Application Note: 251

Create Software Packs for Software Components

ARM® KEIL®
Microcontroller Tools

Abstract

This application note explains how to create a Software Pack for distributing software components to a target audience. It explains the basic steps for generating a Software Pack and demonstrates features such as dependency checking, file selection, variant creation and versioning of software components. How to add code templates and example projects to a Software Pack is also explained in detail.

Contents

What is a Software Pack?

Software Packs provide support for microcontroller devices and development boards, and can contain software components such as drivers and middleware, including example projects and code templates. The following types of Software Packs can be distinguished:

- **Device Family Pack (DFP)**: generated by a silicon supplier or tool vendor; provides support to create software applications for a specific target microcontroller. The creation of a DFP is subject of AN254: www.keil.com/appnotes/docs/apnt_254.asp.
- **Board Support Pack (BSP)**: published by a board vendor to support the peripheral hardware mounted on the board. How to create a BSP is discussed in AN255: www.keil.com/appnotes/docs/apnt_255.asp
- **CMSIS Pack**: provided by ARM® and includes support for CMSIS Core, DSP, and RTOS. It can be downloaded here: www.keil.com/dd2/pack.
- **Middleware Pack**: created by a silicon supplier, tool vendor or a third party; reduces development time by giving access to popular software components (such as software stacks, special hardware libraries, etc). There are various Middleware Packs already available at www.keil.com/dd2/pack.
- **In-house components**: developed by the tool user for internal or external distribution. This application note explains the steps that are necessary to create such a Software Pack.

A Software Pack is a ZIP archive (with the filename extension "PACK") containing all required libraries and files and a package description file (PDSC) with all the information about the Software Pack in XML format. The structure of a Software Pack is defined in CMSIS. Refer to CMSIS-Pack (www.keil.com/CMSIS/Pack) for more information. The following describes how to create a Software Pack from scratch.

After installation using the Pack Installer, you will find them for example in `C:\Keil\ARM\Pack\Keil`. AN256 describes the process of publishing a Software Pack to a wider audience: www.keil.com/appnotes/docs/apnt_256.asp.

AN252 "Product Lifecycle Management with Software Packs" gives you more insight into the benefits of Software Packs (www.keil.com/appnotes/docs/apnt_252.asp).

What is a Software Component?

Software components may contain

- Source code, libraries, header/configuration files and documentation.
- Complete example projects that show the usage of the software component and which can be downloaded and executed on evaluation hardware.
- Code templates that can be used as a starting point for using software components.

Application Note Collaterals

This application note provides a ZIP file (`apnt_251.zip`) containing five Software Packs in ZIP format:

`\MyVendor.MySimplePack.1.0.0.zip`:	Used in Example 1
`\MyVendor.MyConditionPack.1.0.0.zip`:	Used in Example 2
`\MyVendor.MyVariantsPack.1.0.0.zip`:	Used in Example 3
`\MyVendor.MyBundlePack.1.0.0.zip`:	Used in Example 4
`\MyVendor.MySimplePack.1.1.0.zip`:	Used in Example 5

The source files in the Packs are intentionally empty and fulfill demonstation purposes only. All Packs are fully functional in the way that they can be installed using **Pack Installer**. Also, a PDSC template file is included (vendor.pack_name.pdsc). The latest version of this PDF and the ZIP file can be found here: www.keil.com/appnotes/docs/apnt_251.asp.

Example 1: Create a Simple Software Pack

In this example several source code files that you have written should be distributed within an engineering group. You want to distribute them in a controlled manner to ensure overall consistency. This example assumes the `vendor` name "MyVendor", the package `name` "MySimplePack", and `version` 1.0.0.

The XML based package description file (PDSC) contains all required information about a Software Pack. In this example it is called `MyVendor.MySimplePack.pdsc`. It is starts with information about the XML version and the encoding:

```
<?xml version="1.0" encoding="utf-8"?>
<package schemaVersion="1.1" xmlns:xs="www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="PACK.xsd">
  <name>MySimplePack</name>
  <description>Internal Software Pack</description>
  <vendor>MyVendor</vendor>
  <url></url>
  <supportContact></supportContact>
```

The XML schema is defined in the `PACK.xsd` file that is part of the MDK installation (`C:\Keil\UV4`). It opens with any text editor and can be used for the validation of the PDSC file that you are about to generate.

The `<name>` and the `<vendor>` tags define the basics of the Pack and are also used for the file name of the PACK file. The description should contain some additional information on the Pack itself and/or its contents.

The `<url>` tag may contain an URL with a download link of the Pack. This can be any type of valid URL, for example localhost (for your own machine, if a web server is installed), 192.168.0.0 for a machine in your local intranet, or a machine that is available from the Internet, such as www.example.com. This makes it easy to supply users with updates as the Pack Installer checks the given URL for new versions of the Software Pack. For more information on how to publish a Software Pack to a wider audience refer to: www.keil.com/appnotes/docs/apnt_256.asp.

The `<supportContact>` tag is optional and may contain an URL or an email address to point the customer to relevant sources of support.

You need to specify a `<release>` section with a `<release version>` tag as this is used by the **Pack Installer** to identify the version of a Pack. This enables you to work with different versions.

```
<releases>
  <release version="1.0.0">
    Oct/15/2013, Initial version
  </release>
</releases>
```

After that, you may define keywords that are used for search indexing:

```
<keywords>
  <keyword>MyVendor</keyword>
  <keyword>My Software Component</keyword>
</keywords>
```

The `<taxonomy>` section describes your `Cclass` (see below) in more detail:

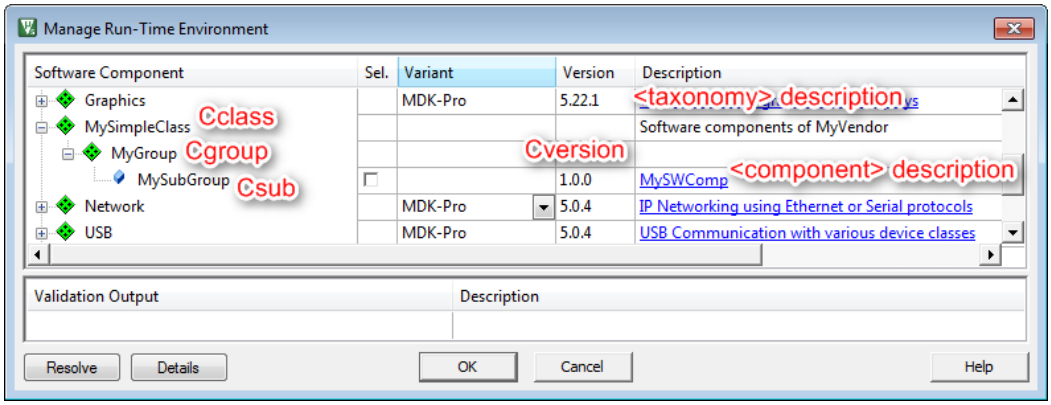
```
<taxonomy>
  <description Cclass="MySimpleClass">Software components of MyVendor</description>
</taxonomy>
```

Finally, you define your software component:

```
<components>
    <component Cclass="MySimpleClass" Cgroup="MyGroup" Csub="MySubGroup" Cversion="1.0.0">
      <description>MySWComp</description>
      <files>
        <file category="doc" name="Docs\MySWComp.htm"/>
        <file category="header" name="MySWComp\header_mylib.h"/>
        <file category="header" name="MySWComp\config_mylib.h" attr="config"/>
        <file category="source" name="MySWComp\mylib_one.c"/>
        <file category="source" name="MySWComp\mylib_two.c"/>
      </files>
    </component>
</components>
```

A software component needs the attributes component class (`Cclass`), component group (`Cgroup`), and a revision number called `Cversion`. Optionally a sub-component attribute (`Csub`) may provide an additional level of hierarchy. You may install multiple Software Components. Each component is therefore identified by its attributes `Cbundle`, `Cvendor`, `Cclass`, `Cgroup`, `Csub`, `Cversion` and `Cvariant`.

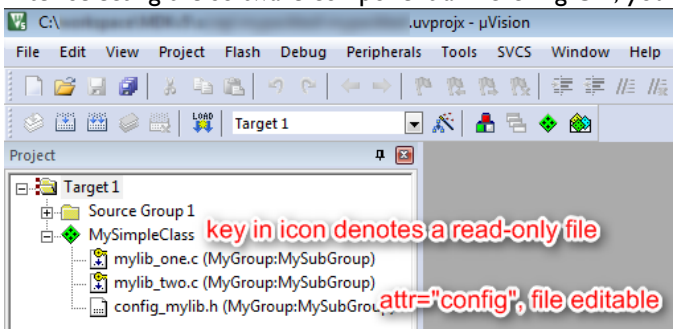
The information above is for example shown in the **Manage Run-Time Environment** window:



Selecting a component, source files of the section `<files>` are added to the project as described in the following table. The attributes `category` and `attr` specify how the file is added to the project.

category	attr	Project Use
doc		File stays in Pack folder; is used for documentation links, i.e. in the Manage Run-Time Environment window.
header		File stays in Pack folder; path is added to the Compiler #include search path which allows usage during project build
	config	Copied to the project for configuration purposes
	template	Copied to the project on demand using the Add New Items to Group functionality
image	template	Can be added to the project manually; will be then translated using FCARM.EXE.
library		File stays in Pack folder and is linked to project
object		File stays in Pack folder and is linked to project
source		File stays in Pack folder and is compiled and linked to project
	config	Copied to the project for configuration purposes
	template	Copied to the project on demand using the Add New Items to Group functionality

After selecting the software component and clicking OK, your project should look like this:

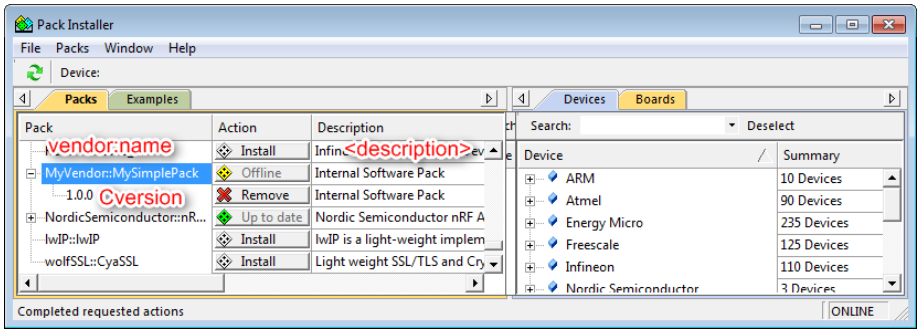


Generate and Install the Software Pack

There are a few steps that need to be taken to generate a Software Pack:

1. Save the PDSC file in the *vendor.name*.pdsc format (MyVendor.MySimplePack.pdsc).
2. Create a ZIP file in the *vendor.name.version*.zip format (MyVendor.MySimplePack.1.0.0.zip), using the folder structure as demonstrated in the application note's ZIP file.
3. Rename the ZIP file to MyVendor.MySimplePack.1.0.0.pack.
4. Double-click the file to install it.

After installation, the Software Pack appears in the Pack tab of the **Pack Installer**:



The Software Pack is automatically copied to two locations. In the directory `C:\Keil\ARM\Pack\.Downloads` you will find the PACK and the PDSC file. In the directory `C:\Keil\ARM\Pack\`*`vendor\name\version`* you will find the extracted files of the Software Pack.

Example 2: Create Conditions for Dependency Checking and File Selection

A condition describes dependencies on device, processor, and tool attributes as well as the presence of other components. Each condition has an `id` that is unique within the scope of the PDSC file. An `id` can be referenced in the `condition` attribute of components, APIs, examples, and files.

In this example, a CMSIS-RTOS compliant operating system needs to be present. Based on the Cortex-M class core, three different versions of a library are distributed:

- `mylib_cm0.lib` for use with a Cortex-M0 and Cortex-M0+ processor
- `mylib_cm3.lib` for use with a Cortex-M3 processor
- `mylib_cm4.lib` for use with a Cortex-M4 processor

Conditions are used to accomplish this:

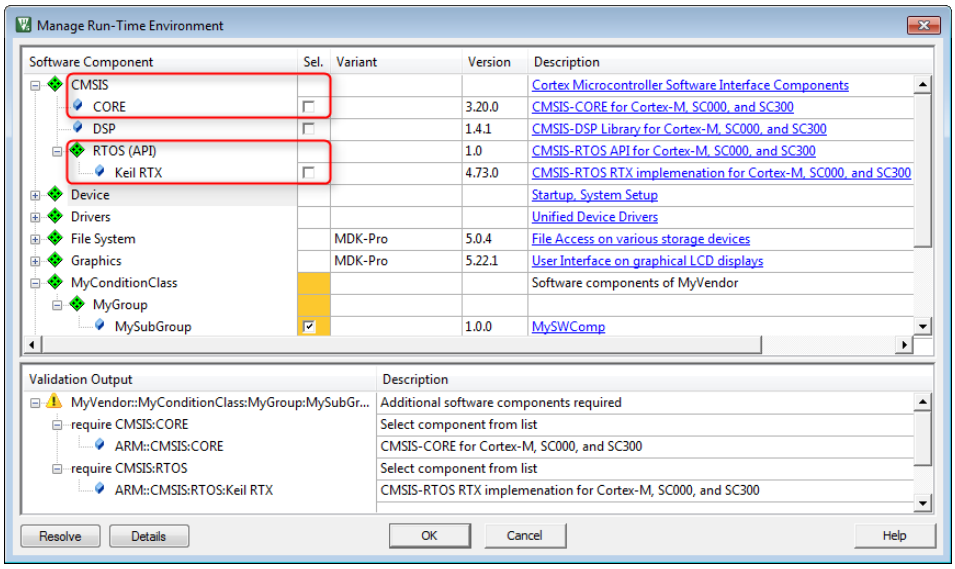
```
  <conditions>
    <condition id="CM0">
      <description>Cortex-M0 based device</description>
      <accept Dcore="Cortex-M0"/>
      <accept Dcore="Cortex-M0+"/>
    </condition>
    <condition id="CM3">
      <description>Cortex-M3 based device</description>
      <accept Dcore="Cortex-M3"/>
    </condition>
    <condition id="CM4">
      <description>Cortex-M4 based device</description>
      <accept Dcore="Cortex-M4"/>
    </condition>
    <condition id="CMSIS Core with RTOS">
      <description>CMSIS Core with RTOS for Cortex-M processor</description>
      <accept condition="CM0"/>
      <accept condition="CM3"/>
      <accept condition="CM4"/>
      <require Cclass="CMSIS" Cgroup="CORE"/>
      <require Cclass="CMSIS" Cgroup="RTOS"/>
    </condition>
  </conditions>
```

An `<accept>` tag gives an option to a condition. In this example, the condition "CMSIS Core with RTOS" is fulfilled, if a Cortex-M0, a Cortex-M0+, a Cortex-M3 or a Cortex-M4 is present. The `<require>` tag describes a condition that is required to be true for the condition to be fulfilled. Here, the software components CMSIS-Core and CMSIS-RTOS need to be present. For more information on the allowed conditions, refer to CMSIS-Pack - /package/conditions level.

In the following example, the `Cclass` "MyConditionClass" uses the condition "CMSIS Core with RTOS". The library files are added to the project depending on the Cortex-M processor used in the selected microcontroller device:

```
    <component Cclass="MyConditionClass" Cgroup="MyGroup" Csub="MySubGroup"
     Cversion="1.0.0" condition="CMSIS Core with RTOS">
      <description>MySWComp</description>
      <files>
        <file category="doc" name="Docs\MySWComp.htm"/>
        <file category="header" name="MySWComp\header_mylib.h"/>
        <file category="header" name="MySWComp\config_mylib.h" attr="config"/>
        <file category="source" name="MySWComp\mylib_one.c"/>
        <file category="source" name="MySWComp\mylib_two.c"/>
        <file category="library" condition="CM0" name="MySWComp\lib\mylib_cm0.lib"/>
        <file category="library" condition="CM3" name="MySWComp\lib\mylib_cm3.lib"/>
        <file category="library" condition="CM4" name="MySWComp\lib\mylib_cm4.lib"/>
      </files>
    </component>
```

The **Manage Run-Time Environment** window shows the new dependencies and asks the user to resolve them:



The **Validation Output** clearly identifies the component dependencies. Clicking on "ARM::CMSIS::CORE" takes you directly to the right entry in the Software Component list. If you select the CORE, you will see that this component has another dependency to "Keil::Device::Startup". All dependencies should be resolved before continuing.

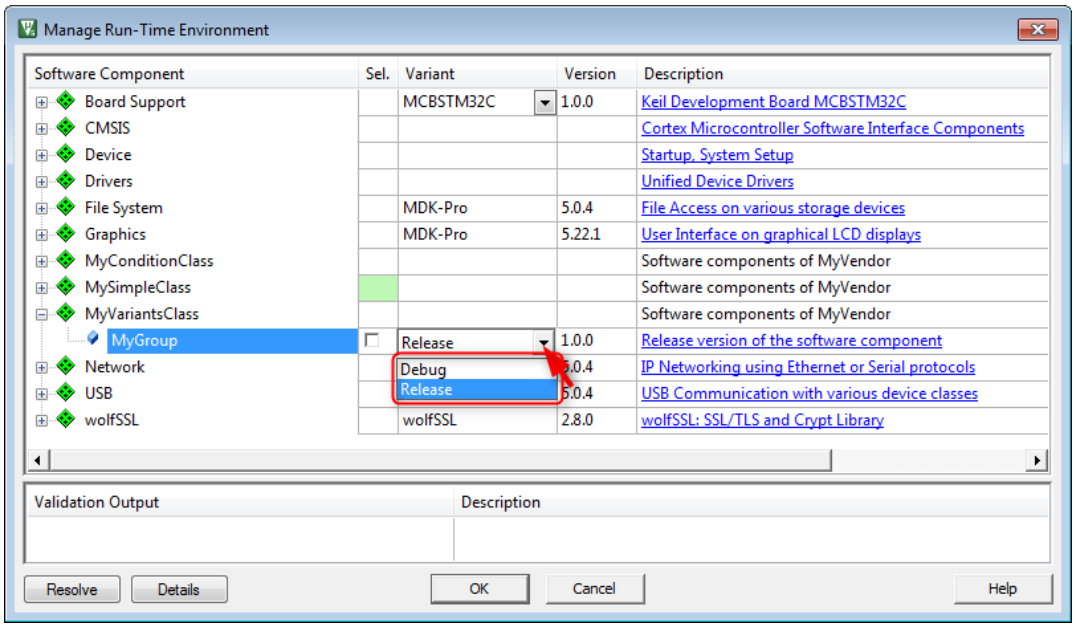
The **Resolve** button will try to check all required boxes for you. Sometimes, an automatic resolution is not possible. This is usually the case if you have a choice between different components (based on an `<accept>` condition). Such dependencies need to be resolved manually.

Example 3: Create Variants

This example creates a software component that consists of two (mutually exclusive) variants "Debug" and "Release". The attribute `Cvariant` allows defining different variant of the same `Cclass` and `Cgroup` component:

```
      <component Cclass="MyVariantsClass" Cgroup="MyGroup" Cvariant="Debug" Cversion="1.0.0"
       condition="CMSIS Core">
        <description>Debug version of the software component</description>
        <files>
          <file category="doc" name="Docs\Debug.htm"/>
          <file category="header" name="MySWComp\debug.h"/>
          <file category="header" name="MySWComp\config_debug.h" attr="config"/>
          <file category="source" name="MySWComp\debug.c"/>
          <file category="source" name="MySWComp\all_variants.c"/>
        </files>
      </component>
      <component Cclass="MyVariantsClass" Cgroup="MyGroup" Cvariant="Release"
       Cversion="1.0.0" condition="CMSIS Core">
        <description>Release version of the software component</description>
        <files>
          <file category="doc" name="Docs\Release.htm"/>
          <file category="header" name="MySWComp\release.h"/>
          <file category="header" name="MySWComp\config_release.h" attr="config"/>
          <file category="source" name="MySWComp\release.c"/>
          <file category="source" name="MySWComp\all_variants.c"/>
        </files>
      </component>
  </components>
```

The variants are selected using a drop-down menu in the **Manage Run-Time Environment** window:



Example 4: Create a Bundle

A bundle is a variant on the `Cclass` level. In this example, a bundle is created to add support for a custom microcontroller board, based on the STM32F2xx family. A condition checks for the right microcontroller device. The `Cclass` "Board Support" is used for the `Cbundle` "MyBoardSupport". The board support package consists of three software components: Board Setup, LED, and ADC. LED and ADC require Board Setup to work properly:

```
<conditions>
  <condition id="STM32F2xx">
    <description>STM32F2xx device required</description>
    <require Dfamily="STM32F2 Series" Dvendor="STMicroelectronics:13"/>
  </condition>
  <condition id="STM32F2xx Board Setup">
    <description>Board Setup Code required</description>
    <require condition="STM32F2xx"/>
    <require Cclass="Board Support" Cgroup="Board Setup"/>
    </condition>
</conditions>

<components>
  <bundle Cbundle="MyBoardSupport" Cclass="Board Support" Cversion="1.0.0">
    <description>MyVendor custom board support package</description>
    <doc>Docs\MyBSP.htm</doc>
    <component Cgroup="Board Setup" condition="STM32F2xx">
      <description>Custom board setup code</description>
      <files>
        <file category="doc" name="Docs\Setup.htm"/>
        <file category="header" name="MySWComp\setup.h"/>
        <file category="source" name="MySWComp\setup.c"/>
      </files>
    </component>
    <component Cgroup="I/O" Csub="LED" condition="STM32F2xx Board Setup">
      <description>LED code for custom board</description>
      <files>
        <file category="doc" name="Docs\LED.htm"/>
        <file category="header" name="MySWComp\led.h"/>
        <file category="header" name="MySWComp\config_led.h" attr="config"/>
        <file category="source" name="MySWComp\led.c"/>
      </files>
```

```
      </component>
      <component Cgroup="I/O" Csub="ADC" condition="STM32F2xx Board Setup">
        <description>ADC code for custom board</description>
        <files>
          <file category="doc" name="Docs\ADC.htm"/>
          <file category="header" name="MySWComp\adc.h"/>
          <file category="header" name="MySWComp\config_adc.h" attr="config"/>
          <file category="source" name="MySWComp\adc.c"/>
        </files>
      </component>
    </bundle>
  </components>
```

Example 5: Versions, Project Examples, and User Code Templates

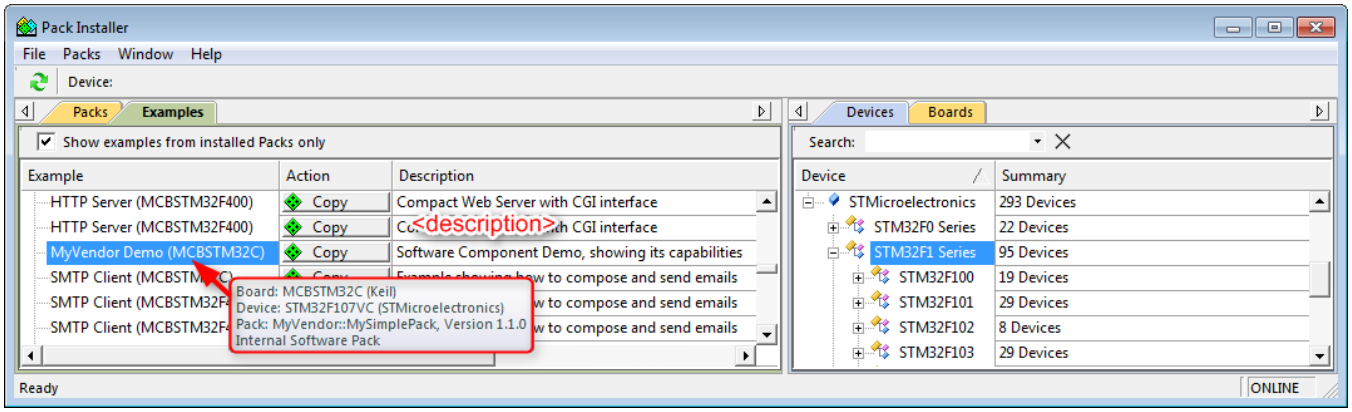
Add Projects Examples to a Software Pack

An example project is added using the `<examples>` section in the PDSC file:

```
  <examples>
    <example name="MyVendor Demo" doc="Abstract.txt" folder="MySWComp\demo">
      <description>Software Component Demo, showing its capabilities</description>
      <board name="MCBSTM32C" vendor="Keil"/>
      <project>
        <environment name="uv" load="MySWComp.uvprojx"/>
      </project>
      <attributes>
        <category>MyCategory</category>
      </attributes>
    </example>
  </examples>
```

You need to specify the `<name>` attribute of the example, a `<doc>` file for documentation, and the project `<folder>` within the Software Pack. The `<vendor>` and `<name>` tag of a Software Pack and the `name`, and `vendor` attributes of the `<board>` tag are required and used to generate the unique ID for the example. The `<attributes>` tag may contain information about the software components that are used in the example. A `<category>` tag can be used to filter the list of example projects.

Pack Installer's Examples tab shows the example. A tooltip is used to show more details about the example project, such as required development board, device, and the version of the Software Pack:

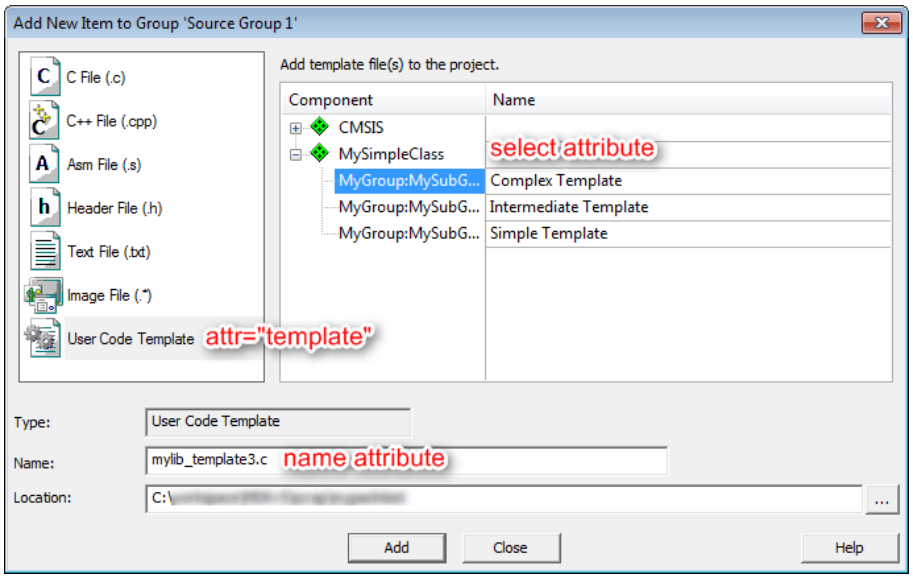


Add a User Code Template to a Software Pack

User code templates help to better understand basic concepts of a software component and to give the user a quick start for implementing his application. They can be added to a Pack using the file `category` "source" and the `attr` "template":

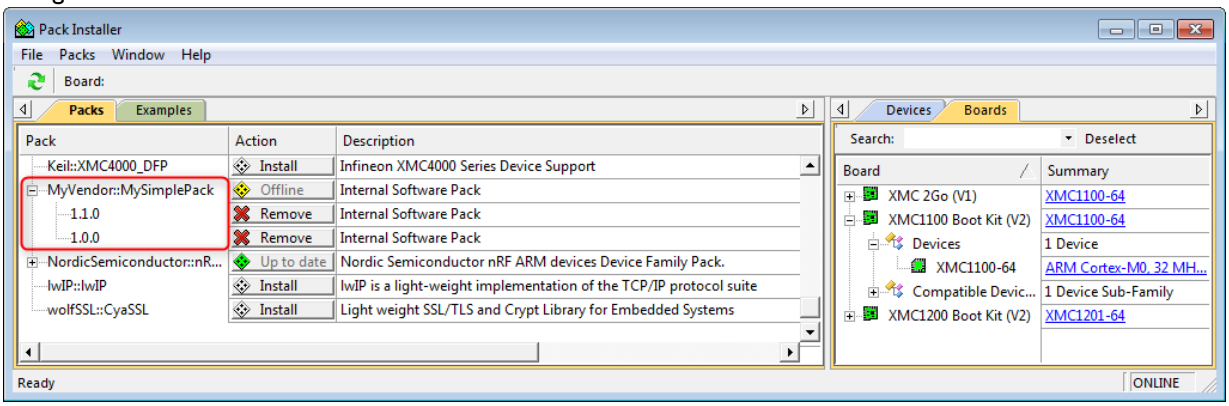
```
<component Cclass="MySimpleClass" Cgroup="MyGroup" Csub="MySubGroup" Cversion="1.1.0">
  <description>MySWComp</description>
  <files>
    <file category="doc" name="Docs\MySWComp.htm"/>
    <file category="header" name="MySWComp\header_mylib.h"/>
    <file category="header" name="MySWComp\config_mylib.h" attr="config"/>
    <file category="source" name="MySWComp\mylib_one.c"/>
    <file category="source" name="MySWComp\mylib_two.c"/>
    <file category="source" name="MySWComp\mylib_template1.c" attr="template"
     select="Simple Template"/>
    <file category="source" name="MySWComp\mylib_template2.c" attr="template"
     select="Intermediate Template"/>
    <file category="source" name="MySWComp\mylib_template3.c" attr="template"
     select="Complex Template"/>
  </files>
</component>
```

You can specify a `select` attribute that is used to identify the template in the **Add New Item to Group** window:

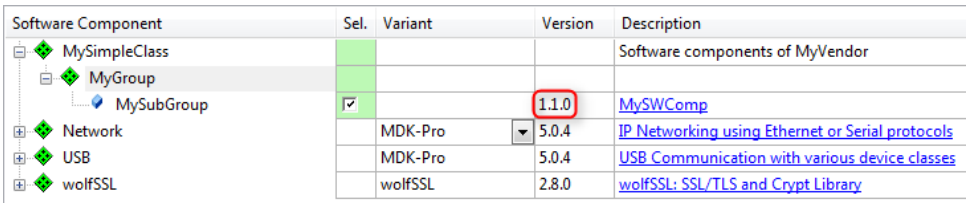


Versioning of a Software Pack

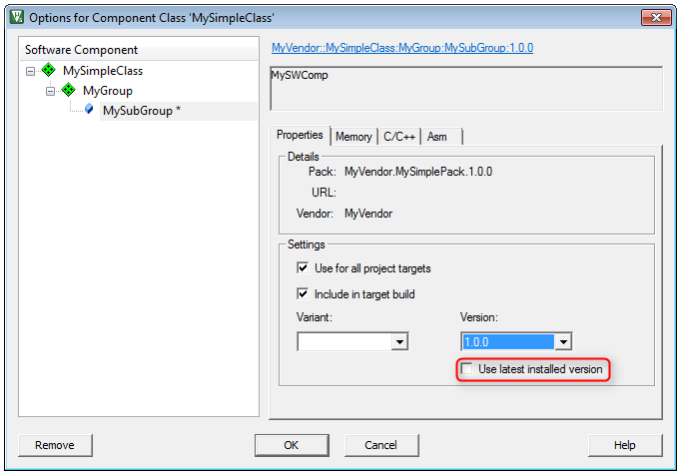
This example's PDSC file is based on the first example. To reflect the changes, the version number is increased. Versioning helps to distinguish between different versions of the same Software Pack. **Pack Installer** shows the version in the Pack tab:



The Version column in the **Manage Run-Time Environment** window also shows the new number:



At some point, the user might want to stick to one particular version of a software component. This can be accomplished using the **Options for Component Class** dialog. Right-click on any software component in the **Project** window and a similar dialog appears:



Unchecking the **Use latest installed version** checkbox lets you select which version of the software component to use in the project. If the software component has multiple variants available, the **Variant** drop-down box lets you select one of them for further use in the project.

Helpful Files and Tools

Here's a list of files and tools that help you to successfully generate a PDSC file and the complete BSP:

Tool	Purpose
Notepad++	Helps to create a XML-based PDSC file. Can check the syntax automatically (if XML file and XSD file reside in the same directory).
Visual Studio (Express)	Same as Notepad++
PACK.xsd	Schema file for PDSC XML syntax checking. Resides in the MDK installation directory (for example `C:\Keil\UV4`).

Conclusion

This application note shows the benefits of using Software Packs when it is applied to Software Components. You can easily supply source and object code. It shows that various different library version, for example for different processor cores can be selected from project context (in this case the microcontroller that is chosen).

Software Packs allow also to bundle multiple components, supply different variants of a Software Component and provide related documentation, example projects, and user code templates.

Revision History

- March 2014: New tags added, Pack Installer screenshots updated to reflect MDK v5.10
- January 2014: Links to other AN's added, minor corrections
- November 2013: Initial Version