Generating Makefiles

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 3
5. Setting up the family model 3
6. Setting up the transformation 8
7. Generating a variant 9
8. Adding the build options 10

1. Overview

This tutorial shows how to generate variable makefiles. For it the files will be added which are needed for the build as well as compiler options set.

For the tutorial a configurable program will be create which prints a given number or the square of the given number. A feature **Square** controls this behaviour. Additional it can be able to select between **Debug** and **Release** build. For the **Debug** build it can be able to enable **Profiling**.

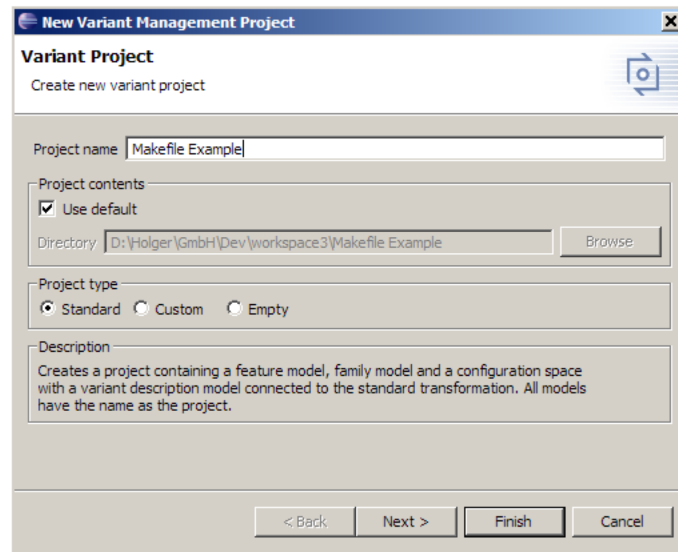
2. About this tutorial

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

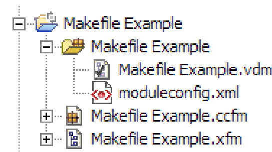
The first step is to create the **pure::variants** project. Switch to the *Variant Management* perspective and open context menu (right mouse click) in the *Variant Projects* view. Select *New -> Variant Project* from the popup menu. Enter "Makefile Example" as project name and leave all other values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

Figure 1. The new project wizard



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 create models are automatically opened.

Figure 2. The created project structure



As the next step you need to create source files. For this you have to create a source folder inside the project. Select the **Makefile Example** project and press the right mouse button. From the context menu select the *New* -> *Folder* menu item. Enter "Source" as folder name and press *Finish*.

In the **Source** folder you have to create the basic makefile. Select the folder and press the right mouse button. From the context menu select the *New* -> *File* item, enter "Makefile" as name and press *Finish*. The text editor with the empty makefile will be opened. The makefile should contain the following rules:

```
include config.mk

foo: $(IMPL_FILES)
        gcc $(CCFLAGS) -o $@ $?
```

The included `config.mk` file will be created by **pure::variants**. The *IMPL_FILES* variable holds all implementation files needed to compile the program. The *CCFLAGS* variable holds configuration dependent compiler options.

As next you need the main function. Create a file `main.c` in the **Source** folder with the following content:

```
#include "foo.h"
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char** argv) {
  int x = atoi(argv[1]);
  printf("foo(%d) = %d\n",x,foo(x));
  return 0;
}
```

Create a header file `foo.h` with the following content:

```
int foo (int value);
```

Now implement two variants of the function "foo". The first variant simply returns the given value. It is implemented in the file `foo.c`.

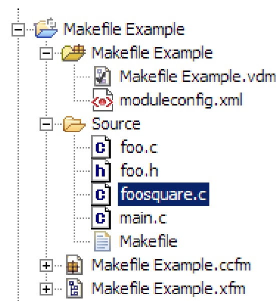
```
int foo (int value) {
  return value;
}
```

The second variant returns the square of the given value. It is implemented in the file `foosquare.c`.

```
int foo (int value) {
  return value*value;
}
```

The final project structure should look like the Figure 3, "The final project structure", below.

Figure 3. The final project structure

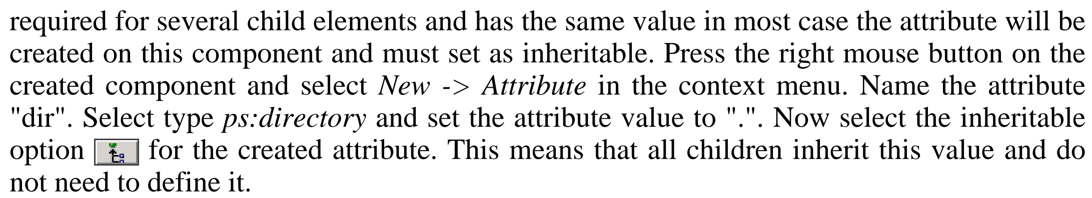


4. Setting up the feature model

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

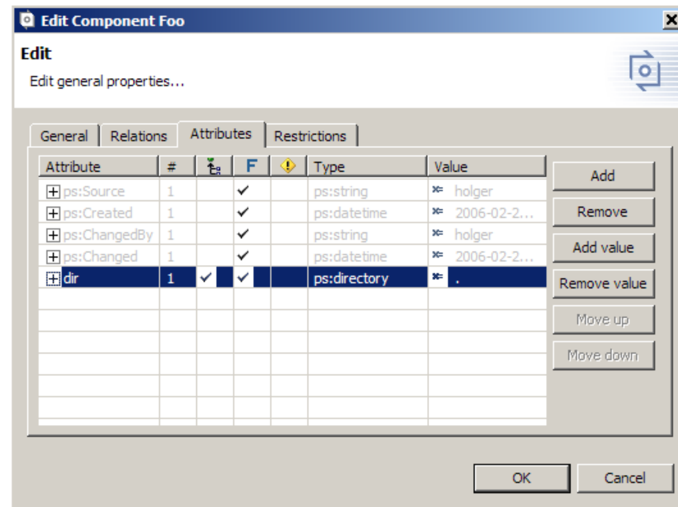
5. Setting up the family model

Go to the family model editor and create a new component in the `Makefile Example.xfm` model. Right click the root element and select *New -> Component* in the context menu. Enter "Program" as *Unique Name* and press *Finish*. Since the **dir** attribute is



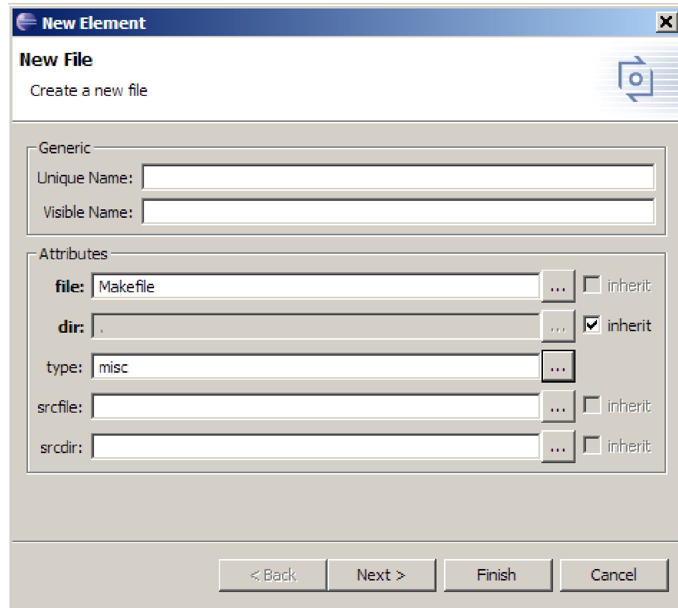
required for several child elements and has the same value in most case the attribute will be created on this component and must set as inheritable. Press the right mouse button on the created component and select *New -> Attribute* in the context menu. Name the attribute "dir". Select type *ps:directory* and set the attribute value to ".". Now select the inheritable option for the created attribute. This means that all children inherit this value and do not need to define it.

Figure 4. The inheritable attribute for all child elements



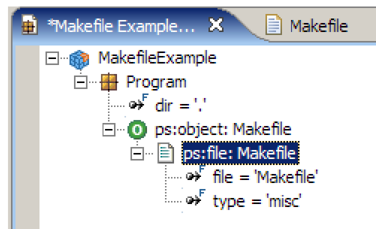
Now the family model element for the makefile can be created. Right click the **Program** component and select the *New -> Object* context menu item. In the upcoming dialog enter "Makefile" as *Unique Name*. After pressing the *Finish* button you get a new model element **Makefile** below the component **Program**. Now you have to specify where the makefile is located. For this you have to create a file element below the **Makefile** element. Right click the Makefile element and select *New -> File* in the context menu. In the upcoming editor enter **Makefile** into the *file* input field. The *dir* attribute is inherited by selecting the inherit check box. The *type* field is set to **misc**.

Figure 5. The file wizard



After pressing the *Finish* button the family model structure show like in the Figure 6, "The family model with the makefile". If there are no attributes values in the model select the *Show In Tree -> Attributes* context menu entry of the family model editor.

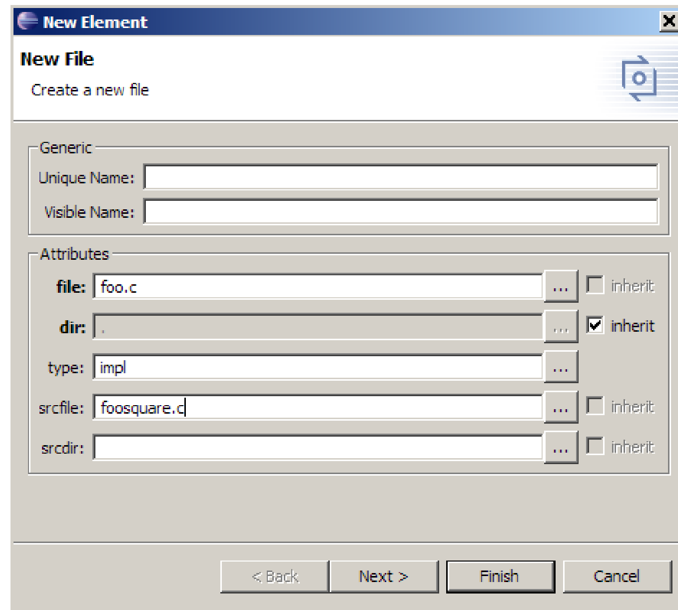
Figure 6. The family model with the makefile



For the `main.c` file from **Source** folder you have to perform the same actions as for the makefile. Create an object with "Main" as *Unique Name*. Below this add a file element. The *file* attribute is set to "main.c". The *dir* attribute is also inherited. Because this is an implementation file set the *type* attribute to **impl**. This will add the file to the **IMPL_FILES** makefile variable later during the transformation process.

The last element in the family model is an object for the function **foo**. Create an other object with "Foo" as *Unique Name*. For this object you have to add two files. The first is the header file `foo.h` and the second is the implementation file `foo.c`. Create a file element for the header file. Set the *file* attribute to "foo.h", the *type* attribute to **def** and inherit the directory. For the implementation file create also a file element. The *file* attribute is set to "foo.c". The *type* attribute is set to **impl**. The *dir* attribute is inherited. Additionally we set the *srcfile* attribute to "foosquare.c".

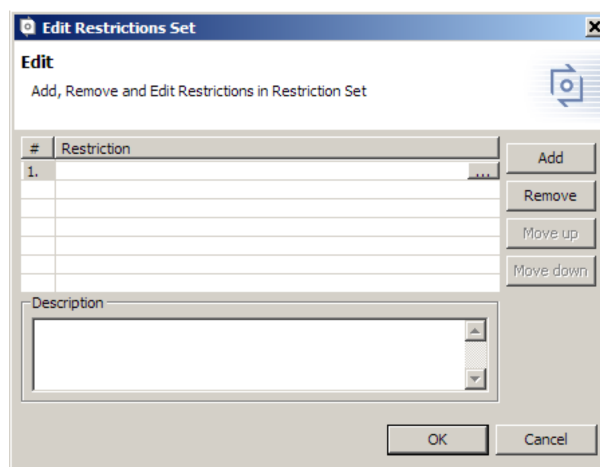
Figure 7. The file definition for `foo.c`



By setting the *srcfile* attribute select an alternative source file for the file `foo.c`. During the transformation the source file is transferred to the destination and renamed to the name specified by the *file* attribute. If the *scrfile* attribute is unset the source and destination name is equal and used form the *file* attribute. You have set the source file name to "foosquare.c" with contains the square implementation. Because of this implementation is only need if the feature **Square** is enabled you need to add an restriction to the *scrfile* attribute.

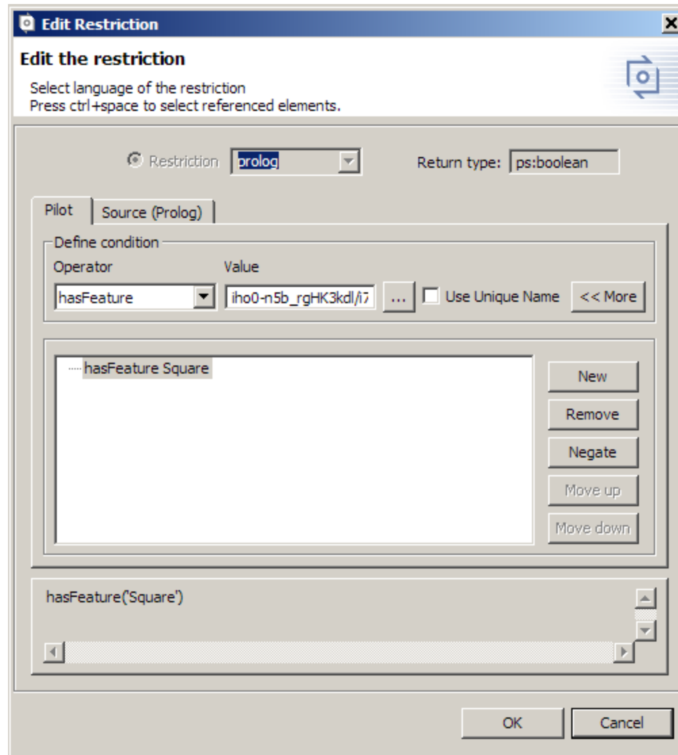
Right click the *scrfile* attribute and select *New -> Restriction* in the context menu. In the upcoming dialog a new restriction is already created. The input line for the restriction code will be activated. You can now enter the code for the restriction or use the restriction pilot. To open the restriction pilot press the button "**...**" at the right end of the input line.

Figure 8. Open the restriction pilot



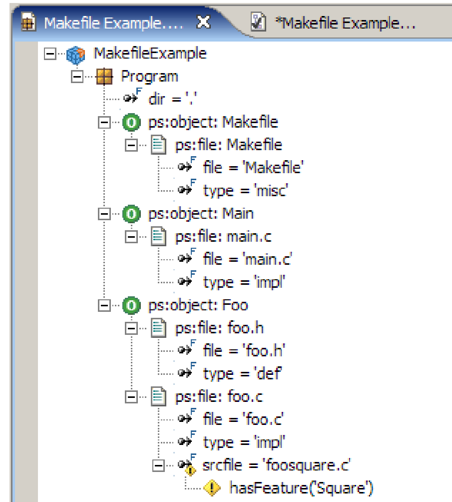
The restriction pilot will be opened. Select the operation *hasFeature*. Now choose the needed feature. Press the button "**...**" on the right side of the value field. In the upcoming element selection dialog double click on the **Square** feature. The restriction pilot shows the final restriction code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

Figure 9. The restriction pilot



After pressing *OK* you get the following model. The *scrfile* attribute is restricted and will only appear if the **Square** feature is selected. If the restriction is not shown in the tree open the *Show In Tree* context menu item again and select the *Restrictions* item. The Figure 10, "The final family model", show the family model `Makefile Example.ccfm` with all created elements.

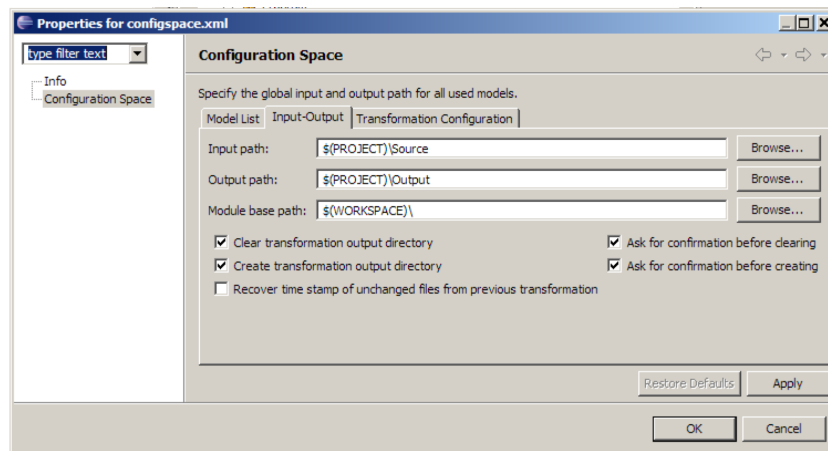
Figure 10. The final family model



6. Setting up the transformation

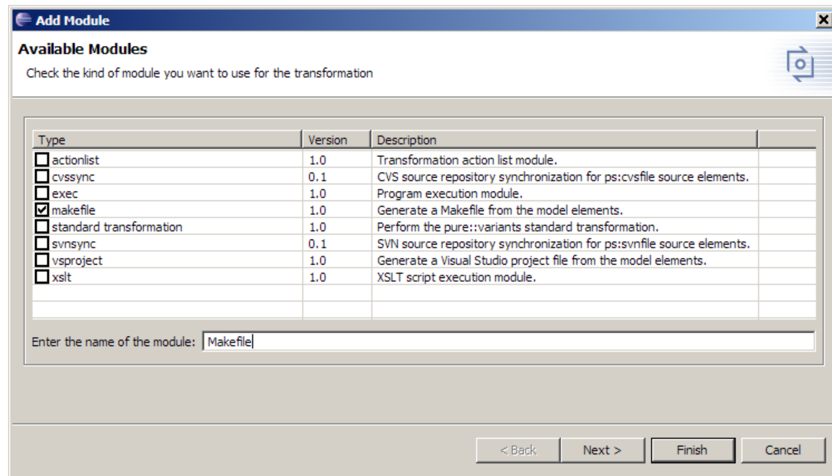
Before start the first transformation you have to configure some option. Go to the *Variant Projects* view and select the **Makefile Example** configuration space and open the properties dialog. To set the input and output directories switch to the *Input-Output* tab. Enter "$(PROJECT)\Source" into the input path field and "$(PROJECT)\Output" into the output path field. Enable the directory options as shown in Figure 11, "The input/output configuration".

Figure 11. The input /output configuration



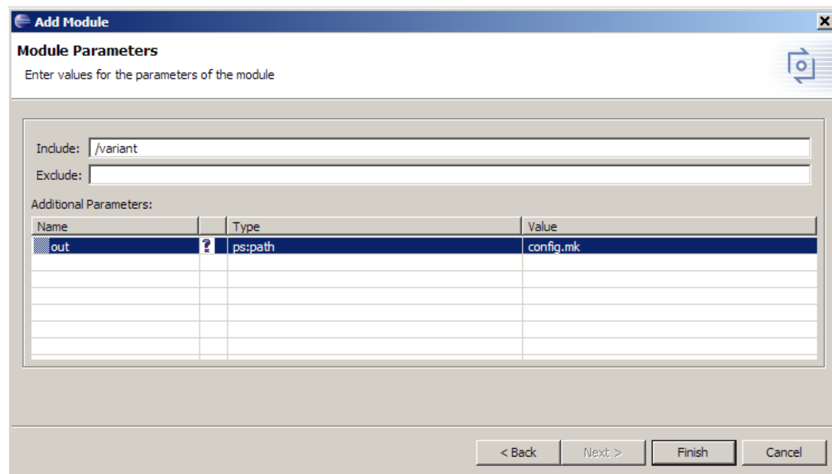
For the makefile creation you need to enable the makefile transformation module. Switch to the *Transformation Configuration* tab. Add a new module by pressing the *Add* button on the right side. In the upcoming dialog check the **makefile** module and enter "Makefile" as module name.

Figure 12. Adding the makefile transformation module



Press the *Next* button to enter the module parameters. The makefile module has only one parameter named *out*. This is the file name of the makefile to be created during the transformation. Set the *out* parameter to "config.mk".

Figure 13. The makefile module parameter dialog

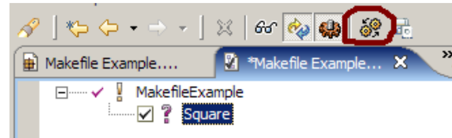


Press *Finish* and *OK* to store the transformation configuration.

7. Generating a variant

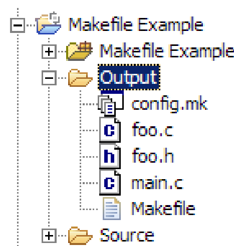
Now you can start the first transformation. You have to create a correct variant for this. Open the variant model by double click the `Makefile Example.vdm` file below the configuration space. Select the **Square** feature and press the **Transform Model** button, shown in the Figure 14, "Start the transformation".

Figure 14. Start the transformation



Refresh the *Project View* to show the output directory. The transformation should generate the **Output** file structure shown in Figure 15, "The resulting file structure" . The file `foo.c` should contain the implementation from the `foosquare.c` file.

Figure 15. The resulting file structure



The `config.mk` contains all files of the project. It should look like this.

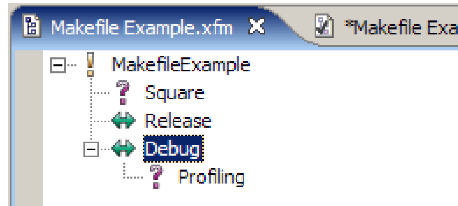
```
DEF_FILES := \
        .\foo.h

IMPL_FILES := \
        .\main.c \
        .\foo.c

MISC_FILES := \
        .\Makefile
```

Change the selection of the **Square** feature and start another transformation. Now the `foo.c` file should contain the code from the `foo.c` file of the Source folder. To build the project open a shell, go to the output directory and call *make*. The program **foo** should be build.

8. Adding the build options

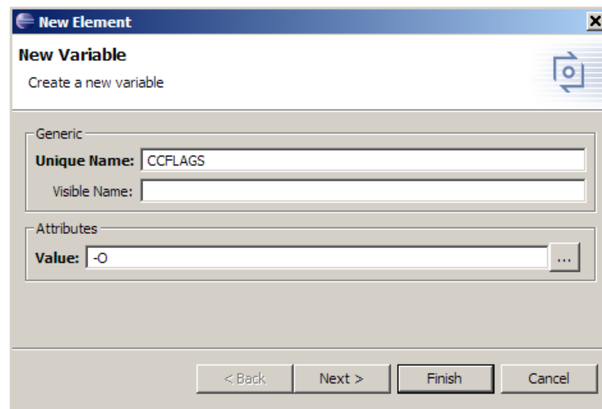
To configure the build options add two more alternative features into the feature model `Makefile Example.xfm`. The features are named "Release" and "Debug". Below the **Debug** feature we create an optional feature "Profiling". The feature model should now look like Figure 16, "The feature model with the build options".

Figure 16. The feature model with the build options



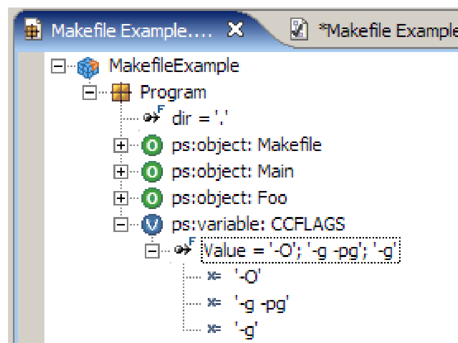
In the family model `Makefile Example.ccfm` you have to add a variable. Right click the **Program** component and select *New -> Variable*. In the upcoming dialog enter "CCFLAGS" as *Unique Name*. The attribute **Value** is set to "-O". This value should be used for the **Release** build configuration.

Figure 17. Creating the CCFLAGS variable



For the other configurations you need to create some more values for the attribute. Right click the attribute **Value** and select *New -> Attribute Value* in the context menu. In the upcoming dialog enter "-g -pg" for the **Debug** with **Profiling** configuration. For the **Debug** without **Profiling** configuration we add another value which is set to "-g". Now the family model should look like Figure 18.

Figure 18. The family model with build options



Now you need to add a restriction to every value. Right click the **-O** value and select *New* -

> *Restriction* from the context menu. Open the restriction pilot and create the following restriction:

```
hasFeature('Release')
```

For the **-g -pg** value create the restriction:

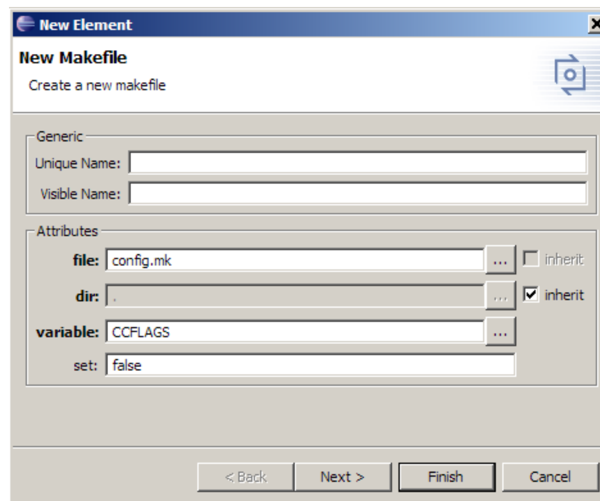
```
hasFeature('Profiling')
```

For the **-g** value create the restriction:

```
hasFeature('Debug')
```

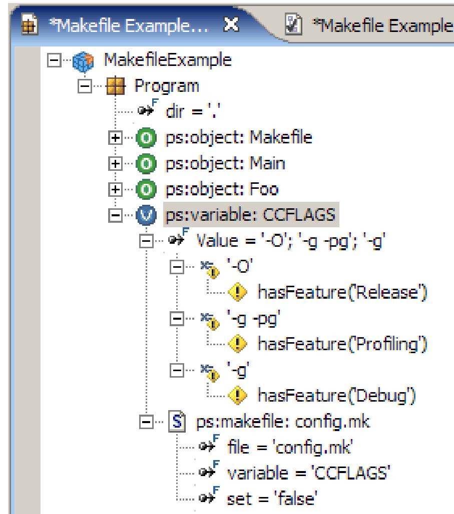
As the last step you need to specify where the **CCFLAGS** variable should be stored. For this create a Makefile element below the variable. Right click the variable **CCFLAGS** and select *New -> Makefile* in the context menu. The *file* input field is set to "config.mk". The *dir* is inherited. The *variable* is **CCFLAGS**. The value for the *set* option is **false**.

Figure 19. The makefile for the CCFLAGS variable



After pressing the *Finish* button you get the following family model.

Figure 20. The final family model



After the next transformation the CCFLAGS variable is in the `config.mk` file. The **Value** depend from the selection of the **Release**, **Debug** and **Profiling** features. An example for **Debug** with **Profiling** is shown below.

```
DEF_FILES := \
        .\foo.h

IMPL_FILES := \
        .\main.c \
        .\foo.c

MISC_FILES := \
        .\Makefile \
        .\config.mk

CCFLAGS += -g -pg
```