

] [4] **View** 0.1: from [to 4 ()
] **View** 0.2 from 1 to 2 (4)
 [3] Axiom 2 is proven by ??

`structview.sty`: Structures and Views in \S T E X^*

Michael Kohlhasse
Jacobs University, Bremen
<http://kwarc.info/kohlhasse>

November 19, 2015

Abstract

The `structview` package is part of the \S T E X collection, a version of $\text{T E X}/\text{\La T E X}$ that allows to markup $\text{T E X}/\text{\La T E X}$ documents semantically without leaving the document format, essentially turning $\text{T E X}/\text{\La T E X}$ into a document format for mathematical knowledge management (MKM).

This package supplies infrastructure for OMDOC structures and views: complex semantic relations between modules/theories.

*Version v1.1 (last revised 2012/01/28)

Contents

1	Introduction	4
2	The User Interface	4
2.1	Package Options	4
2.2	Structures	4
2.3	Views	5
3	Limitations & Extensions	5
4	The Implementation	5
4.1	Structures	6
4.2	Views	7

1 Introduction

EdN:1

1

2 The User Interface

The main contributions of the `modules` package are the `module` environment, which allows for lexical scoping of semantic macros with inheritance and the `\symdef` macro for declaration of semantic macros that underly the `module` scoping.

2.1 Package Options

EdN:2

<code>showmods</code>	The <code>modules</code> package takes two options: If we set <code>showmods</code> ² , then the views (see Section 2.3) are shown. If we set the <code>qualifiedimports</code> option, then qualified imports are enabled. Qualified imports give more flexibility in module inheritance, but consume more internal memory. As qualified imports are not fully implemented at the moment, they are turned off by default see Limitation ??.
<code>qualifiedimports</code>	
<code>noauxreq</code>	The option <code>noauxreq</code> prohibits the registration of <code>\@requiremodules</code> commands in the <code>aux</code> file. They are necessary for preloading the module signatures so that entries in the table of contents can have semantic macros; but as they sometimes cause trouble the option allows to turn off preloading.
<code>showmeta</code>	If the <code>showmeta</code> is set, then the metadata keys are shown (see [Kohlhase:metakeys:ctan] for details and customization options).

2.2 Structures

EdN:3

<code>importmodulevia</code>	The <code>\importmodule</code> macro has a variant <code>\importmodulevia</code> that allows the specification of a theory morphism to be applied. <code>\importmodulevia{\langle thyid \rangle}{\langle assignments \rangle}</code> specifies the “source theory” via its identifier <code>\langle thyid \rangle</code> and the morphism by <code>\langle assignments \rangle</code> . There are four kinds: ³
<code>\vassign</code>	symbol assignments via <code>\vassign{\langle sym \rangle}{\langle exp \rangle}</code> , which defines the symbol <code>\langle sym \rangle</code> introduced in the current theory by an expression <code>\langle exp \rangle</code> in the source theory.
<code>\fassign</code>	function assignments via <code>\fassign{\langle bvars \rangle}{\langle pat \rangle}{\langle exp \rangle}</code> , is a variant which defines a function symbol <code>\langle sym \rangle</code> introduced in the current theory by mapping a pattern expression <code>\langle pat \rangle</code> (<code>\langle sym \rangle</code> applied to <code>\langle bvars \rangle</code>) to an expression <code>\langle exp \rangle</code> in the source theory on bound variables <code>\langle bvars \rangle</code> .
<code>\tassign</code>	term assignments via <code>\tassign[\langle source-cd \rangle]{\langle tname \rangle}{\langle source-tname \rangle}</code> , which assigns to the term with name <code>\langle tname \rangle</code> in the current theory a term with name <code>\langle source-tname \rangle</code> in the theory <code>\langle source-cd \rangle</code> whose default value is the source theory.

¹EdNOTE: What are structures and views?

²EdNOTE: This mechanism does not work yet, since we cannot disable it when importing modules and that leads to unwanted boxes. What we need to do instead is to tweak the `sms` utility to use an internal version that never shows anything during `sms` reading.

³EdNOTE: MK: this needs to be consolidated and researched better.

`\ttassign` **term text assignments** via `\tassign{⟨tname⟩}{⟨text⟩}`, which defines a term with name `⟨tname⟩` in the current theory via a definitional text.

```
\begin{module}[id=ring]
\begin{importmodulevia}{monoid}
  \vassign{rbase}\magbase
  \fassign{a,b}{\rtimes{A}B}{\magmaop{a}b}
  \vassign{rone}\monunit
\end{importmodulevia}
\symdef{rbase}{G}
\symdef[name=rtimes]{rtimesOp}{\cdot}
\symdef{rtimes}[2]{\infix\rtimesOp{#1}{#2}}
\symdef{rone}{1}
\begin{importmodulevia}{cgroup}
  \vassign{rplus}\magmaop
  \vassign{rzero}\monunit
  \vassign{rinvOp}\cginvOp
\end{importmodulevia}
\symdef[name=rplus]{rplusOp}{+}
\symdef{rplus}[2]{\infix\rplusOp{#1}{#2}}
\symdef[name=rminus]{rminusOp}{-}
\symdef{rminus}[1]{\infix\rminusOp{#1}{#2}}
...
\end{module}
```

Example 0: A Module for Rings with inheritance from monoids and commutative groups

2.3 Views

A view is a mapping between modules, such that all model assumptions (axioms) of the source module are satisfied in the target module. ⁴

3 Limitations & Extensions

In this section we will discuss limitations and possible extensions of the `modules` package. Any contributions and extension ideas are welcome; please discuss ideas, requests, fixes, etc on the [sTeX TRAC \[sTeX:online\]](#).

4 The Implementation

The `modules` package generates two files: the \LaTeX package (all the code between `⟨*package⟩` and `⟨/package⟩`) and the \LaTeX ML bindings (between `⟨*ltxml⟩` and

⁴EdNOTE: Document and make Examples

`</ltxml>`). We keep the corresponding code fragments together, since the documentation applies to both of them and to prevent them from getting out of sync.

First the general setup for L^AT_EX_ML

```

1 <*ltxml>
2 # -*- CPERL -*-
3 package LaTeXML::Package::Pool;
4 use strict;
5 use LaTeXML::Package;
6 </ltxml>
7 % \begin{macrocode}
8 %
9 % \subsection{Package Options}\label{sec:impl:options}
10 %
11 % We declare some switches which will modify the behavior according to the package
12 % options. Generally, an option |xxx| will just set the appropriate switches to true
13 % (otherwise they stay false). The options we are not using, we pass on to the |sref|
14 % package we require next.
15 % \begin{macrocode}
16 <*package>
17 \newif\if@structview@mh@\@structview@mh@false
18 \DeclareOption{mh}{\@structview@mh@true}
19 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{modules}}
20 \ProcessOptions
21 </package>
22 <*ltxml>
23 \DeclareOption{mh}{sub {AssignValue ('@structview' => 1);
24 \PassOptions('modules','sty',ToString(Digest(T_CS('\CurrentOption'))));}};
25 \DeclareOption(undef,sub {PassOptions('modules','sty',ToString(Digest(T_CS('\CurrentOption'))));}};
26 \ProcessOptions();
27 </ltxml>

```

The next measure is to ensure that the `sref` and `xcomment` packages are loaded (in the right version). For L^AT_EX_ML, we also initialize the package inclusions.

```

28 <*package>
29 \if@structview@mh@\RequirePackage{structview-mh}\fi
30 \RequirePackage{modules}
31 </package>
32 <*ltxml>
33 if(LookupValue('@structview')) {RequirePackage('structview-mh');}
34 RequirePackage('modules');
35 </ltxml>

```

4.1 Structures

`\importmodulevia` The `importmodulevia` environment just calls `\importmodule`, but to get around the group, we first define a local macro `\@@doit`, which does that and can be called with an `\aftergroup` to escape the environment grouping introduced by `importmodulevia`.

```

36 <*package>
37 \newenvironment{importmodulevia}[2][]{%
38   \gdef\@doit{\importmodule[#1]{#2}}%
39   \ifmod@show\par\noindent importing module #2 via \@doit\fi%
40 }{%
41   \aftergroup\@doit\ifmod@show end import\fi%
42 }%

\*assign

43 \newrobustcmd\vassign[3][]{\ifmod@show\ensuremath{#2\mapsto #3}, \fi}%
44 \newrobustcmd\tassign[3][]{\ifmod@show #2\ensuremath{\mapsto} #3, \fi}%
45 \newrobustcmd\fassign[4][]{\ifmod@show \ensuremath{#3\mapsto #4}, \fi}%
46 \newrobustcmd\ttassign[3][]{\ifmod@show #2\ensuremath{\mapsto} ‘‘#3’’, \fi}%
47 </package>

```

4.2 Views

We first prepare the ground by defining the keys for the `view` environment.

```

48 <*package>
49 \srefaddidkey{view}
50 \addmetakey*{view}{title}
51 \addmetakey{view}{display}
52 \addmetakey{view}{from}
53 \addmetakey{view}{to}
54 \addmetakey{view}{creators}
55 \addmetakey{view}{contributors}
56 \addmetakey{view}{srccite}
57 \addmetakey{view}{type}
58 \addmetakey[sms]{view}{ext}
59 </package>
60 <*ltxml>
61 DefKeyVal('view','id','Semiverbatim');
62 DefKeyVal('view','from','Semiverbatim');
63 DefKeyVal('view','to','Semiverbatim');
64 DefKeyVal('view','title','Semiverbatim');
65 DefKeyVal('view','creators','Semiverbatim');
66 DefKeyVal('view','contributors','Semiverbatim');
67 DefKeyVal('view','display','Semiverbatim');
68 DefKeyVal('view','ext','Semiverbatim');
69 </ltxml>

```

`\view@heading` Then we make a convenience macro for the view heading. This can be customized.

```

70 <*package>
71 \newcounter{view}[section]
72 \newrobustcmd{view@heading}[4]{%
73   \if@importing%
74   \else%
75     \stepcounter{view}%
76     \edef\@display{#3}\edef\@title{#4}%

```

```

77 \noindent%
78 \ifx\@display\st@flow%
79 \else%
80 {\textbf{View} {\thesection.\theview} from \textsf{#1} to \textsf{#2}}%
81 \sref@label{id{View \thesection.\theview}}%
82 \ifx\@title\@empty%
83 \quad%
84 \else%
85 \quad(\@title)%
86 \fi%
87 \par\noindent%
88 \fi%
89 \ignorespaces%
90 \fi%
91 }%ifmod@show

```

view The `view` environment relies on the `@view` environment (used also in the \TeX module signatures) for module bookkeeping and adds presentation (a heading and a box) if the `showmods` option is set.

```

92 \newenvironment{view}[3][{}]{%
93 \metasetkeys{view}{#1}%
94 \sref@target%
95 \begin{@view}{#2}{#3}%
96 \view@heading{#2}{#3}{\view@display}{\view@title}%
97 }{%
98 \end{@view}%
99 \ignorespaces%
100 }%
101 \ifmod@show\surroundwithmdframed{view}\fi%
102 \</package>
103 \<*txml>
104 DefMacroI(T_CS('\begin{view}'), 'OptionalKeyVals:view {}{}', sub {
105 my ($gullet, $keyvals, $from_arg, $to_arg) = @_;
106 my $from = ToString(Digest($from_arg));
107 my $to = ToString(Digest($to_arg));
108 my $from_file = ToString(GetKeyVal($keyvals, 'from'));
109 my $to_file = ToString(GetKeyVal($keyvals, 'to'));
110 my $ext = ToString(GetKeyVal($keyvals, 'ext')) if $keyvals;
111 $ext = 'sms' unless $ext;
112 return (
113 Tokenize("\importmoduleI[load=$from_file]{$from}")->unlist,
114 Tokenize("\importmoduleI[load=$to_file]{$to}")->unlist,
115 Invocation(T_CS('\begin{viewenv}'), $keyvals, $from_arg, $to_arg)->unlist
116 );
117 });
118 DefMacroI('\end{view}', undef, '\end{viewenv}');
119 DefEnvironment('{viewenv} OptionalKeyVals:view {}{}',
120 "<omdoc:theory-inclusion from='#{2}' to='#{3}'"
121 . " ?&defined(&GetKeyVal(#1, 'id'))(xml:id='&GetKeyVal(#1, 'id'))()>"
122 . "<omdoc:morphism>#body</omdoc:morphism>"

```



```

123 . "</omdoc:theory-inclusion>");
124 </ltxml>

```

@view The @view does the actual bookkeeping at the module level.

```

125 <*package>
126 \newenvironment{@view}[2]{%from, to
127   \@importmodule[\view@from]{#1}{\view@ext}%
128   \@importmodule[\view@to]{#2}{\view@ext}%
129 }{}%

```

viewsketch The viewsketch environment behaves like view, but only has text contents.

```

130 \newenvironment{viewsketch}[3] []{%
131   \metasetkeys{view}{#1}%
132   \sref@target%
133   \begin{@view}{#2}{#3}%
134   \view@heading{#2}{#3}{\view@display}{\view@title}%
135 }{%
136   \end{@view}%
137 }%
138 \ifmod@show\surroundwithmdframed{viewsketch}\fi%
139 </package>
140 <*ltxml>
141 # do the same for viewsketch, pity we cannot share some code.
142 DefMacroI(T_CS('\begin{viewsketch}'), 'OptionalKeyVals:view {}{}', sub {
143   my ($gullet, $keyvals, $from_arg, $to_arg) = @_;
144   my $from = ToString(Digest($from_arg));
145   my $to = ToString(Digest($to_arg));
146   my $from_file = ToString(GetKeyVal($keyvals, 'from'));
147   my $to_file = ToString(GetKeyVal($keyvals, 'to'));
148   my $ext = ToString(GetKeyVal($keyvals, 'ext')) if $keyvals;
149   $ext = 'sms' unless $ext;
150   return (
151     Tokenize("\importmoduleI[load=$from_file]{$from}")->unlist,
152     Tokenize("\importmoduleI[load=$to_file]{$to}")->unlist,
153     Invocation(T_CS('\begin{viewsketchenv}'), $keyvals, $from_arg, $to_arg)->unlist
154   );
155 });
156 DefMacroI('\end{viewsketch}', undef, '\end{viewsketchenv}');
157 DefEnvironment('{viewsketchenv} OptionalKeyVals:view {}{}',
158   "<omdoc:theory-inclusion from='#2' to='#3'"
159   . "    "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()>"
160   . "  #body"
161   . "</omdoc:theory-inclusion>");
162 </ltxml>

```

EdN:5 **\obligation** The \obligation element does not do anything yet on the latexml side.⁵

```

163 <*package>
164 \newrobustcmd\obligation[3] []{%

```

⁵EdNOTE: document above

```

165 \if@importing%
166 \else Axiom #2 is proven by \sref{#3}%
167 \fi%
168 }%
169 \end{package}
170 \end{*ltxml}
171 DefConstructor('\obligation [] {} {}',"<omdoc:obligation induced-by='#2' assertion='#3'/>");
172 \end{ltxml}

```

Finally, we need to terminate the file with a success mark for perl.

```

173 \end{ltxml}1;

```