Generating C/C++ style flags

Table of Contents

1. Overview 1
2. About this tutorial 1
3. Setting up the pure::variants project 1
4. Setting up the feature model 2
5. Setting up the family model 2
6. Setting up the transformation 7
7. Creating the flags 7

1. Overview

This tutorial shows how to generate C/C++ style flags with **pure::variants**. The flags are modeled in a family model and setup a transformation generating the defines for the selected features.

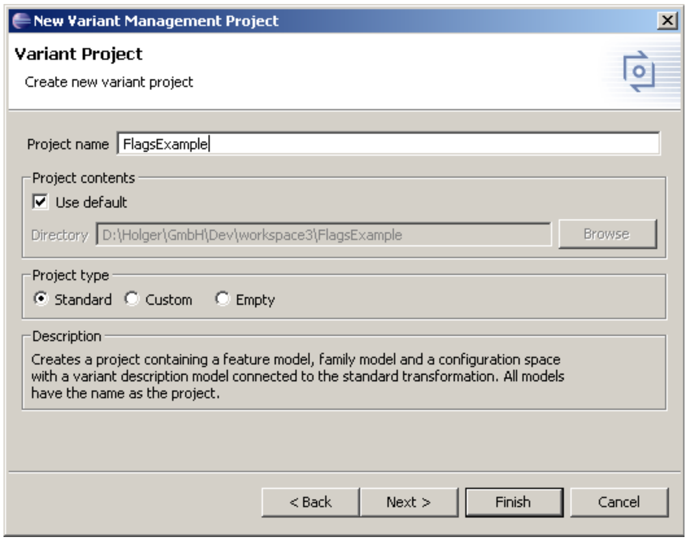
2. About this tutorial

The reader of this tutorial is expected to have basic knowledge about and experiences with **pure::variants** and how the **pure::variants Standard Transformation** works. Please consult its introductory material before reading this tutorial. This tutorial is available in online help as well as in printable PDF format here.

3. Setting up the pure::variants project

The first step is to create the **pure::variants** project. Switch to *Variant Management* perspective and open the context menu (right mouse click) in the *Variant Projects* view. Select *New* -> *Variant Project* from the popup menu. Enter "FlagsExample" as project name and leave all other values as they are.

Figure 1. The new project wizard


New Variant Management Project

Variant Project

Create new variant project

Project name: FlagsExample

Project contents

[✓] Use default

Directory: D:\Holger\GmbH\Dev\workspace3\FlagsExample

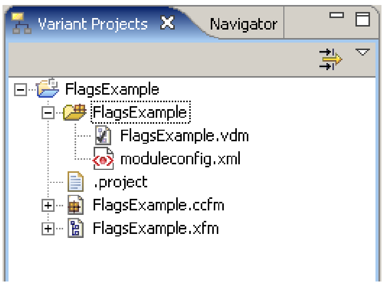
Project type: [✓ Standard] Custom Empty

Description

Creates a project containing a feature model, family model and a configuration space with a variant description model connected to the standard transformation. All models have the name as the project.

After pressing the *Finish* button, the project will be created. Inside the project there is a feature model, a family model, a configuration space and a variant model. All created models are automatically opened.

Figure 2. The created project structure



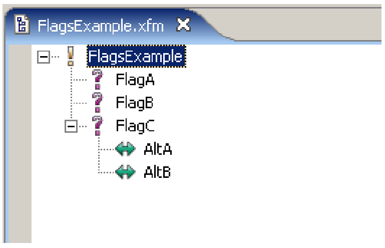
```
FlagsExample
  FlagsExample
    FlagsExample.vdm
    moduleconfig.xml
  .project
  FlagsExample.ccfm
  FlagsExample.xfm
```

4. Setting up the feature model

Select the feature model editor with `FlagsExample.xfm` model and create a new feature below the root feature. Right click on the root feature **FlagsExample** and select *New -> Generic Feature* from the context menu. The new feature wizard will be opened. Enter "FlagA" in the *Unique Name* field and change the type to *ps:optional*. After pressing the *OK* button the new feature named **FlagA** will be created.

Create all other features as shown in Figure 3, "The created feature model". You can use the wizard as before or copy & paste the **FlagA** feature and rename it. The type of the features **AltA** and **AltB** is *ps:alternative*.

Figure 3. The created feature model

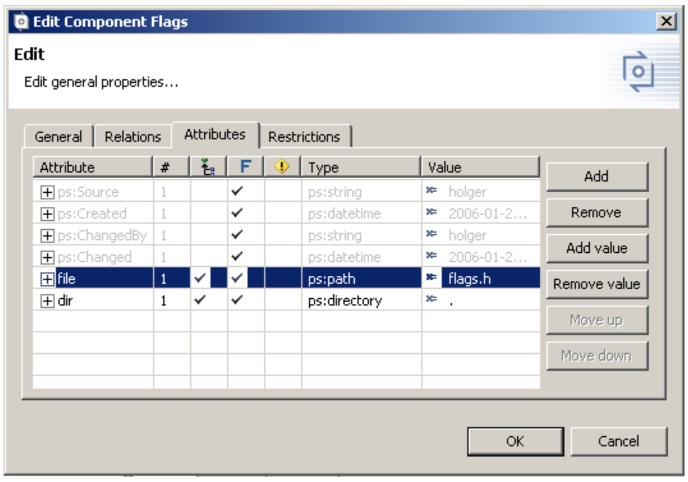


```
FlagsExample
  FlagA
  FlagB
  FlagC
    AltA
    AltB
```

5. Setting up the family model

After mapping the features of the application it is now time to map its compoments to a family model. Go to the family model editor with `FlagsExample.ccfm` and create a new component. Right click the root element **FlagsExample** and select *New -> Component* in the context menu. Enter "Flags" as *Unique Name* and press *Finish*. Since some attributes are required for several child elements and have the same value in most case you have to create these attributes on this component and make them inheritable. Press the right mouse button on the created component and select *New -> Attribute* in the context menu. Name the attribute "file" (note all attribute names are case-sensitive, so i.e. "File" won't work). Select type *ps:path* and set the value to "flags.h". Add another attribute by pressing the *Add* button on the right side of the dialog. Name this attribute "dir" with type *ps:directory* and value ".". Now select for both created attributes the inheritable option . This means that all children inherit this attribute and do not need to define it theirselves.

Figure 4. The inheritable attributes for all flags

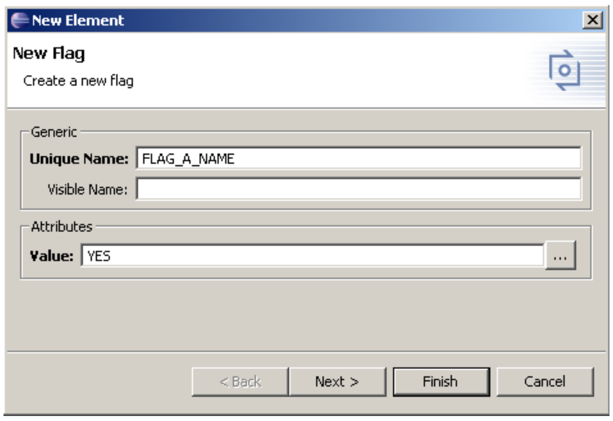


Attribute	#		F		Type	Value
ps:Source	1		✓		ps:string	holger
ps:Created	1		✓		ps:datetime	2006-01-2...
ps:ChangedBy	1		✓		ps:string	holger
ps:Changed	1		✓		ps:datetime	2006-01-2...
file	1	✓	✓		ps:path	flags.h
dir	1	✓	✓		ps:directory	.

Now you can create first flag object. The value of this flag should be set to "YES" if the feature **FlagA** is selected. If the feature **FlagA** is not selected the value should be set to "NO".

Right click the **Flags** component and select *New* -> *Flag* in the context menu. Enter "FLAG_A_NAME" as *Unique Name* and "YES" as *Value* for the flag. Press the *Finish* button to create the flag object.

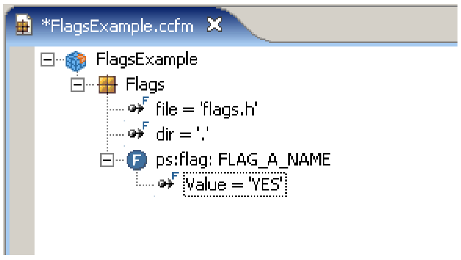
Figure 5. The basic flag object



Unique Name: FLAG_A_NAME
Visible Name:
Value: YES

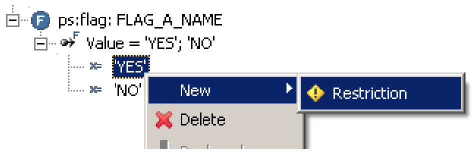
After this the family model should look like the Figure 6, "The family model with the first flag". If there are no attributes values in the model select the *Show In Tree* -> *Attributes* context menu entry of the family model editor.

Figure 6. The family model with the first flag



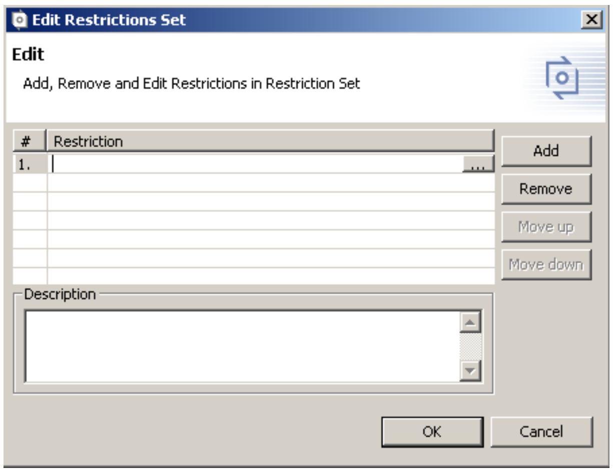
In the next step create the second value for the attribute **Value**. For this right click on the **Value** attribute and select *New -> Attribute Value*. Enter "NO" and press the *OK* button. Now there are two possible values for the attribute and the attribute must be restricted. The **YES** value is only set if the feature **FlagA** is selected in a certain configuration. Right click the **YES** value and select *New -> Restriction* from the context menu.

Figure 7. Add a restriction



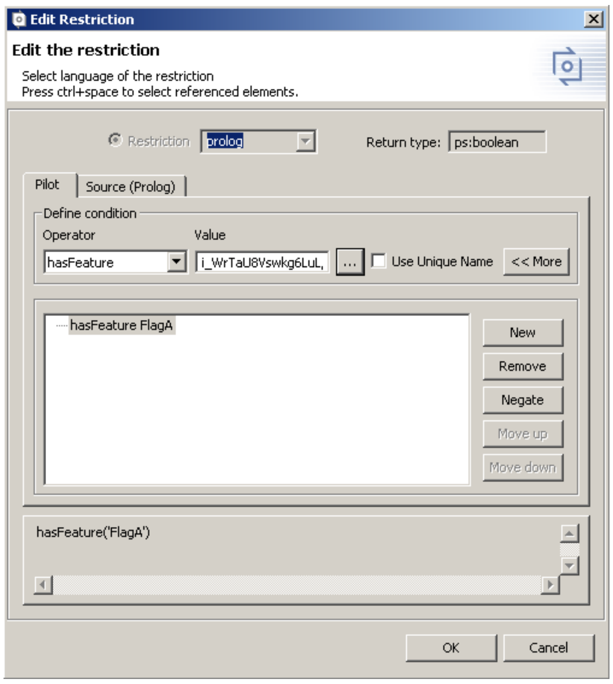
In the upcoming dialog a new restriction is already created. The input line for the restriction code will be activated. You can now enter the code for the restriction or use the restriction pilot. To open the restriction pilot press the button "**...**" at the right end of the input line.

Figure 8. Open the restriction pilot



The restriction pilot will be opened. Select the operation *hasFeature*. Now you have to choose the feature. Press the button "**...**" on the right side of the value field. In the upcoming element selection dialog double click on the **FlagA** feature. The restriction pilot shows the final restriction code in the lower part of the dialog.

Figure 9. The restriction pilot



Edit Restriction

Edit the restriction

Select language of the restriction
Press ctrl+space to select referenced elements.

Restriction: prolog   Return type: ps:boolean

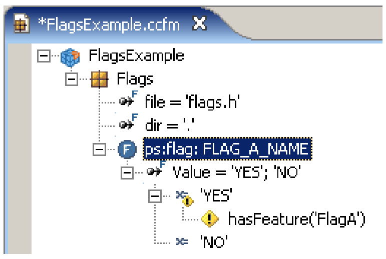
Define condition
Operator: hasFeature   Value: i_WrTaU8Vswkg6LuL,   ☐ Use Unique Name

hasFeature FlagA

hasFeature('FlagA')

After pressing *OK* the created family model look like in the Figure 10, "The complete flag object". The **YES** value is restricted and will only appear if the **FlagA** feature is selected. In all other cases the value will be set to **NO**. If the restriction is not shown in the tree open the *Show In Tree* context menu item again and select the *Restrictions* item.

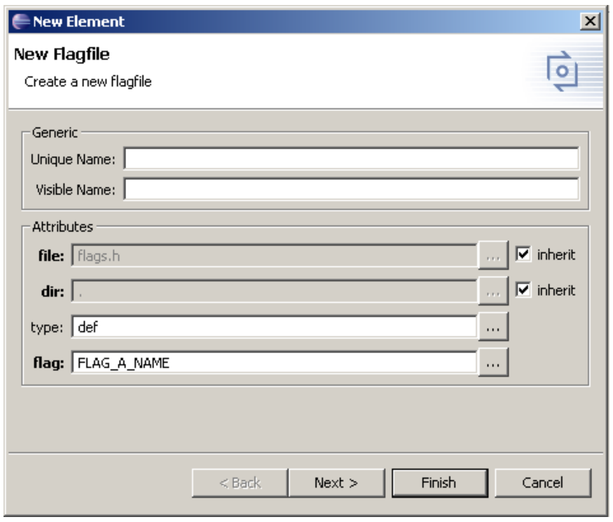
Figure 10. The complete flag object



```
*FlagsExample.ccfm
FlagsExample
  Flags
    file = 'flags.h'
    dir = '.'
    ps:flag: FLAG_A_NAME
      Value = 'YES'; 'NO'
        'YES'
          hasFeature('FlagA')
        'NO'
```

As last information the transformer needs to known the file in which the flag has to be generated. For this you have to add a *Flagfile* element as child of the Flag to the model. Select the created flag object **FLAG_A_NAME** and press the right mouse button. Choose *New -> Flagfile* from the context menu. The **file** and **dir** attributes will be inherited from the parent. For the **type** and **flag** attributes use the provided content in the Figure 11, "The flagfile dialog". Press *Finish* to create the *ps:flagfile* object.

Figure 11. The flagfile dialog


New Element

New Flagfile

Create a new flagfile

Generic

Unique Name:

Visible Name:

Attributes

file: flags.h ☑ inherit

dir: . ☑ inherit

type: def

flag: FLAG_A_NAME

The next flag to create named "FLAG_B_NAME" is only generated if the **FlagB** feature is chosen. For this you have to restrict the *ps:flag* object **FLAG_B_NAME** with the following restriction code:

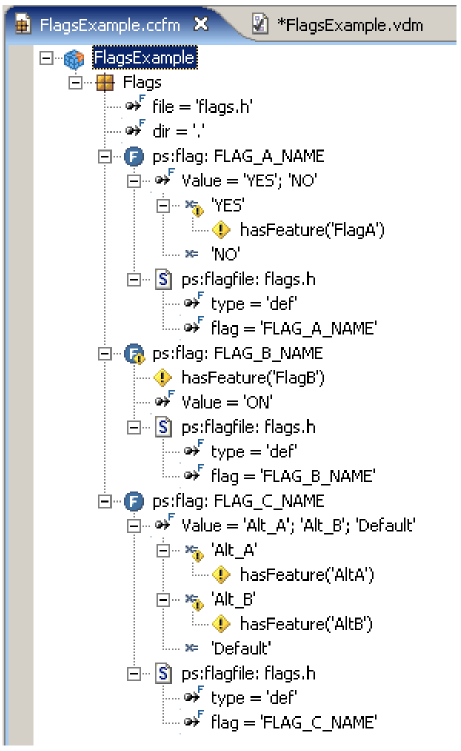
```
hasFeature('FlagB')
```

Please note the difference. In this case is the flag element restrict, not an attribute value. This causes the flag to disappear altogether if the restriction is false (i.e. FlagB is not selected). The attribute value of this flag is always set to "ON".

The third flag named "FLAG_C_NAME" should be created with the first value "Default" and with the next values "Alt_A" if the feature **AltA** is selected or "Alt_B" if the feature **AltB** is selected.

For both last elements **FLAG_B_NAME** and **FLAG_C_NAME** you have to create the *ps:flagfiles* elements like for the flag **FLAG_A_NAME** above. The Figure 12, "The completele family model", shows the complete family model.

Figure 12. The complete family model

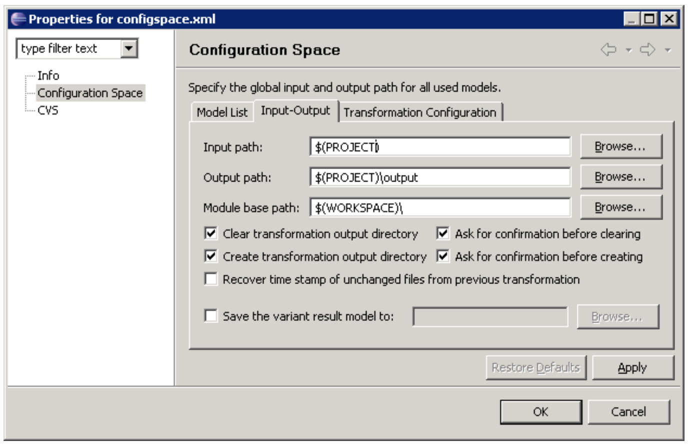


```
FlagsExample.ccfm   *FlagsExample.vdm
FlagsExample
  Flags
    file = 'flags.h'
    dir = '.'
    ps:flag: FLAG_A_NAME
      Value = 'YES'; 'NO'
        'YES'
          hasFeature('FlagA')
        'NO'
      ps:flagfile: flags.h
        type = 'def'
        flag = 'FLAG_A_NAME'
    ps:flag: FLAG_B_NAME
      hasFeature('FlagB')
      Value = 'ON'
      ps:flagfile: flags.h
        type = 'def'
        flag = 'FLAG_B_NAME'
    ps:flag: FLAG_C_NAME
      Value = 'Alt_A'; 'Alt_B'; 'Default'
        'Alt_A'
          hasFeature('AltA')
        'Alt_B'
          hasFeature('AltB')
        'Default'
      ps:flagfile: flags.h
        type = 'def'
        flag = 'FLAG_C_NAME'
```

6. Setting up the transformation

Before start the first transformation you have to define the transformation input and output directories. Go to the *Variant Projects* view and select the **FlagExample** configuration space and open the properties dialog. Switch to the *Input-Output* tab. Enter "$(PROJECT)\output" into the output path field and "$(PROJECT)" into the input path field. Enable the directory options as shown in Figure 13, "The input/output configuration". All other values are initialized correctly during project creation.

Figure 13. The input/output configuration



Properties for configspace.xml

Configuration Space

Specify the global input and output path for all used models.

Model List | Input-Output | Transformation Configuration

Input path: $(PROJECT)

Output path: $(PROJECT)\output

Module base path: $(WORKSPACE)\

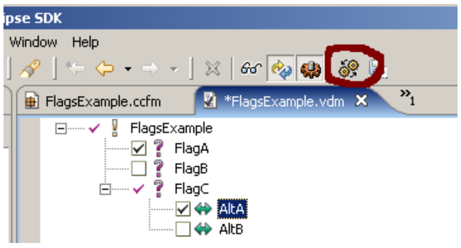
[✓] Clear transformation output directory
[✓] Ask for confirmation before clearing
[✓] Create transformation output directory
[✓] Ask for confirmation before creating
[ ] Recover time stamp of unchanged files from previous transformation

[ ] Save the variant result model to:

7. Creating the flags

Now you can start out the transformation. You have to create a correct variant for this. Open the variant model by double click the `FlagsExample.vdm` file below the configuration space. Select certain features and press the **Transform Model** button.

Figure 14. Start the transformation



Refresh the *Project View* of the project **FlagsExample**. The transformation should generate a file `flags.h` in a new **output** directory in the project . The content for the selection from Figure 14, "Start the transformation", should look like this code:

```
#ifndef __pv_guard_ir9uJMFk1l2_xpdbr_iF8LtpZNecx73cpcO
#define __pv_guard_ir9uJMFk1l2_xpdbr_iF8LtpZNecx73cpcO
#undef FLAG_A_NAME
#define FLAG_A_NAME YES
#endif
#ifndef __pv_guard_iUiRxAPEuCPXg8zSH_iVMk718U4EvwJmlqF
#define __pv_guard_iUiRxAPEuCPXg8zSH_iVMk718U4EvwJmlqF
#undef FLAG_C_NAME
#define FLAG_C_NAME Alt_A
#endif
```