The svn-prov package

Use SVN Id keywords for package, class and file header

Martin Scharrer
martin@scharrer-online.de

Version v2.1747 - 2010/03/25

1 Introduction

This package is directed to authors of LaTeX packages and classes which use the version control software Subversion[1] (SVN) for their source files. It introduces three macros which are Subversion variants of the standard LaTeX header macros `\ProvidesPackage`, `\ProvidesClass` and `\ProvidesFile` which are used to identify package, class and other files, respectively. Instead of providing the package/class/file name and date manually they are extracted from a Subversion Id keywords string which is updated automatically by every time the source file is committed to the repository.

A similar package exists for RCS, the pre-predecessor of Subversion, in the `pgf`[2] bundle which is called `pgfrcs`. For further support for Subversion keywords see the author's other package `svn-multi`[3].

2 Usage

The following macros need an Id keyword which can initially be written as '`$Id:$`' and will be expanded by Subversion into the following format at the next commit:

`$Id:` ⟨*filename*⟩ ⟨*revision*⟩ ⟨*date*⟩ ⟨*time*⟩ ⟨*author*⟩ `$`

e.g. for the source file of this document:

```
$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $
```

For this to work the Subversion *property* `svn:keywords` must be set to (at least) '`Id`' for the source file(s). e.g. using the command line:

`svn propset 'svn:keyword' 'Id'` ⟨*filename(s)*⟩

More information about using Subversion in the LaTeX workflow can be found in the PracTeX Journal issue 2007-3[4].

[1] WWW: `http://subversion.tigris.org/`
[2] CTAN: `http://tug.ctan.org/pkg/pgf`
[3] CTAN: `http://tug.ctan.org/pkg/svn-multi`
[4] URL: `http://www.tug.org/pracjourn/2007-3/{skiadas-svn|ziegenhagen|scharrer}`

`\ProvidesPackageSVN` [⟨*file name*⟩]{$Id: ... $}[⟨*Package Information (version, description)*⟩]
`\ProvidesClassSVN` [⟨*file name*⟩]{$Id: ... $}[⟨*Class Information (version, description)*⟩]
`\ProvidesFileSVN` [⟨*file name*⟩]{$Id: ... $}[⟨*File Information (version, description)*⟩].

All of these macros await a valid Subversion Id keyword string as a mandatory argument. The file name and date is extracted from this string. For cases when the file source is not stored in the correct file but packed inside a different one, like a `.dtx` file, the correct file name can be provided by an optional argument. Because the file extension of package and class files is predefined and therefore ignored this is not needed for them when they are packed inside a corresponding `.dtx` file, i.e. one with the same base name.

As with the standard macros mentioned above an optional argument can be given afterwards which contains additional information (date, version, description) of the package, class or file. However, the SVN macros automatically insert the date, so only a version number and a short description should be given. If this argument is not given a default information string is used which is shown below as `\revinfo`.

Both optional arguments can include the following macros which are only valid inside them, but not afterwards:

`\rev` File revision.

`\Rev` File revision followed by a space.

`\revinfo` The default information used: "`(SVN Rev: ⟨revision⟩)`".

`\filebase` File base name (file name without extension).

`\fileext` File extension (without leading dot).

`\filename` File name.

`\filedate` File date (in the format YYYY/MM/DD).

`\filerev` File revision, like `\rev`.

`\GetFileInfoSVN*` The star version of this macro provides the file information of the last file which called one of the `\Provides...SVN` macros. It is meant to be used inside a `.dtx` file directly after the provide macro so that the file information can be typeset inside the documentation.

A 'normal', non-star version is not yet implemented.

The provided information macros are `\filebase`, `\fileext`, `\filename`, `\filedate`, `\filerev` and `\fileinfo`. The last one contains the file description, e.g. the content of the optional argument without date and version. The other macros were already described earlier.

`\DefineFileInfoSVN`[⟨*name*⟩]
New in v1. 2009/05/03

Defines a set of macros which provide the information collected by a previous `\Provides...` macro. The macros have the form \⟨*name*⟩`@`⟨*data*⟩ where ⟨*name*⟩ is by default the filename either with the file extension (general files) or without

(packages and classes). This default can be overwritten by the optional argument. The ⟨*data*⟩ stands for `version`, `rev` (revision), `date` and `info` (the information part without the version number).
Example: Applied to the `.dtx` file of this very package the following macros are defined:

Macro	Definition
`\svn-prov.dtx@version`	v2.1747
`\svn-prov.dtx@rev`	1747
`\svn-prov.dtx@date`	2010/03/25
`\svn-prov.dtx@info`	DTX for svn-prov.sty

The style file however would get macros like `\svn-prov@version`. Because '`-`' is not a letter the macros can only be accessed using `\csname`. Therefore the optional argument `[svnprov]` is used to name the macros `\svnprov@version` etc..

3 Examples

The following examples illustrate the usage of the provided macros and how they call the equivalent standard macros internally. The example *results* are produced by expanding the corresponding example *code* while the standard provide macros are locally redefined to typeset their own name and arguments in verbatim style. This does not only simplifies the generation of this document but makes this examples also test cases which allow the package author to test the result of the defined macros.

While mostly the package macro is used here the usage is identical to the class and file macros. Of course before this macros are used it must be made sure that the `svn-prov` package is loaded which is done by using the following code direct before them:

```
\RequirePackage{svn-prov}
```

Minimal usage

The following code:

```
\ProvidesPackageSVN
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
```

is equivalent to:

```
\ProvidesPackage{svn-prov}[2010/03/25 (SVN Rev: 1747)]
```

The following code:

```
\ProvidesClassSVN
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
```

is equivalent to:

```
\ProvidesClass{svn-prov}[2010/03/25 (SVN Rev: 1747)]
```

The following code:

```
\ProvidesFileSVN
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
```

is equivalent to:

```
\ProvidesFile{svn-prov.dtx}[2010/03/25 (SVN Rev: 1747)]
```

Normal Usage

The following code:

```
\ProvidesPackageSVN
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
  [v1.0 Example Description]
```

is equivalent to:

```
\ProvidesPackage{svn-prov}[2010/03/25 v1.0 Example Description]
```

The following code:

```
\ProvidesClassSVN
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
  [v1.0 Example Description]
```

is equivalent to:

```
\ProvidesClass{svn-prov}[2010/03/25 v1.0 Example Description]
```

The following code:

```
\ProvidesFileSVN
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
  [v1.0 Example Description]
```

is equivalent to:

```
\ProvidesFile{svn-prov.dtx}[2010/03/25 v1.0 Example Description]
```

Overwriting Name

The following code:

```
\ProvidesPackageSVN[othername]
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
  [v1.0 Example Description]
```

is equivalent to:

```
\ProvidesPackage{othername}[2010/03/25 v1.0 Example Description]
```

Overwriting Name including unneeded Extension

The following code:

```
\ProvidesPackageSVN[othername.sty]
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
  [v1.0 Example Description]
```

is equivalent to:

```
\ProvidesPackage{othername}[2010/03/25 v1.0 Example Description]
```

Overwriting Name using Macros

The following code:

```
\ProvidesFileSVN[\filebase.cfg]
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
  [v1.0 Example Description]
```

is equivalent to:

```
\ProvidesFile{svn-prov.cfg}[2010/03/25 v1.0 Example Description]
```

Using Macros in File Information String

The following code:

```
\ProvidesPackageSVN
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
  [v1.\Rev Example Description]
```

is equivalent to:

```
\ProvidesPackage{svn-prov}[2010/03/25 v1.1747 Example Description]
```

Adding Text to Default Information

The following code:

```
\ProvidesPackageSVN
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
  [v1.\Rev Extra Text \revinfo]
```

is equivalent to:

```
\ProvidesPackage{svn-prov}[2010/03/25 v1.1747 Extra Text (SVN Rev: 1747)]
```

Getting the File Information

The following code:

```
\ProvidesPackageSVN
  {$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}
  [v1.\Rev Extra Text \revinfo]
\GetFileInfoSVN*
% ...
\begin{tabular}{l@{\ :\ \ }l}
  File Name      & \filename    \\
  File Base Name & \filebase    \\
  File Extension & \fileext     \\
  File Date      & \filedate    \\
  File Revision  & \filerev     \\
  File Version   & \fileversion \\
  File Info      & \fileinfo    \\
\end{tabular}
```

results in:

File Name	:	svn-prov.dtx
File Base Name	:	svn-prov
File Extension	:	dtx
File Date	:	2010/03/25
File Revision	:	1747
File Version	:	v1.1747
File Info	:	Extra Text (SVN Rev: 1747)

The correct package file extension '`.sty`' for `\fileext` can be forced by using `[\filebase.sty]` as a first optional argument.

4 Implementation

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
```

`\ProvidesClassSVN` Calls the generic macro with the original LaTeX macro and the string to be used as filename.

```
\def\ProvidesClassSVN{%
  \svnprov@generic\ProvidesClass{\svnprov@filebase}%
}
```

`\ProvidesFileSVN` Calls the generic macro with the original LaTeX macro and the string to be used as filename.

```
\def\ProvidesFileSVN{%
  \svnprov@generic\ProvidesFile{\svnprov@filebase.\svnprov@fileext}%
}
```

`\ProvidesPackageSVN` Calls the generic macro with the original LaTeX macro and the string to be used as filename.

```
\def\ProvidesPackageSVN{%
  \svnprov@generic\ProvidesPackage{\svnprov@filebase}%
}
```

`\svnprov@generic` Stores the arguments (1: original macro, 2: file mask (full filename of only base is used?)). Then tests if a explicit file name was given as optional argument. If not the file name from the SVN Id string is used.

```
\def\svnprov@generic#1#2{%
  \def\svnprov@ltxprov{#1}%
  \def\svnprov@filemask{#2}%
  \begingroup
  \svnprov@catcodes
  \@ifnextchar[{%
    {\svnprov@getid}%
    {\svnprov@getid[\svnprov@svnfilename]}%
}
```

`\svnprov@catcodes` Sets the normal catcodes for all characters required by the `getid` macro.

```
\def\svnprov@catcodes{%
  \catcode`\ =10%
  \catcode`\$=3%
  \@makeother\:%
  \@makeother\-%
}
```

Enforce normal catcodes for the definition of the `Id` scanning macros. This makes sure that all scan patterns have the same catcodes during definition and execution.

```
\begingroup
\svnprov@catcodes
```

`\svnprov@getid` Saves first argument as filename and calls the scan macro with the second. A fall-back string is provided to avoid TEX parsing errors.

```
\gdef\svnprov@getid[#1]#2{%
  \endgroup
  \def\svnprov@filename{#1}%
  \svnprov@scanid #2\relax $%
    Id: unknown.xxx 0 0000-00-00 00:00:00Z user $\empty\svnprov@endmarker
}
```

`\svnprov@scanid` Parses the Id string and tests if it is correct (#1=empty, #8=`\relax`). If correct the values are stored in macros and the next macro is called. Otherwise a warning message is printed. In both cases any remaining text of the parsing procedure is gobbled before the next step.

```
\gdef\svnprov@scanid#1$%
  Id: #2 #3 #4-#5-#6 #7 $#8{%
  \def\next{%
    \begingroup
    \def\@tempa####1>####2 {####2}%
    \PackageWarning{svn-prov}{Invalid SVN Id line found! File name might be
    '#2' or '\expandafter\@tempa\meaning\@filef@und'. This occured}{}{}{}%
    \endgroup
    \svnprov@gobbleopt
  }%
  \ifx\relax#1\relax
    \ifx\relax#8\empty
      \def\svnprov@svnfilename{#2}%
      \svnprov@splitfilename{#2}%
      \def\svnprov@filerev{#3}%
      \def\svnprov@filedate{#4/#5/#6}%
      \def\next{\begingroup\svnprov@catcodes\svnprov@buildstring}%
    \fi
  \fi
  \expandafter\next\svnprov@gobblerest
}% $
```

End of area with enforced catcodes.

```
\endgroup
```

`\svnprov@splitfilename` Expands the argument and initialises the file base macro before it calls the next macro with the expanded argument and a dot to protect for TEX parsing errors. The `\relax` is used as end marker.

```
\def\svnprov@splitfilename#1{%
  \edef\g@tempa{#1}%
  \let\svnprov@filebase\@gobble
  \expandafter
  \svnprov@splitfilename@\g@tempa.\relax
}
```

`\svnprov@splitfilename@` The second argument is tested if it is empty (end of file name reached). If not empty the first argument is concatenated to the file base macro and the macro calls itself on the second argument. This ensures correct handling of file name which contain multiple dots.

If the second argument was empty it is tested if the file base name is still in its initialised state which means that there is no file extension. Then the file base is defined to the first argument and the extension as empty. Otherwise the file extension is defined to the first argument and the file base macro is unchanged because it is already correct.

```
\def\svnprov@splitfilename@#1.#2\relax{%
  \if&#2&
    \ifx\svnprov@filebase\@gobble
      \gdef\svnprov@filebase{#1}%
      \gdef\svnprov@fileext{}%
    \else
      \gdef\svnprov@fileext{#1}%
    \fi
    \let\next\relax
  \else
    \xdef\svnprov@filebase{\svnprov@filebase.#1}%
    \def\next{\svnprov@splitfilename@#2\relax}%
  \fi
  \next
}
```

`\svnprov@gobblerest` Simply gobbles everything up to the next endmarker.

```
\def\svnprov@gobblerest#1\svnprov@endmarker{}
```

`\svnprov@endmarker` This is the end marker which should never be expanded. However it gets defined and set to an unique definition which will gobble itself if ever expanded.

```
\def\svnprov@endmarker{\@gobble{svn-prov endmarker}}
```

`\svnprov@gobbleopt` Gobbles an optional argument if present.

```
\newcommand*\svnprov@gobbleopt[1][]{}
```

`\svnprov@defaultdesc` Default description text to be used. Does not include the file date which is prepended later.

```
\def\svnprov@defaultdesc{%
  (SVN Rev:\space\svnprov@filerev)%
}
```

`\svnprov@buildstring` First aliases the internal macro to user-friendly names and then builds the info string. Finally the stored original LaTeX macro is called with the filename and information.

```
\newcommand*\svnprov@buildstring[1][\svnprov@defaultdesc]{%
  \endgroup
  \begingroup
    \let\rev\svnprov@filerev
```

```
    \let\filerev\svnprov@filerev
    \def\Rev{\rev\space}%
    \let\revinfo\svnprov@defaultdesc
    \let\filebase\svnprov@filebase
    \let\fileext\svnprov@fileext
    \ifx\fileversion\@undefined
      \def\fileversion{v0.0}%
    \fi
    \edef\filename{\filebase.\fileext}%
    \xdef\svnprov@filename{\svnprov@filename}%
    \ifx\svnprov@filename\filename\else
      \svnprov@splitfilename{\svnprov@filename}%
    \fi
    \let\filename\svnprov@filename
    \xdef\svnprov@fileinfo{#1}%
  \endgroup
  \svnprov@ltxprov{\svnprov@filemask}[\svnprov@filedate\space\svnprov@fileinfo]%
}
```

`\GetFileInfoSVN` At the moment this macro **must** be called with a star '`*`' which indicated that the current file is to be used. Other arguments are not implemented yet.

The macro provides the file information of "the current file", i.e. the last file which called one of the above `\Provides...` macros. For this the internal macros are simply copied to user-friendly names.

This macro is inspired by the macro `\GetFileInfo{`⟨*file name*⟩`}` from the `doc` package.

```
\def\GetFileInfoSVN#1{%
  \ifx*#1\relax
    \let\filebase\svnprov@filebase
    \let\fileext\svnprov@fileext
    \let\filename\svnprov@filename
    \let\filedate\svnprov@filedate
    \let\filerev\svnprov@filerev
    \expandafter\svnprov@getversion
    \svnprov@fileinfo\relax{} \relax\svnprov@endmarker
    \let\fileversion\svnprov@fileversion
    \let\fileinfo\svnprov@fileinfoonly
  \else
    \PackageError{svn-prov}{Macro \textbackslash GetFileInfoSVN without '*' is
    not implemented yet.}{}{}{}%
  \fi
}
```

`\DefineFileInfoSVN` Defines macros in the form `\`⟨*filename*⟩`@`⟨*xxx*⟩, where ⟨*xxx*⟩ is date, version, rev(ision) and info.

```
\newcommand*\DefineFileInfoSVN[1][\svnprov@filemask]{%
  \expandafter\svnprov@getversion
  \svnprov@fileinfo\relax{} \relax\svnprov@endmarker
  \expandafter
```

```
  \let\csname#1@date\endcsname\svnprov@filedate
  \expandafter
  \let\csname#1@version\endcsname\svnprov@fileversion
  \expandafter
  \let\csname#1@rev\endcsname\svnprov@filerev
  \expandafter
  \let\csname#1@info\endcsname\svnprov@fileinfoonly
}
```

`\svnprov@getversion` Splits the string at the first space into arguments #1 (version) and #2 (info). Argument #3 will be empty if there was no space in the string.

```
\def\svnprov@getversion#1 #2\relax#3\svnprov@endmarker{%
  \if&#3&%
    \def\svnprov@fileversion{??}%
  \else
    \def\svnprov@fileversion{#1}%
    \def\svnprov@fileinfoonly{#2}%
  \fi
}
```

Finally, call the macro for this package itself.

```
\ProvidesPackageSVN{$Id: svn-prov.dtx 1747 2010-03-25 20:25:48Z martin $}%
  [\svnprov@version\space Package Date/Version from SVN Keywords]
```

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.