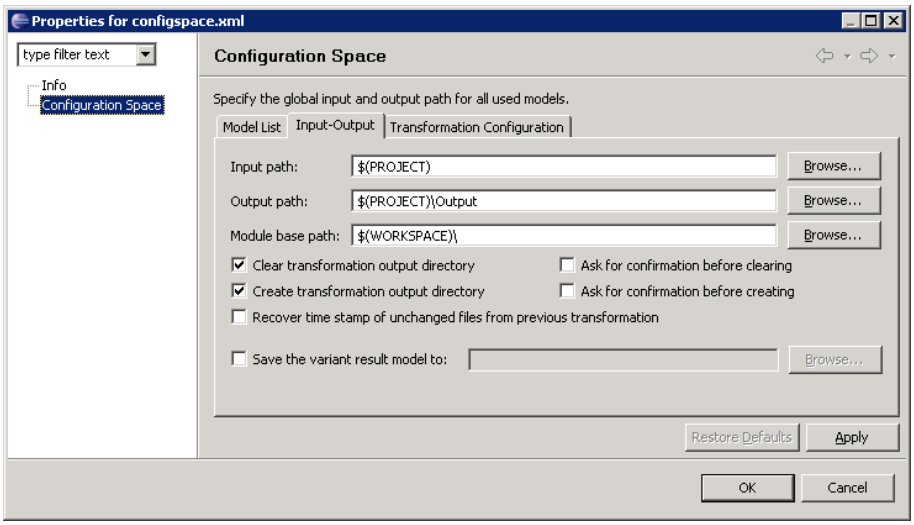
```
<xsl:template name="list-component">
  <xsl:if test="@cm:type='ps:component'">
    <h2><i><xsl:value-of select="cm:vname"/></i></h2>
  </xsl:if>
  <xsl:if test="@cm:type='size'">
    <p>
      <xsl:value-of select="cm:vname"/> :
      <b><xsl:value-of select=".//cm:property[@cm:name='Prize']"/></b>
      EUR
    </p>
  </xsl:if>
</xsl:template>
```

This part of the script generates a list entry for a component of the laptop. It is executed either for a general component of the laptop, like hard disc or display, or for a specific variant of this component, like 60GB hard disc or 17in display. In the first case the visible name of the general component is printed. In the second case the specific size of the component is printed followed by its price.

7. Setting up the transformation

For the transformation some configuration options have to be set. Switch to the *Variant Projects* view and right-click on the name of the configuration space *Configuration*. Select *Properties* from the context menu and switch to the *Configuration Space* page of the dialog that is opened. On the *Input-Output* tab of the dialog enter "$(PROJECT)" as input and "$(PROJECT)\Output" as output directory for the transformation. Enable at least the "Clear transformation output directory" and "Create transformation output directory" check boxes.

Figure 8. The input and output paths configuration



Switch to the *Transformation Configuration* tab and click on button *Add*. In the dialog that is opened select the XSLT script execution module used to execute the XSLT script for generating the order form. Enter "Generate HTML" as name of the module and click *Next*. On the *Module Parameters* page enter "$(PROJECT)/genhtml.xsl" as value of attribute *in* and "offer.html" as value of attribute *out*. After clicking *Finish* the XSLT transformation module is added to the configuration. When the transformation is started this configuration means that the XSLT execution module executes the script `genhtml.xsl` and writes the output of the script to the file `offer.html` in the transformation output directory `Output`.

8. Generating a variant

Now the project is prepared to start a first transformation. Open the variant description model by double-clicking on the file `Configuration.vdm` in the configuration space folder. Select the feature *Gaming Laptop* and click on the tool bar button *Transform Model*. This will start the generation of the HTML order form for a gaming laptop. After the transformation is finished refresh the project in the *Variant Projects* view by selecting the project and pressing key **F5**. The new directory `Output` appears in the project containing the generated HTML file `order.html` (see Figure 9, “After the transformation”).

Figure 9. After the transformation

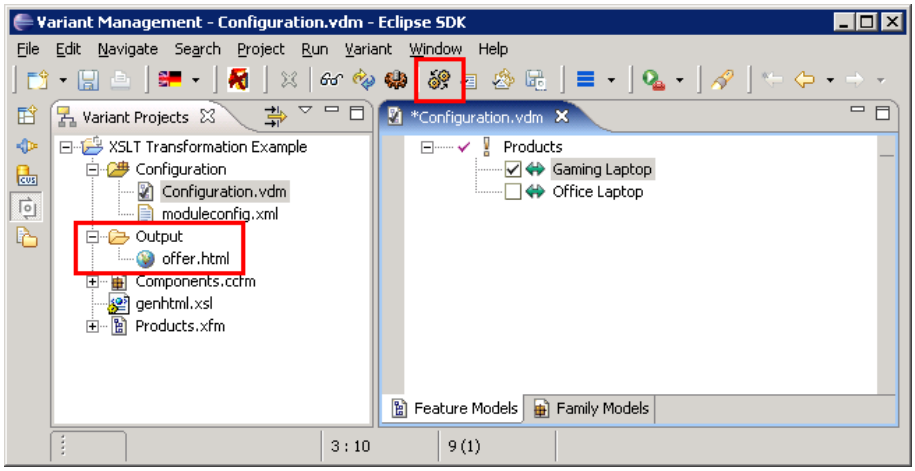


Figure 10, “Order form for the gaming and office laptop” shows the generated order forms for both the gaming laptop and the office laptop.

Figure 10. Order form for the gaming and office laptop