Generating Visual Studio Project Files

Table of Contents

1. Overview

This tutorial shows how to generate variable **Visual Studio** project files. As well as setting project options, files which are needed for the build will be added.

For the purposes of the tutorial a configurable program will be creates that prints a given number or the square of the given number. A feature, **Square**, controls this behaviour. **Debug** and **Release** features may also be selected for the build.

2. About this tutorial

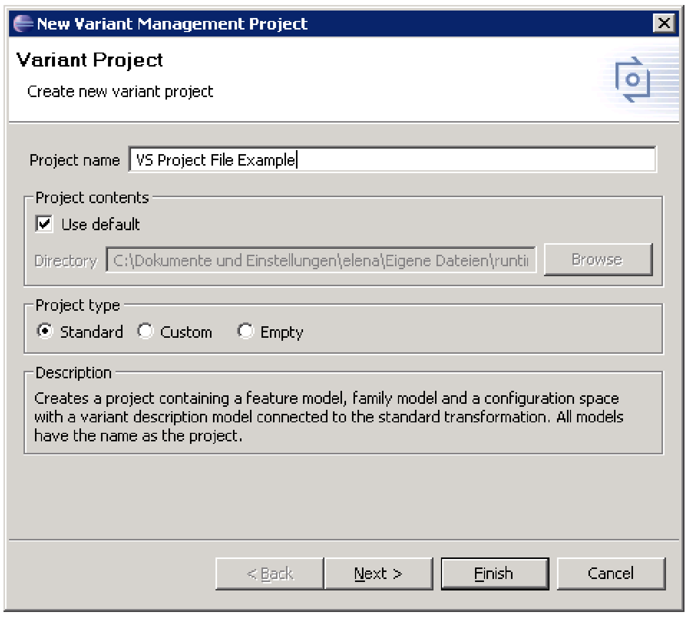
The reader of this tutorial should have basic knowledge of **pure::variants** and how the **pure::variants Standard Transformation** works. Please consult pure::variants introductory material before reading this tutorial.

This tutorial is available as online help or in a printable PDF format here.

3. Setting up the pure::variants project

The first step is to create a **pure::variants** project. Switch to the *Variant Management* perspective and open a Context menu (right mouse click) in the *Variant Projects* view. Select *New* -> *Variant Project* from the popup menu. Enter "VS Project File Example" as the 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: VS Project File Example

Project contents

[✓] Use default

Directory: C:\Dokumente und Einstellungen\elena\Eigene Dateien\runti

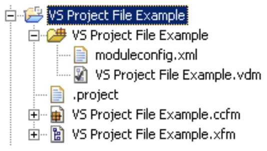
Project type: [● Standard] ○ Custom ○ Empty

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.

Press the *Finish* button to create the project. Inside the project there is a feature model, a family model, a configuration space and a variant model. pure::variants opens all created models automatically.

Figure 2. The created project structure

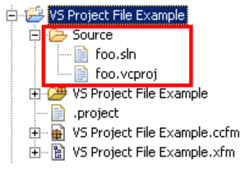


```
VS Project File Example
  VS Project File Example
    moduleconfig.xml
    VS Project File Example.vdm
  .project
  VS Project File Example.ccfm
  VS Project File Example.xfm
```

In the next step you need to create the source files. To do this create a source folder inside the project. Select the project **VS Project File Example** and click the right mouse button. Select the *New -> Folder* menu item from the context menu. Enter "Source" as the folder name and click *Finish*.

Now open Visual Studio and create an **empty** Visual C++ Win32 Console Project named "foo". Deselect *Create directory for Solution* on the project dialog page. Then copy the project and solution files (`foo.sln` and `foo.vcproj`) into the **Source** directory.

Figure 3. The copied Visual Studio project and the solution file



```
VS Project File Example
  Source
    foo.sln
    foo.vcproj
  VS Project File Example
  .project
  VS Project File Example.ccfm
  VS Project File Example.xfm
```

All files referred to by *ps:file* source elements in the family model will be added to the VS project during the transformation.

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("Running version %s\n", VERSION_STRING);
  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 one simply returns the given value. It is implemented in the file `foo.c`.

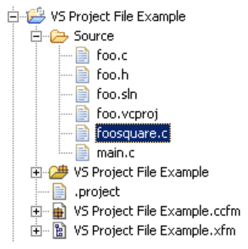
```
int foo (int value) {
  return value;
}
```

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 4 below.

Figure 4. The final project structure



```
VS Project File Example
  Source
    foo.c
    foo.h
    foo.sln
    foo.vcproj
    foosquare.c
    main.c
  VS Project File Example
  .project
  VS Project File Example.ccfm
  VS Project File Example.xfm
```

4. Setting up the feature model

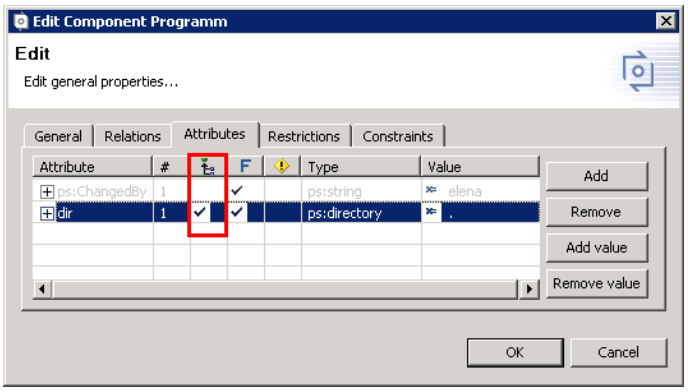
Select the feature model editor `VS Project File Example.xfm` and create a new feature below the root feature. Right click the root feature **VS Project File Example** and select *New* -> *Generic Feature* from the context menu. The new feature wizard will be opened. Enter "Square" in the *Unique Name*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

5. Setting up the family model

Go to the family model editor `VS Project File Example.ccfm` and create a new component. Right click the root element **VS Project File Example** and select *New* ->

Component from the context menu. Enter "Program" as the *Unique Name* and click *Finish*. Tthe **dir** attribute is required for several child elements and since it has the same value in most cases the attribute will be created on the **Program** component and set as inheritable. Click the right mouse button on **Program** and select *New* -> *Attribute* from the context menu. Name the attribute "dir". Select type *ps:directory* and set the value to ".". Now select the inheritable option for the created attribute. This means that all children will inherit this value and so do not need to define it themselves.

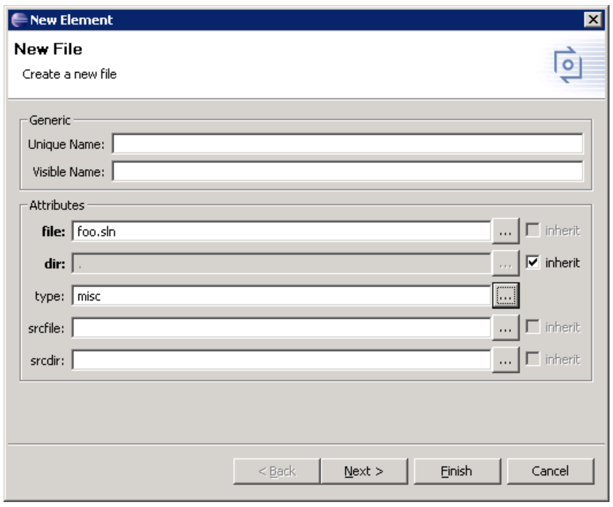
Figure 5. The inheritable attribute for all child elements



Attribute	#		F		Type	Value
ps:ChangedBy	1		✓		ps:string	elena
dir	1	✓	✓		ps:directory	.

Right click the **Program** component and select the *New* -> *Object* context menu item. In the dialog that opens enter "Solution" as *Unique Name*. After clicking *Finish* you get a new model element **Solution** below the component **Program**. Next you have to specify where the project file is located. Create a file element below the **Solution** element to do this. Right click the **Solution** element and select *New* -> *File* in the context menu. In the editor that opens enter "foo.sln" into the *file* input field. The *dir* attribute is inherited by selecting the *inherit* check box. The *type* field is set to **misc**.

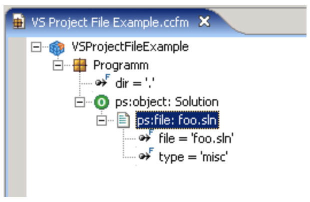
Figure 6. The file wizard



Unique Name:
Visible Name:
file: foo.sln [ ] inherit
dir: . [✓] inherit
type: misc
srcfile: [ ] inherit
srcdir: [ ] inherit

After pressing the *Finish* button we get the following model structure like in Figure 7 below. If there are no attributes values in the model select the *Show In Tree* -> *Attributes* context menu entry of the family model editor. The family model should now show the following elements.

Figure 7. The family model with the foo solution file

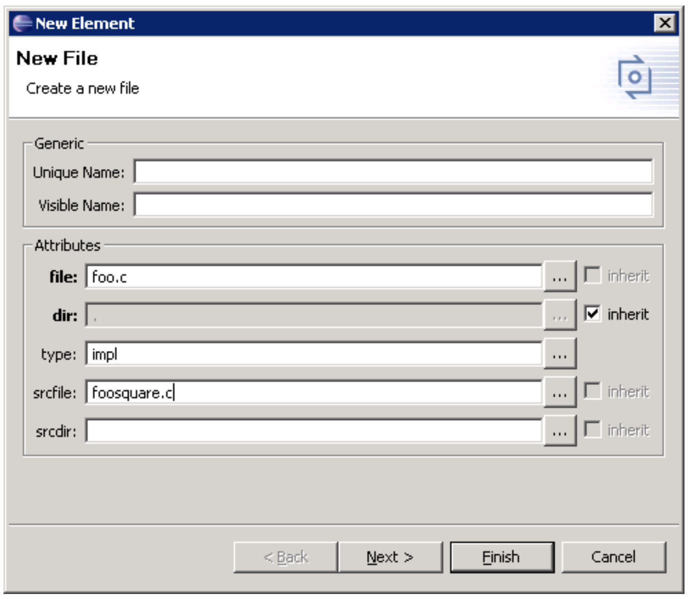


```
VS Project File Example.ccfm
VSProjectFileExample
  Programm
    dir = '.'
    ps:object: Solution
      ps:file: foo.sln
        file = 'foo.sln'
        type = 'misc'
```

For the `main.c` file you have to perform the same actions as for the **foo** project file. 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 *Files* section of the VS project file generated during the transformation process.

The last element in the family model is an object for the function **foo**. Create another 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 8. The file definition for foo.c



New Element

New File

Create a new file

Generic

Unique Name:	
Visible Name:	

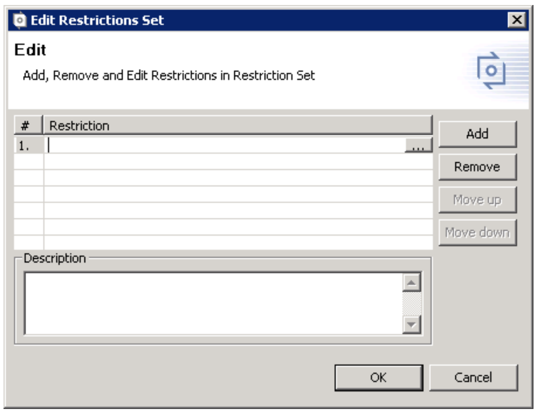
Attributes

file:	foo.c	☐ inherit
dir:	.	☑ inherit
type:	impl	
srcfile:	foosquare.c	☐ inherit
srcdir:		☐ inherit

Setting the *srcfile* attribute selects 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 then the source and destination name are equal and used from the *file* attribute. You have to set the source file name to "foosquare.c" which contains the square implementation. Press *Finish* to create the file element.

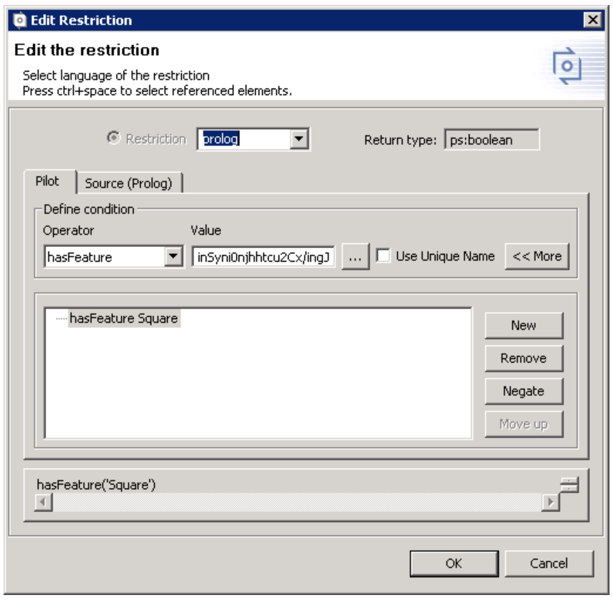
Because this implementation should used only if the feature **Square** is selected, you need to add a restriction to the attribute *scrfile*. Right click the *scrfile* attribute in the tree and select *New -> Restriction* from the context menu. In the dialog that opens a new restriction is already created and the input line for the restriction code is 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 9. Open the restriction pilot



Select the operation *hasFeature*. Now you have to choose the desired feature. Press the "**...**" button on the right side of the value field. Select the **Square** feature in the element selection dialog that opens by moving it to the right hand side. After closing the element selection dialog the restriction pilot shows the final restriction code in the lower part of the dialog.

Figure 10. The restriction pilot



Restriction: prolog
Return type: ps:boolean

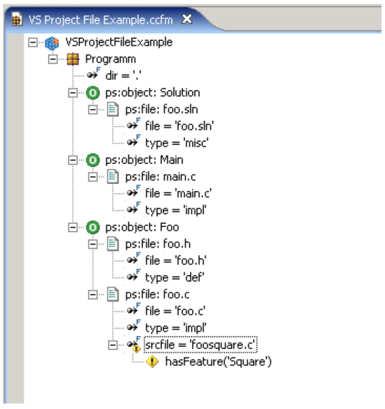
Operator: hasFeature
Value: inSyni0njhhtcu2Cx/ingJ

hasFeature Square

```
hasFeature('Square')
```

After clicking *OK* you get the model as shown in Figure 11. The *scrfile* attribute is restricted and will only appear if the **Square** feature is selected. If the restriction is not shown in the tree then open the *Show In Tree* context menu item again and select the *Restrictions* item. Save the model.

Figure 11. The final family model

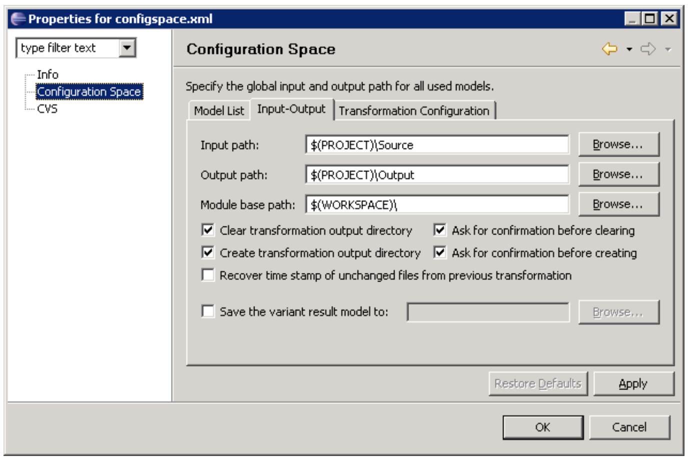

VS Project File Example.ccfm

```
VSProjectFileExample
  Programm
    dir = '.'
    ps:object: Solution
      ps:file: foo.sln
        file = 'foo.sln'
        type = 'misc'
    ps:object: Main
      ps:file: main.c
        file = 'main.c'
        type = 'impl'
    ps:object: Foo
      ps:file: foo.h
        file = 'foo.h'
        type = 'def'
      ps:file: foo.c
        file = 'foo.c'
        type = 'impl'
        srcfile = 'foosquare.c'
          hasFeature('Square')
```

6. Setting up the transformation

Before you perform a transformation you have to specify some options. Go to the *Variant Projects* view and select the **VS Project File Example** configuration space. Open the properties dialog and select the **Configuration Space** page. Switch to the *Input-Output* tab to set the input and output directories. Enter "$(PROJECT)\Source" into the input path field and "$(PROJECT\Output" into the output path field. Set all directory options as shown in Figure 12, "The input and output configuration".

Figure 12. The input and output configuration


Properties for configspace.xml

Configuration Space

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

Input path: $(PROJECT)\Source

Output path: $(PROJECT)\Output

Module base path: $(WORKSPACE)\

☑ Clear transformation output directory ☑ Ask for confirmation before clearing

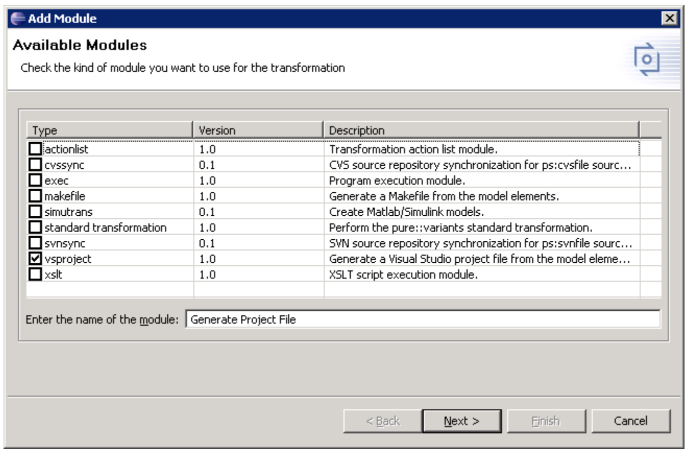
☑ Create transformation output directory ☑ Ask for confirmation before creating

☐ Recover time stamp of unchanged files from previous transformation

☐ Save the variant result model to:

In order to create a VS project file you need to enable the VS project transformation module. Switch to the *Transformation Configuration* tab and add a new module by pressing the *Add* button on the right side. In the dialog that opens check the **vcproject** module as in Figure 13 and enter "Generate Project File" as the module name.

Figure 13. Adding the VS project transformation module



Add Module

Available Modules

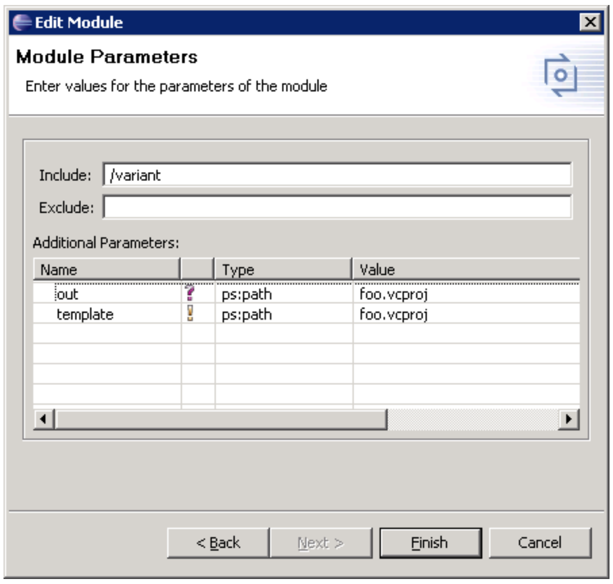
Check the kind of module you want to use for the transformation

Type	Version	Description
☐ actionlist	1.0	Transformation action list module.
☐ cvssync	0.1	CVS source repository synchronization for ps:cvsfile sourc…
☐ exec	1.0	Program execution module.
☐ makefile	1.0	Generate a Makefile from the model elements.
☐ simutrans	0.1	Create Matlab/Simulink models.
☐ standard transformation	1.0	Perform the pure::variants standard transformation.
☐ svnsync	0.1	SVN source repository synchronization for ps:svnfile sourc…
☑ vsproject	1.0	Generate a Visual Studio project file from the model eleme…
☐ xslt	1.0	XSLT script execution module.

Enter the name of the module: Generate Project File

Click the *Next* button to enter the module parameters. The VS project module has two parameters named *out* and *template*. The first parameter is the file name of the vs project file to be created during the transformation. Set the *out* parameter to "foo.vcproj". Set the value of the *template* parameter to "foo.vcproj".

Figure 14. The vs project module parameter dialog



Edit Module

Module Parameters

Enter values for the parameters of the module

Include: /variant

Exclude:

Additional Parameters:

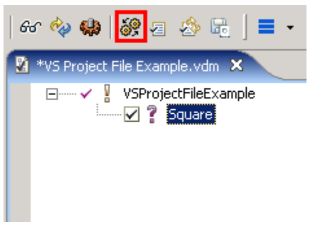
Name		Type	Value
out	?	ps:path	foo.vcproj
template	!	ps:path	foo.vcproj

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

7. Generating a variant

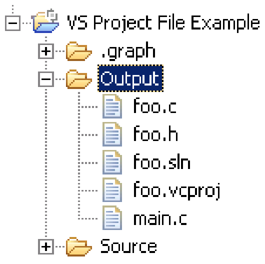
Before a transformation can be performed you must first create a variant. Open the variant model by double-clicking the `VS Project File Example.vdm` file in the configuration space. Select the **Square** feature and press the **Transform Model** button. Confirm the next messages.

Figure 15. Start the transformation



The transformation should generate the directory structure shown in Figure 16, "The resulting directory structure" (refresh the *Project View* if the output directory is not shown). The `foo.c` file should contain the implementation from the `foosquare.c` file.

Figure 16. The resulting directory structure



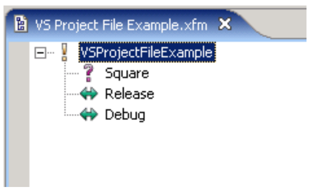
The `foo.vcproj` contains all files of the project.

Change the selection of the **Square** feature and perform 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 `foo.vcproj` in the **Output** directory with Visual Studio and *build* the solution. The program **foo** should be built.

8. Adding the build options

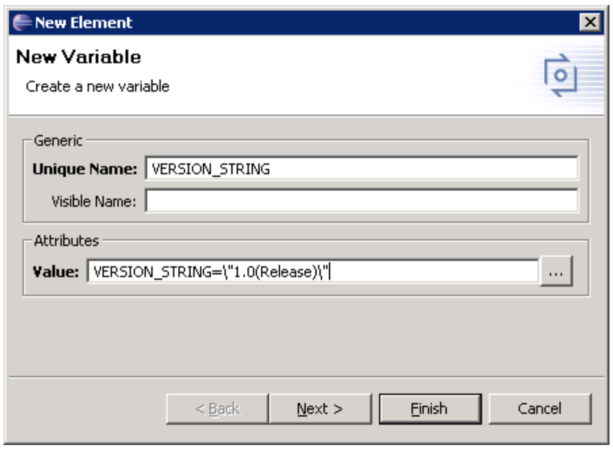
To configure the build options add two more alternative features into the feature model. The features are named "Release" and "Debug". The feature model should now look like Figure 17, "The feature model with build options".

Figure 17. The feature model with build options



You need to add a variable in the family model `VS Project File Example.ccfm`. Right click the **Program** component and select *New -> Variable*. In the dialog that opens enter "VERSION_STRING" as the *Unique Name*. The attribute **Value** should be set to "VERSION_STRING=\"1.0(Release)"". This value should be used for the **Release** build configuration. Click *Finish* button to create the variable.

Figure 18. Creating the VERSION_STRING variable


New Element

New Variable

Create a new variable

Generic

Unique Name: VERSION_STRING

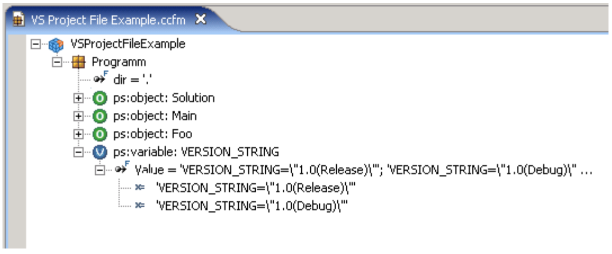
Visible Name:

Attributes

Value: VERSION_STRING=\"1.0(Release)\"

For the other configuration create another value for the attribute **Value**. Right click the Value attribute of the variable and select *New -> Attribute Value* in the context menu. In the dialog that opens enter "VERSION_STRING=\"1.0(Debug)"" for the **Debug** configuration. Now the family model should look like Figure 19, "The family model with build options".

Figure 19. The family model with build options


```
VS Project File Example.ccfm
VSProjectFileExample
  Programm
    dir = '.'
    ps:object: Solution
    ps:object: Main
    ps:object: Foo
    ps:variable: VERSION_STRING
      Value = 'VERSION_STRING=\"1.0(Release)\"'; 'VERSION_STRING=\"1.0(Debug)\" ...
        'VERSION_STRING=\"1.0(Release)\"'
        'VERSION_STRING=\"1.0(Debug)\"'
```

Now you need to add a restriction to every value. Right click the first value **VERSION_STRING=\"1.0(Release)** and select *New -> Restriction* from the context menu. Open the restriction pilot and create the following restriction:

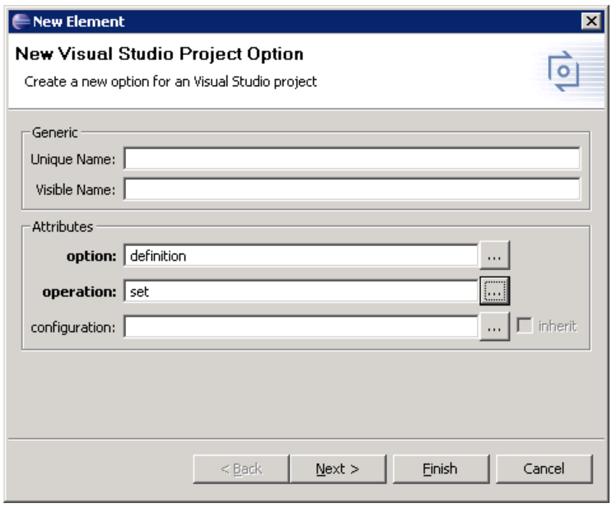
```
hasFeature('Release')
```

For the second value **VERSION_STRING=\"1.0(Debug)** we create the restriction:

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

As the last step you have to specify where the **VERSION_STRING** variable should be stored. For this create a *Visual Studio Project Option Element*. Right click the **VERSION_STRING** variable and select *VS Project Option* from the context menu. The *option* input field is **definition** and the *operation* is set to **set**.

Figure 20. The Visual Studio Project Option element


New Element

New Visual Studio Project Option

Create a new option for an Visual Studio project

Generic

Unique Name:

Visible Name:

Attributes

option: definition

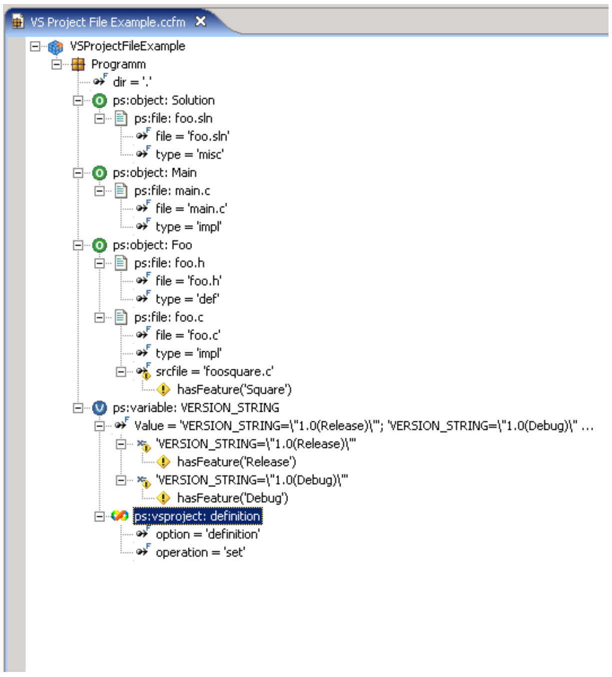
operation: set

configuration:

inherit

After pressing the *Finish* button the family model look like in the Figure 21, below.

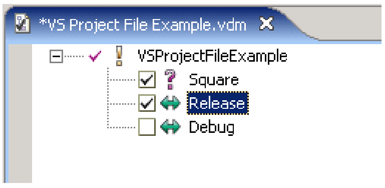
Figure 21. The final family model


```
VS Project File Example.ccfm
VSProjectFileExample
  Programm
    dir = '.'
    ps:object: Solution
      ps:file: foo.sln
        file = 'foo.sln'
        type = 'misc'
    ps:object: Main
      ps:file: main.c
        file = 'main.c'
        type = 'impl'
    ps:object: Foo
      ps:file: foo.h
        file = 'foo.h'
        type = 'def'
      ps:file: foo.c
        file = 'foo.c'
        type = 'impl'
        srcfile = 'foosquare.c'
          hasFeature('Square')
    ps:variable: VERSION_STRING
      Value = 'VERSION_STRING=\"1.0(Release)\"'; 'VERSION_STRING=\"1.0(Debug)\" ...
        'VERSION_STRING=\"1.0(Release)\"'
          hasFeature('Release')
        'VERSION_STRING=\"1.0(Debug)\"'
          hasFeature('Debug')
      ps:vsproject: definition
        option = 'definition'
        operation = 'set'
```

After the next transformation you have the preprocessor definition **VERSION_STRING** variable in the VS project file.

The value depends on the selection of the **Release** or **Debug** feature. To begin select the **Release** feature in the `VS Project File Example.vdm` and start the transformation.

Figure 22. Selected Release feature



After performing the transformation open the `foo.vcproj` file from **Output** directory with Visual Studio. In the next step build the project in the Visual Studio in **Release** mode. The built program will be found in the **Output\Release** directory. Now you can run the program with **value** say *3*.

```
foo.exe 3
```

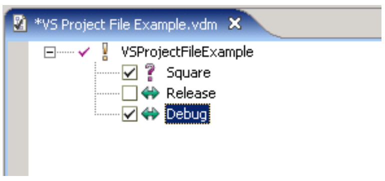
The output look like following code:

```
Running version 1.0(Release)
foo(3) = 9
```

The version is here **Release**.

To run the program in **Debug** mode select the **Debug** feature in the `VS Project File Example.vdm` and start the transformation again.

Figure 23. Selected Debug feature



The VS project file is in the **Output** directory and open `foo.vcproj` file with **Visual Studio** like for *Release* feature transformation. But now build the project with **Debug** mode in Visual Studio. The program to run is found in the **Output\Debug** directory. We run the program with **value** *3* again.

```
foo.exe 3
```

And the output looks like this:

```
Running version 1.0(Debug)
foo(3) = 9
```

We can see the version has changed to **Debug**.