
pure::variants Transformer for Software Configuration Management Manual

pure-systems GmbH

Copyright © 2003-2007 pure-systems GmbH

2007

Table of Contents

1. Synopsis	1
2. Concept	1
2.1. Client Authorization	2
3. Installing the Transformer	3
4. Using the Transformer	3
4.1. Transformer Configuration	4
4.2. ps:scmfile Source Element	5
5. Implementing a Wrapper	7
5.1. Environment Variables	7
5.2. Calculating File Locations	8
5.3. Example Wrapper Program	9
6. Supported pure::variants Editions	10

1. Synopsis

The pure::variants Transformer for Software Configuration Management Systems (SCMS) is used to synchronize local files used in variant transformations with files from a Software Configuration Management repository.

This extension provides a generic transformation module using a SCMS specific wrapper program to connect pure::variants with the SCMS client.



Note

For the popular SCM systems CVS and Subversion special transformers are available, i.e. the pure::variants Transformer for CVS and the pure::variants Transformer for Subversion.

This document describes how the transformer is used and how the interface of the SCMS specific program has to be implemented. Additionally it provides an example of a wrapper for the popular CVS SCMS.

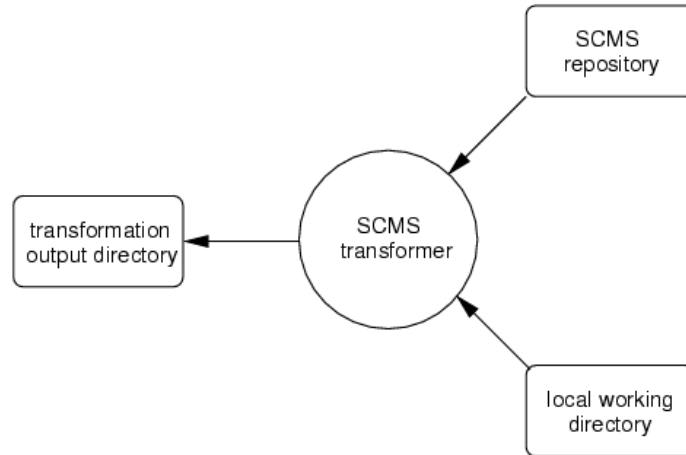
A printable version of this document is [available](#).

2. Concept

The purpose of this transformer is to synchronize files in the transformation output directory with corresponding files from a local or remote SCMS repository in the following way.

SCMS managed files are modeled using `ps:scmfile` source elements. If a such modeled file does not yet exist in the transformation output directory it is checked out from the SCMS repository. If a local SCMS repository (local working directory) is given, the file is first searched there. Otherwise the file is checked out from the remote SCMS repository.

Figure 1. SCMS Synchronization



If a file already exists in the transformation output directory it is synchronized with the corresponding file in the SCMS repository according to its revision and change state. Again it is first searched in the local working directory if given.

To synchronize files with various SCM systems, the transformer uses an SCMS specific wrapper program for the actual communication with the SCMS. The implementer of this wrapper program is responsible for implementing the above algorithm. It is within the discretion of the implementer what functionality is provided by the wrapper and what by the SCMS client. If for instance a specific SCMS does not support local working directories, the implementer of the wrapper program could rebuild this or a similar functionality in the wrapper.



Note

Please ask the implementer of the wrapper program if some functionality is missing or what other functionality is implemented by the wrapper and the specific SCMS.

2.1. Client Authorization

The SCMS transformer relies on external authorization of the SCMS client. If during the synchronization process the SCMS client program requests a username and/or password interactively, it is therefor either not possible to use the SCMS transformer or the wrapper program has to provide a functionality to pass the required authorization information to the SCMS client (if the client itself does not provide a corresponding functionality).



Note

Please ask the implementer of the wrapper program or consult the documentation of the SCMS to get information about how to provide authorization information for the SCMS client if required.

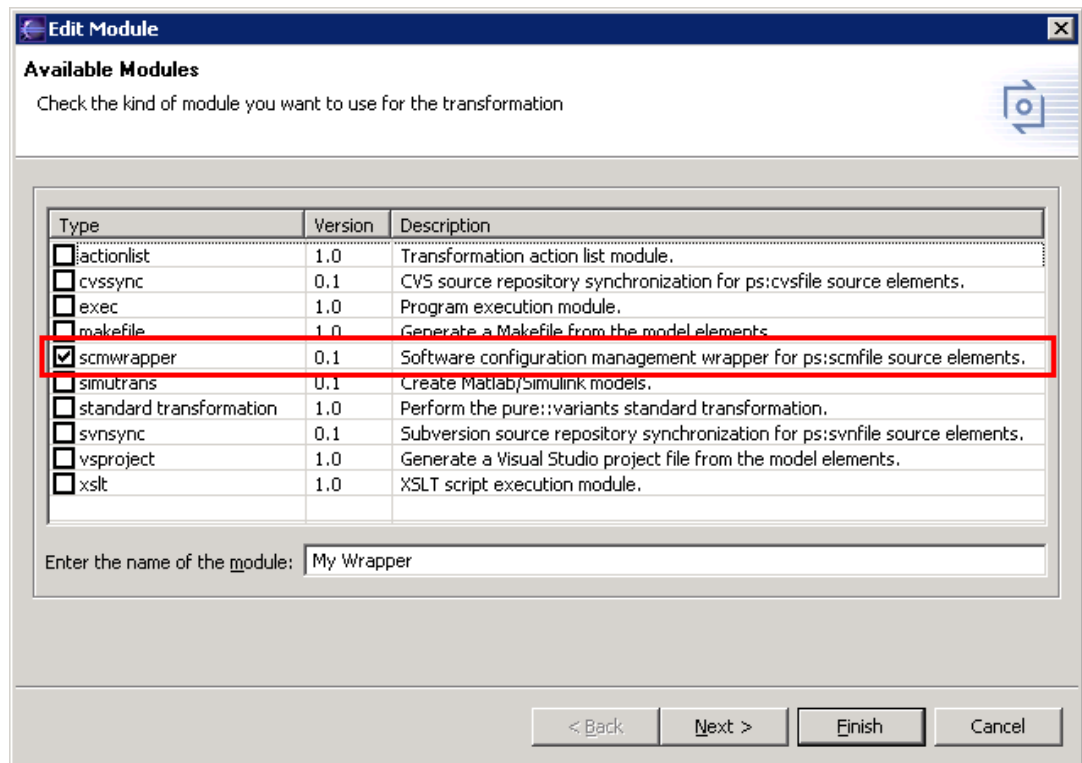
3. Installing the Transformer

The pure::variants Transformer for Software Configuration Management is available on the [pure::variants Update Manager Site for Eclipse](#). Please refer to this site for more information about how to install the transformer using the Eclipse Update Manager.

4. Using the Transformer

After installing the transformer a new transformation module is available called "scmwrapper". To see if the transformer is available open the properties of a configuration space by choosing **Properties** from the context menu of the configuration space. On the **Configuration Space** page change to the **Transformation Configuration** tab and then click on button **Add**. This opens a dialog showing the list of available transformation modules. It has to contain the "scmwrapper" module as shown in [Figure 2, "Available Transformation Modules"](#).

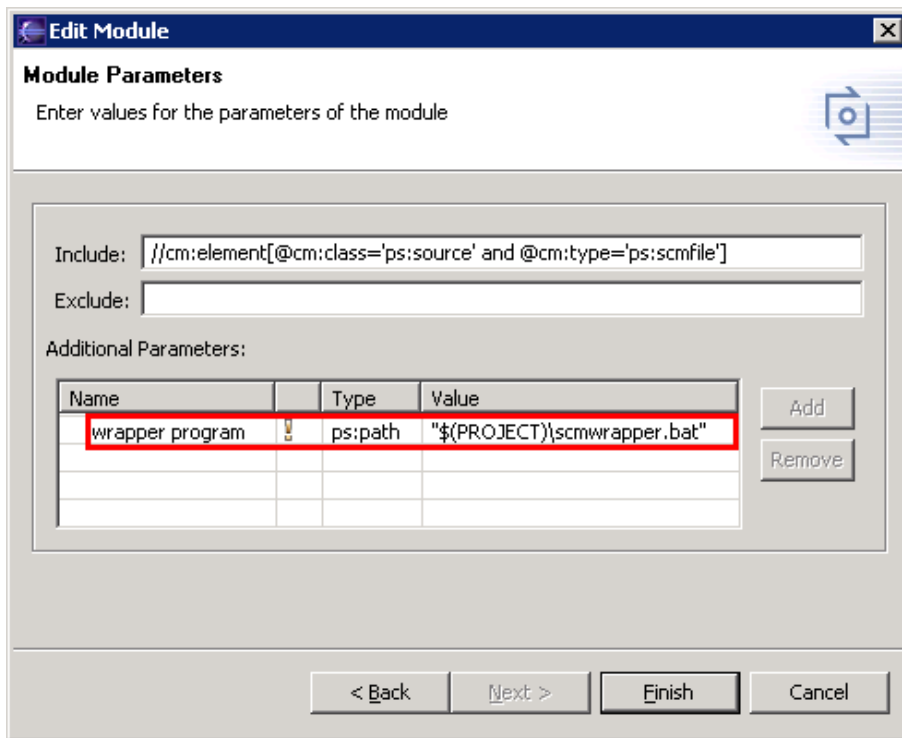
Figure 2. Available Transformation Modules



4.1. Transformer Configuration

As described above the SCMS transformer uses a wrapper program for the actual file synchronization. The command line to invoke this wrapper program has to be entered on the configuration page of the transformer. Open the list of available transformation modules as described in the beginning of this chapter. Ensure that the check box of the module "scm-wrapper" is checked. Then click on button **Next** to open the configuration page of the SCMS transformer (see [Figure 3, "SCMS Transformer Configuration"](#)).

Figure 3. SCMS Transformer Configuration

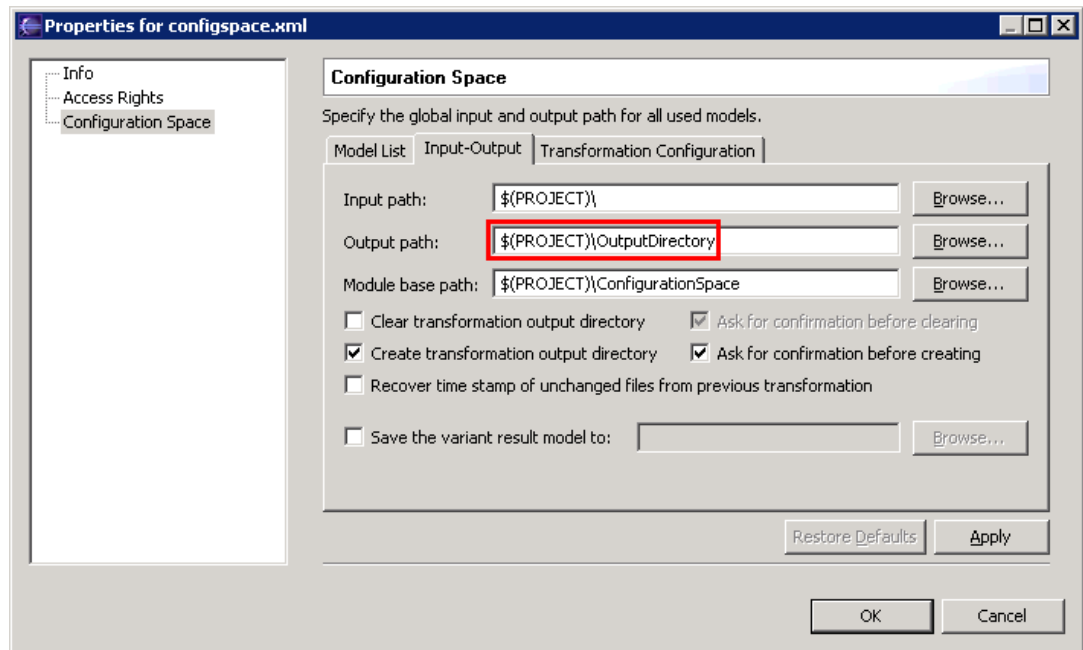


The "scmwrapper" module has exactly one parameter.

wrapper program The command line for the wrapper program call. The wrapper program is executed for each `ps:scmfile` source element in the input models. The information about the current `ps:scmfile` source element is passed to the wrapper using environment variables.

For a valid transformation configuration a transformation output directory has to be specified. Into this directory the files from the SCMS repository are written. The output directory is specified on the **Input-Output** page of the configuration space properties dialog (see [Figure 4, "Transformation Output Directory"](#)).

Figure 4. Transformation Output Directory



For more information on how to setup a transformation see section "Transformation Configuration Page" in chapter "5.3.7. Configuration Space Editor" of the pure::variants Eclipse Plugin User's Guide.

4.2. ps:scmfile Source Element

SCMS managed files are modeled using `ps:scmfile` source elements. The attributes of this source element are listed below. These attributes represent a common set of the information needed by an SCMS to synchronize a file. For specific SCMS systems some of the attributes may have no meaning or may have a slightly different meaning as for other SCMS systems. Thus only the attributes for specifying the SCMS repository location and the name and path of the file to synchronize are mandatory.



Note

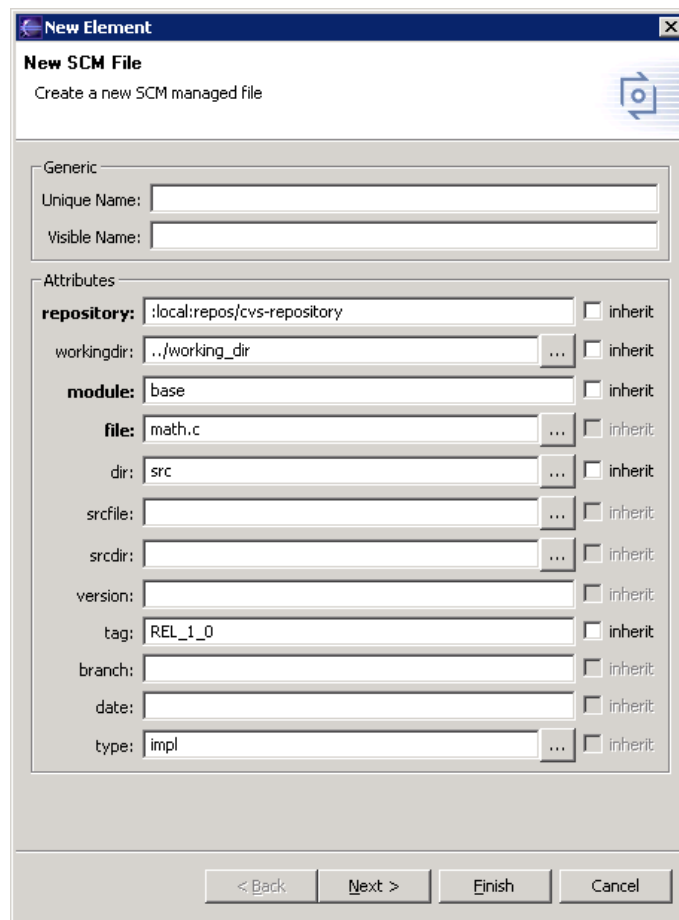
Please consult the documentation of the SCMS or ask the implementer of the wrapper program for the SCMS specific meaning of these attributes.

repository	[Mandatory] The SCMS repository containing the file.
dir	[Mandatory] The directory part of the SCMS managed file relative to the given SCMS module.
file	[Mandatory] The file name part of the SCMS managed file.
srcdir	[Optional] The directory part of the SCMS managed file relative to the given module. To be used if the source directory location differs from the target directory location.

srcfile	[Optional] The file part of the SCMS managed file. To be used if the source file name differs from the target file name.
type	[Optional] The type of the file according to attribute "type" of the <code>ps:file</code> source element.
module	[Optional] The name of the module in the SCMS repository containing the file.
branch	[Optional] The name of the SCMS repository branch containing the file.
tag	[Optional] The tag of the SCMS managed file.
version	[Optional] The version of the SCMS managed file.
date	[Optional] The date specifying the version of the SCMS managed file.
workingdir	[Optional] The local working directory containing the local copies of the files managed in the SCMS repository.

To create a new `ps:scmfile` source element choose **New->SCM File** from the context menu of a part element. This opens the **New SCM File** wizard as shown in [Figure 5, "New SCM File Wizard"](#).

Figure 5. New SCM File Wizard



After setting up the transformation and the input models the SCMS synchronization of the modeled files is started by performing a variant transformation.

A complete example of the use of the pure::variants Transformer for Software Configuration Management is presented by the "SCM Module" variant management example project. This example project can be installed in the current workspace by choosing **New->Example** from the context menu of the Variant Projects view and then Examples->Variant Management->SCM Module.

5. Implementing a Wrapper

The wrapper program is responsible for realizing the synchronization of a file with an SCMS repository (local and/or remote).

During the variant transformation the wrapper is called for every `ps:scmfile` source element of the input models. The information about the current `ps:scmfile` source element, i.e. the values of its attributes, and the transformation input and output directories are passed to the wrapper using environment variables. With this information the wrapper can construct a suitable SCMS client call to synchronize the file it was called for.

The following tasks have to be performed by the wrapper.

1. Prepare the transformation output directory. For instance if it does not exist, checking out files to it may fail depending on whether the SCMS client expects the target directory to be present or not.
2. Calculate the location of the source file in the SCMS repository and the location of the target file in the transformation output directory. If necessary also calculate the location of the source file in the local working directory.
3. Call the specific SCMS client.

If needed the wrapper also has to interact with the user or has to implement functionality not provided by the SCMS client. For instance if the SCMS client program does not write the target file by itself, the wrapper has to save the checked out file to the calculated target file location. Or a SCMS could not support local working directories. In this case the wrapper program could provide the missing functionality if desired. But this depends on the wrapped SCMS client and may strongly differ for different SCM systems.

5.1. Environment Variables

The following environment variables are set before the SCMS wrapper program is executed.

SCM_INPUTDIR	The path to the transformation input directory.
SCM_OUTPUTDIR	The path to the transformation output directory.
SCM_MODULE	The value of the 'module' attribute of the current <code>ps:scmfile</code> source element.
SCM_DIR	The value of the 'dir' attribute of the current <code>ps:scmfile</code> source element.

SCM_FILE	The value of the 'file' attribute of the current <code>ps:scmfile</code> source element.
SCM_SRCDIR	The value of the 'srcdir' attribute of the current <code>ps:scmfile</code> source element.
SCM_SRCFILE	The value of the 'srcfile' attribute of the current <code>ps:scmfile</code> source element.
SCM_BRANCH	The value of the 'branch' attribute of the current <code>ps:scmfile</code> source element.
SCM_TAG	The value of the 'tag' attribute of the current <code>ps:scmfile</code> source element.
SCM_VERSION	The value of the 'version' attribute of the current <code>ps:scmfile</code> source element.
SCM_DATE	The value of the 'date' attribute of the current <code>ps:scmfile</code> source element.
SCM_REPOSITORY	The value of the 'repository' attribute of the current <code>ps:scmfile</code> source element.
SCM_WORKINGDIR	The value of the 'workingdir' attribute of the current <code>ps:scmfile</code> source element.

5.2. Calculating File Locations

An essential task of the wrapper is to calculate the correct locations of the source and target files.

The target files have to be written to the transformation output directory. The path to this directory is set in the variable `SCM_OUTPUTDIR`. The location of the file in the output directory is calculated by appending the name (variable `SCM_FILE`) and directory (variable `SCM_DIR`) of the file as follows:

```
SCM_OUTPUTDIR + <path_separator> + SCM_DIR + <path_separator> + SCM_FILE
```

The path separator is for instance a backslash on Windows (e.g. `dir\file.txt`) and a slash on Linux (e.g. `dir/file.txt`). For an SCMS supporting modules (like CVS) it may be necessary to include the module name (variable `SCM_MODULE`) in the target path, e.g.

```
SCM_OUTPUTDIR + <path_separator> + SCM_MODULE + <path_separator> + SCM_DIR +  
<path_separator> + SCM_FILE
```

The calculation of the source file location is slightly different from the calculation of the target file location since it may have to consider revision information.

If the name of the source and target file is identical, and thus `SCM_SRCFILE` and `SCM_SRCDIR` are empty, the name of the source file is as follows.

```
SCM_DIR + <path_separator> + SCM_FILE
```

If the name or location of the source and target file is different, `SCM_SRCFILE` resp. `SCM_SRCDIR` has to be used instead of `SCM_FILE` resp. `SCM_DIR`. E.g. if the directory part differs, then the source file location is as follows.

```
SCM_SRCDIR + <path_separator> + SCM_FILE
```

If a local working directory is given, then the path to the working directory (variable `SCM_WORKINGDIR`) is prepended.

```
SCM_WORKINGDIR + <path_separator> + SCM_DIR + <path_separator> + SCM_FILE
```

Depending on the SCMS, revision and module information has to be added to the source file path. Please consult the documentation of the specific SCMS to find out how files in the SCMS repository are addressed.

5.3. Example Wrapper Program

The following batch file is an example wrapper for the CVS SCMS. It supports revision information (date, version, tag, and branch) and modules. Local working directories are not supported by this CVS wrapper.

This wrapper is part of the "SCM Module" example project. This example project can be installed in the current workspace by choosing **New->Example** from the context menu of the Variant Projects view and then Examples->Variant Management->SCM Module.

For every `ps:scmfile` source element of the input models it performs a checkout operation on the given CVS repository for the file described by the `ps:scmfile` source element. For this purpose it uses the environment variables listed above to calculate the location and name of the source file in the CVS repository and the location and name of the target file in the local repository. Then it calls the CVS client with the calculated arguments and writes the checked out file to the target file location.

First the variable `client` is set to the full path of the CVS client program that is used later in the script to perform the actual checkout operation.

```
REM path to cvs client
set client="C:\Program Files\WinCvs 1.2\cvs.exe"
```

In the next step the revision information of the source file is collected. If a date is given, then this date is taken as the revision. Otherwise a given version, tag, or branch is taken, checked in this order. If no revision can be determined, the newest revision of the source file in the SCMS repository will be checked out.

```
REM get file revision
set revision=
if not "%SCM_DATE%"==" " (
  set revision=-D %SCM_DATE%
) else (
  if not "%SCM_VERSION%"==" " (
    set revision=-r %SCM_VERSION%
  ) else (
    if not "%SCM_TAG%"==" " (
      set revision=-r %SCM_TAG%
    ) else (
      if not "%SCM_BRANCH%"==" " set revision=-r %SCM_BRANCH%
```

```
) )  
) )  
) )
```

The next code blocks calculate the complete path to the source file as described above. The name of the source file is taken from the variable `SCM_SRCFILE` if not empty, otherwise from the variable `SCM_FILE`. The directory part is taken from the variable `SCM_SRCDIR` if not empty, otherwise from the variable `SCM_DIR`. Since CVS expects slashes as path separators, all backslashes in the path are replaced by slashes. Then the complete path is constructed using the file name, the directory part, and, if given, the module name taken from the variable `SCM_MODULE`.

```
REM get source file  
set sourcefile=%SCM_FILE%  
if not "%SCM_SRCFILE%"==" " set sourcefile=%SCM_SRCFILE%  
  
REM get source file directory (replace backslashes with slashes, required by  
cvs)  
set sourcedir=%SCM_DIR:\=%  
if not "%SCM_SRCDIR%"==" " set sourcedir=%SCM_SRCDIR:\=%  
  
REM remove trailing separator  
if "%sourcedir:~-1%"=="\" set sourcedir=%sourcedir:~0,-1%  
  
REM build complete source file path  
if not "%sourcedir%"==" " set sourcefile=%sourcedir%/%sourcefile%  
if not "%SCM_MODULE%"==" " set sourcefile=%SCM_MODULE%/%sourcefile%
```

After calculating the source file path the next code blocks calculate the complete path to the target file. The directory part of the target file path is taken from the variable `SCM_DIR`. Here slashes in the directory part are replaced by backslashes because the target file is located in the local file system using backslashes as path separators. As described above, the target file has to be written to the transformation output directory. Using the output directory (variable `SCM_OUTPUTDIR`), the directory part of the target file path, and the target file name (variable `SCM_FILE`), the full path to the target file is constructed.

```
REM get target file  
set targetdir=%SCM_DIR:/=%  
  
REM remove trailing separator  
if "%targetdir:~-1%"=="\" set targetdir=%targetdir:~0,-1%  
set targetdir=%SCM_OUTPUTDIR%\%targetdir%  
  
REM build complete target file path  
set targetfile=%targetdir%\%SCM_FILE%
```

To be able to write the target file, it has to be ensured that the target directory exists. It is created if it doesn't exist.

```
REM create the target directory  
if not exist "%targetdir%" mkdir "%targetdir%"
```

After calculating the source and target file locations and the revision of the source file to check out, the last step is to call the CVS client. The CVS client gets the repository location, the revision, and the source file path as input and writes the checked out file to stdout which is redirected to the target file location.

```
REM call cvs client  
%client% -d "%SCM_REPOSITORY%" co -n -p %revision% "%sourcefile%" >  
"%targetfile%"
```

6. Supported pure::variants Editions

This plugin works only in a full-featured pure::variants and pure::variants (Evaluation).
