CNXLaTeX: A LaTeX-based Syntax for Connexions Modules*

Michael Kohlhase
Jacobs University, Bremen
http://kwarc.info/kohlhase

November 10, 2015

Abstract

We present CNXLaTeX, a collection of LaTeX macros that allow to write CONNEXIONS modules without leaving the LaTeX workflow. Modules are authored in CNXLaTeX using only a text editor, transformed to PDF and proofread as usual. In particular, the LaTeX workflow is independent of having access to the CONNEXIONS system, which makes CNXLaTeX attractive for the initial version of single-author modules.

For publication, CNXLaTeX modules are transformed to CNXml via the LaTeXML translator and can be uploaded to the CONNEXIONS system.

---

*Version ? (last revised ?)

Contents

1 Introduction

The Connexions project is a[1]

The CNXML format — in particular the embedded content MATHML — is hard to write by hand, so we provide a set of environments that allow to embed the CNXML document model into LaTeX.

2 The User Interface

This document is not a manual for the Connexions XML encoding, or a practical guide how to write Connexions modules. We only document the LaTeX bindings

for CNXML and will presuppose experience with the format or familiarity with[2]. Note that formatting CNXLaTeX documents with the LaTeX formatter does little to enforce the restrictions imposed by the CNXML document model. You will need to run the LaTeXML converter for that (it includes DTD validation) and any

CNX-specific quality assurance tools after that. [3]

The CNXLaTeX class makes heavy use of the `KeyVal` package, which is part of your LaTeX distribution. This allows to add optional information to LaTeX macros in the form of key-value pairs: A macro `\foo` that takes a KeyVal argument and a regular one, so a call might look like `\foo{bar}` (no KeyVal information given) or `\foo[key1=val1,...,keyn=valn]{bar}`, where `key1,...,keyn` are predefined keywords and values are LaTeX token sequences that do not contain comma characters (though they may contain blank characters). If a value needs to contain commas, then it must be enclosed in curly braces, as in `\foo[args={a,comma,separated,list}]`. Note that the order the key/value pairs appear in a KeyVal Argument is immaterial.

2.1 Package Options

showmeta   The `cnx` package takes a single option: `showmeta`. If this is set, then the metadata keys are shown (see [Koh15a] for details and customization options).

2.2 Document Structure

The first set of CNXLaTeX environments concern the top-level structure of the modules. The minimal Connexions document in LaTeX can be seen in Figure 1:

cnxmodule   we still need the LaTeX document environment, then the `cnxmodule` environment contains the module-specific information as a KeyVal argument with the two keys: `id` for the module identifier supplied by the CONNEXIONS system) and `name` for the title of the module.

ccontent   The `content` environment delineates the module content from the metadata (see Section 2.6). It is needed to make the conversion to CNXML simpler.

c*section   CNXML knows three levels of sectioning, so the CNXLaTeX class supplies three

---

[1]EDNOTE: continue; copy from somewhere...

[2]EDNOTE: cite the relevant stuff here

[3]EDNOTE: talk about Content MATHML and cmathml.sty somewhere

```
\documentclass{cnx}
\begin{document}
  \begin{cnxmodule}[name=Hello World,id=m4711]
    \begin{ccontent}
      \begin{cpara}[id=p01] Hello World\end{cpara}
    \end{ccontent}
  \end{cnxmodule}
\end{document}
```

Example 1: A Minimal CNXLᴬTᴇX Document

as well: `csection`, `csubsection` and `csubsubsection`. In contrast to regular LᴬTᴇX, these are environments to keep the tight connection between the formats. These environments take an optional KeyVal argument with key `id` for the identifier and a regular argument for the title of the section (to be transformed into the CNXᴍʟ `name` element).

cpara, cnote   The lowest levels of the document structure are given by paragraphs and notes. The `cpara` and `cnote` environment take a KeyVal argument with the `id` key for identification, the latter also allows a `type` key for the note type (an unspecified string[4]).

2.3 Mathematics

Mathematical formulae are integrated into text via the LᴬTᴇX math mode, i.e. wrapped in `$` characters or between `\(` and `\)` for inline mathematics and wrapped in `$$` or between `\[` and `\]` for display-style math. Note that CNXᴍʟ expects Content Mᴀᴛʜᴍʟ as the representation format for mathematical formulae, while run-of-the-mill LᴬTᴇX only specifies the presentation (i.e. the two-dimensional layout of formulae). The LᴬTᴇXML converter can usually figure out some of the content Mᴀᴛʜᴍʟ from regular LᴬTᴇX, in other cases, the author has to specify it e.g. using the infrastructure supplied by the `cmathml` package.

cequation   For numbered equations, CNXᴍʟ supplies the `equation` element, for which CNXLᴬTᴇX provides the `cequation` environment. This environment takes a KeyVal argument with the `id` key for the (required) identifier.

2.4 Statements

CNXᴍʟ provides special elements that make various types of claims; we collectively call them statements.

cexample   The `cexample` environment and `definition` elements take a KeyVal argument with key `id` for identification.

crule, statement, proof   In CNXᴍʟ, the `rule` element is used to represent a general assertion about the state of the world. The CNXLᴬTᴇX `rule`[5] environment is its CNXLᴬTᴇX coun-

---

[4]EᴅNᴏᴛᴇ: what are good values?

[5]EᴅNᴏᴛᴇ: we have called this "crule", since "rule" is already used by TᴇX.

4

terpart. It takes a KeyVal attribute with the keys `id` for identification, `type` to specify the type of the assertion (e.g. "Theorem", "Lemma" or "Conjecture"), and `name`, if the assertion has a title. The body of the `crule` environment contains the statement of assertion in the `statement` environment and (optionally) a proof in the `proof` environment. Both take a KeyVal argument with an `id` key for identification.

```
\begin{crule}[id=prop1,type=Proposition]
   \begin{statement}[id=prop1s]
        Sample statement
   \end{statement}
   \begin{proof}[id=prop1p]
        Your favourite proof
   \end{proof}
\end{crule}
```

Example 2: A Basic crule Example

definition, cmeaning   A definition defines a new technical term or concept for later use. The `definition` environment takes a KeyVal argument with the keys `id` for identification and `term` for the concept (definiendum) defined in this form. The definition text is given in the `cmeaning` environment[1], which takes a KeyVal argument with key `id` for identification. After the `cmeaning` environment, a `definition` can contain arbitrarily many `cexample`s.

```
\begin{definition}{term=term-to-be-defined, id=termi-def]
  \begin{cmeaning}[id=termi-meaning]
    {\term{Term-to-be-defined}} is defined as: Sample meaning
  \end{cmeaning}
\end{definition}
```

Example 3: A Basic `definition` and `cmeaning` Example

2.5   Connexions: Links and Cross-References

As the name CONNEXIONS already suggests, links and cross-references are very important for CONNEXIONS modules. CNXML provides three kinds of them. Module links, hyperlinks, and concept references.

cnxn   Module links are specified by the `\cnxn` macro, which takes a keyval argument with the keys `document`, `target`, and `strength`. The `document` key allows to specify the module identifier of the desired module in the repository, if it is empty, then the current module is intended. The `target` key allows to specify the document fragment. Its value is the respective identifier (given by its `id` attribute in

---

[1] we have called this `cmeaning`, since `meaning` is already taken by TeX

CNXᴍʟ or the `id` key of the corresponding environment in CNXLᴬTᴇX). Finally, the `strength` key allows to specify the relevance of the link.

The regular argument of the `\cnxn` macro is used to supply the link text.

link   Hyperlinks can be specified by the `\link` macro in CNXLᴬTᴇX. It takes a KeyVal argument with the key `src` to specify the URL of the link. The regular argument of the `\link` macro is used to supply the link text.

term   The `\term` marco can be used to specify the[6]

2.6 Metadata

Metadata is mostly managed by the system in Connexions, so we often do not need to care about it. On the other hand, it influences the system, so if we have work on the module extensively before converting it to CNXᴍʟ, it may be worth-wile specify some of the data in advance.

```
\begin{metadata}[version=2.19,
                  created=2000/07/21,revised=2004/08/17 22:07:27.213 GMT-5]
\begin{authorlist}
  \cnxauthor[id=miko,firstname=Michael,surname=Kohlhase,
            email=m.kohlhase@iu-bremen.de]
\end{authorlist}
\begin{keywordlist}\keyword{Hello}\end{keywordlist}
\begin{cnxabstract}
  A Minimal CNXLaTeX Document
\end{cnxabstract}
\end{metadata}
```

Example 4: Typical CNXLᴬTᴇX Metadata

metadata   The `metadata` environment takes a KeyVal argument with the keys `version`, `created`, and `revised` with the obvious meanings. The latter keys take ISO 8601 norm representations for dates and times. Concretely, the format is `CCYY-MM-DDThh:mm:ss` where "`CC`" represents the century, "`YY`" the year, "`MM`" the month, and "`DD`" the day, preceded by an optional leading "`-`" sign to indicate a negative number. If the sign is omitted, "`+`" is assumed. The letter "`T`" is the date/time separator and "`hh`", "`mm`", "`ss`" represent hour, minutes, and seconds respectively.

authorlist, maintainerlist   The lists of authors and maintainers can be specified in the `authorlist` and `maintainerlist` environments, which take no arguments.

cnxauthor,maintainer   The entries on this lists are specified by the `\cnxauthor` and `\maintainer` macros. Which take a KeyVal argument specifying the individual. The `id` key is the identifier for the person, the `honorific`, `firstname`, `other`, `surname`, and `lineage` keys are used to specify the various name parts, and the `email` key is used to specify the e-mail address of the person.

keywordlist, keyword   The keywords are specified with a list of `keyword` macros, which take the

---

[6]EdNote: continue, pending Chuck's investigation.

respective keyword in their only argument, inside a `keyword` environment. Neither take any KeyVal arguments.

cnxabstract   The abstract of a Connexions module is considered to be part of the meta-data. It is specified using the `cnxabstract` environment. It does not take any arguments.

2.7 Exercises

cexercise, cproblem,   An exercise or problem in Connexions is specified by the `cexercise` environment,
csolution   which takes an optional keyval argument with the keys `id` and `name`. It must contain a `cproblem` environment for the problem statement and a (possibly) empty set of `csolution` environments. Both of these take an optional keyval argument with the key `id`.

2.8   Graphics, etc.

cfigure   For graphics we will use the `cfigure`[7] macro, which provides a non-floating envi-
EdN:7   ronment for including graphics into CNXml files. `cfigure` takes three arguments first an optional CNXml keys, then the keys of the `graphicx` package in a regular argument (leave that empty if you don't have any) and finally a path. So

`\cfigure[id=foo,type=image/jpeg,caption=The first FOO]{width=7cm,height=2cm}{../images/f`

EdN:8   Would include a graphic from the file at the path `../images/foo`, equip this image with a caption, and tell LaTeXML that[8] the original of the images has the MIME type `image/jpeg`.

3   Limitations

In this section we document known limitations. If you want to help alleviate them, please feel free to contact the package author. Some of them are currently discussed in the sTeX GitHub repository [sTeX].

1. none reported yet

---

[7]EdNote: probably better call it `cgraphics`
[8]EdNote: err, exactly what does it tell latexml?

4 The Implementation

The cnx package generates to files: the LaTeX package (all the code between ⟨*package⟩ and ⟨/package⟩) and the LaTeXML bindings (between ⟨*ltxml⟩ and ⟨/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.

4.1 Package Options

We declare some switches which will modify the behavior according to the package options. Generally, an option xxx will just set the appropriate switches to true (otherwise they stay false). First we have the general options

```
1 ⟨*package⟩
2 \DeclareOption{showmeta}{\PassOptionsToPackage{\CurrentOption}{metakeys}}
3 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{omdoc}}
```

Finally, we need to declare the end of the option declaration section to LaTeX.

```
4 \ProcessOptions
5 ⟨/package⟩
```

We first make sure that the sref [Koh15b] and graphicx packages are loaded.

```
6 ⟨*cls⟩
7 \RequirePackage{sref}
8 \RequirePackage{graphicx}
```

The next step is to declare (a few) class options that handle the paper size; this is useful for printing.

```
9 \DeclareOption{letterpaper}
10    {\setlength\paperheight {11in}%
11     \setlength\paperwidth  {8.5in}}
12 \DeclareOption{a4paper}
13    {\setlength\paperheight {297mm}%
14     \setlength\paperwidth  {210mm}}
15 \ExecuteOptions{letterpaper}
16 \ProcessOptions
```

Finally, we input all the usual size settings. There is no sense to use something else, and we initialize the page numbering counter and tell it to output the numbers in arabic numerals (otherwise label and reference do not work).

```
17 \input{size10.clo}
18 \pagenumbering{roman}
19 ⟨/cls⟩
```

Now comes the equivalent for LaTeXML: this is something that we will have throughout this document. Every part of the TeX/LaTeX implementation has a LaTeXML equivalent. We keep them together to ensure that they do not get out of sync.

```
20 ⟨*ltxml⟩
21 # -*- CPERL -*-
22 package LaTeXML::Package::Pool;
```

```
23 use strict;
24 use LaTeXML::Package;
25 DeclareOption('letterpaper', '');
26 DeclareOption('a4paper', '');
27 ProcessOptions();
28 RequirePackage('metakeys');
```

We set up the necessary namespaces, the first one is the default one for CNXml

```
29 RegisterNamespace('cnx'=>"http://cnx.rice.edu/cnxml");
30 RegisterNamespace('md'=>"http://cnx.rice.edu/mdml/0.4");
31 RegisterNamespace('bib'=>"http://bibtexml.sf.net/");
32 RegisterNamespace('m'=>"http://www.w3.org/1998/Math/MathML");
```

For LaTeXML we also have to set up the correct document type information. The first line gives the root element. The second gives the public identifier for the CNX DTD, then we have its URL, and finally the CNX namespace.

```
33 DocType("cnx:document",
34 "-//CNX//DTD CNXML 0.5 plus LaTeXML//EN",
35 "../dtd/cnxml+ltxml.dtd",
36 '#default'=>"http://cnx.rice.edu/cnxml",
37         'md'=>"http://cnx.rice.edu/mdml/0.4",
38         'bib'=>"http://bibtexml.sf.net/",
39         'm'=>"http://www.w3.org/1998/Math/MathML",
40         'ltx'=>"http://dlmf.nist.gov/LaTeXML");
```

And finally, we need to set up the counters for itemization, since we are defining a class file from scratch.[9]

```
41 NewCounter('@itemizei',   'document',   idprefix=>'I');
42 NewCounter('@itemizeii',  '@itemizei',  idprefix=>'I');
43 NewCounter('@itemizeiii', '@itemizeii', idprefix=>'I');
44 NewCounter('@itemizeiv',  '@itemizeiii',idprefix=>'I');
45 NewCounter('@itemizev',   '@itemizeiv', idprefix=>'I');
46 NewCounter('@itemizevi',  '@itemizev',  idprefix=>'I');
47
48 NewCounter('enumi',  '@itemizei',   idprefix=>'i');
49 NewCounter('enumii', '@itemizeii',  idprefix=>'i');
50 NewCounter('enumiii','@itemizeiii', idprefix=>'i');
51 NewCounter('enumiv', '@itemizeiv',  idprefix=>'i');
52 # A couple of more levels, since we use these for ID's!
53 NewCounter('enumv',  '@itemizev',   idprefix=>'i');
54 NewCounter('enumvi', '@itemizevi',  idprefix=>'i');
55
56 DefMacro('\theenumi',           '\arabic{enumi}');
57 DefMacro('\theenumii',          '\alph{enumii}');
58 DefMacro('\theenumiii',         '\roman{enumiii}');
59 DefMacro('\theenumiv',          '\Alph{enumiv}');
60
61 NewCounter('equation', 'document', idprefix=>'E');
62 DefMacro('\theequation', '\arabic{equation}');
```

---

[9]EDNOTE: this will have to change, when Bruce updates to the next version (0.6?)

```
63 DefMacro('\textwidth','16cm');
```

And another thing that is now needed:

```
64 Let('\thedocument@ID','\@empty');
65 ⟨/ltxml⟩
```

4.2   Document Structure

Now, we start with the document structure markup. The `cnxmodule` environment does not add anything to the LATEX output, it's attributes only show up in the XML. There we have a slight complication: we have to put an `id` attribute on the `document` element in CNXML, but we cannot redefine the `document` environment in LATEX. Therefore we specify the information in the `cnxmodule` environment. This means however that we have to put in on the `document` element when we are already past this. The solution here is that when we parse the `cnxmodule` environement, we store the value and put it on the `document` element when we leave the `document` environment (thanks for Ioan Sucan for the code).

cnxmodule

```
66 ⟨*cls⟩
67 \addmetakey{cnxmodule}{name}
68 \srefaddidkey{cnxmodule}{id}
69 \newenvironment{cnxmodule}[1][]{\metasetkeys{cnxmodule}{#1}}{}
70 ⟨/cls⟩
71 ⟨*ltxml⟩
72 DefKeyVal('cnxmodule','name','Semiverbatim');
73 DefKeyVal('cnxmodule','id','Semiverbatim');
74 DefEnvironment('{document}','<cnx:document>#body</cnx:document>',
75     beforeDigest=> sub { AssignValue(inPreamble=>0); },
76     afterDigest=> sub { $_[0]->getGullet->flush; return; });
77 DefEnvironment('{cnxmodule} OptionalKeyVals:cnxmodule',
78     "<cnx:name>&GetKeyVal('#1','name')</cnx:name>\n#body\n",
79     afterDigestBegin => sub {
80  AssignValue('cnxmodule_id',
81      KeyVal($_[1]->getArg(1), 'id')->toString,
82      'global');
83         });#$
84 Tag('cnx:document', afterClose => sub {
85       $_[1]->setAttribute('id', LookupValue('cnxmodule_id'));
86     });
87 ⟨/ltxml⟩
```

ccontent   The `ccontent` environment is only used for transformation. Its optional `id` attribute is not taken up in the LATEX bindings.

```
88 ⟨*cls⟩
89 \newenvironment{ccontent}{}{}
90 ⟨/cls⟩
91 ⟨*ltxml⟩
92 DefEnvironment('{ccontent}',"<cnx:content>#body</cnx:content>");
```

93 ⟨/ltxml⟩

c*section   The sectioning environments employ the obvious nested set of counters.

```
94 ⟨∗cls⟩
95 \newcounter{section}
96 \srefaddidkey{sectioning}{id}
97 \newenvironment{csection}[2][]%
98 {\stepcounter{section}\strut\\[1.5ex]\noindent%
99 {\Large\bfseries\arabic{section}.~{#2}}\\[1.5ex]
100 \metasetkeys{sectioning}{#1}}
101 {}
102 \newcounter{subsection}[section]
103 \newenvironment{csubsection}[2][]
104 {\refstepcounter{subsection}\strut\\[1ex]\noindent%
105 {\large\bfseries{\arabic{section}.\arabic{subsection}.~#2\\[1ex]}}%
106 \metasetkeys{sectioning}{#1}}%
107 {}
108 \newcounter{subsubsection}[subsection]
109 \newenvironment{csubsubsection}[2][]
110 {\refstepcounter{subsubsection}\strut\\[.5ex]\noindent
111 {\bfseries\arabic{section}.\arabic{subsection}.\arabic{subsubsecction}~#2\\[.5ex]}%
112 \metasetkeys{sectioning}{#1}}{}
113 ⟨/cls⟩
114 ⟨∗ltxml⟩
115 DefKeyVal('sectioning','id','Semiverbatim');
116 DefEnvironment('{csection}OptionalKeyVals:sectioning{}',
117      "<cnx:section %&GetKeyVals(#1)>\n"
118           . "?#2(<cnx:name>#2</cnx:name>\n)()"
119           . "#body\n</cnx:section>\n");
120 DefEnvironment('{csubsection}OptionalKeyVals:sectioning{}',
121      "<cnx:section %&GetKeyVals(#1)>\n"
122           . "?#2(<cnx:name>#2</cnx:name>\n)()"
123           . "#body\n</cnx:section>\n");
124 DefEnvironment('{csubsubsection}OptionalKeyVals:sectioning{}',
125      "<cnx:section %&GetKeyVals(#1)>\n"
126           . "?#2(<cnx:name>#2</cnx:name>\n)()"
127           . "#body\n</cnx:section>\n");
128 ⟨/ltxml⟩
```

cpara   For the `<cnx:para>` element we have to do some work, since we want them to be numbered. This handling is adapted from Bruce Miller's `LaTeX.ltxml` numbered.

```
129 ⟨∗cls⟩
130 \srefaddidkey{para}{id}
131 \newenvironment{cpara}[1][]{\metasetkeys{para}{#1}}{\par}
132 ⟨/cls⟩
133 ⟨∗ltxml⟩
134 DefKeyVal('para','id','Semiverbatim');
135 DefEnvironment('{cpara} OptionalKeyVals:para','<cnx:para %&GetKeyVals(#1)>#body</cnx:para>');
136 sub number_para {
137   my($document,$node,$whatsit)=@_;
```

```
138    # Get prefix from first parent with an id.
139    my(@parents)=$document->findnodes('ancestor::*[@id]',$node); # find 1st id'd parent.
140    my $prefix= (@parents ? $parents[$#parents]->getAttribute('id')."." : '');
141    # Get the previous number within parent; Worried about intervening elements around para's, bu
142    my(@siblings)=$document->findnodes("preceding-sibling::cnx:para",$node);
143    my $n=1;
144    $n = $1+1  if(@siblings && $siblings[$#siblings]->getAttribute('id')=~/(\d+)$/);
145    $node->setAttribute(id=>$prefix."p$n"); }
146 Tag('cnx:para',afterOpen=>\&number_para);
147 DefConstructor('\par',sub { $_[0]->maybeCloseElement('cnx:para'); },alias=>"\\par\n");
148 Tag('cnx:para', autoClose=>1, autoOpen=>1);
149 ⟨/ltxml⟩
```

cnote

```
150 ⟨*cls⟩
151 \srefaddidkey{note}
152 \addmetakey{note}{type}
153 \newenvironment{cnote}[1][]%
154 {\metasetkeys{note}{#1}\par\noindent\strut\hfill\begin{minipage}{10cm}{\bfseries\note@type}:~}%
155 {\end{minipage}\hfill\strut\par}
156 ⟨/cls⟩
157 ⟨*ltxml⟩
158 DefKeyVal('note','id','Semiverbatim');
159 DefKeyVal('note','type','Semiverbatim');
160 DefEnvironment('{cnote}OptionalKeyVals:note','<cnx:note %&GetKeyVals(#1)>#body</cnx:note>');
161 ⟨/ltxml⟩
```

4.3   Mathematics

cequation

```
162 ⟨*cls⟩
163 \srefaddidkey{equation}{id}
164 \newenvironment{cequation}[1][]%
165 {\metasetkeys{equation}{#1}\begin{displaymath}}
166 {\end{displaymath}}
167 ⟨/cls⟩
168 ⟨*ltxml⟩
169 DefKeyVal('equation','id','Semiverbatim');
170 DefEnvironment('{cequation} OptionalKeyVals:equation',
171      "<cnx:equation %&GetKeyVals(#1)>"
172          . "<ltx:Math mode='display'>"
173          . "<ltx:XMath>#body</ltx:XMath>"
174          . "</ltx:Math></cnx:equation>",
175      mode=>'display_math');
176 ⟨/ltxml⟩
```

4.4 Rich Text

In this section, we redefine some of LaTeX commands that have their counterparts in CNXML.

quote

```
177 ⟨∗cls⟩
178 \srefaddidkey{cquote}
179 \addmetakey{cquote}{type}
180 \addmetakey{cquote}{src}
181 \newenvironment{cquote}[1][]{%
182 \metasetkeys{cquote}{#1}\begin{center}\begin{minipage}{.8\textwidth}}{\end{minipage}\end{center
183 ⟨/cls⟩
184 ⟨∗ltxml⟩
185 DefKeyVal('cquote','id','Semiverbatim');
186 DefKeyVal('cquote','type','Semiverbatim');
187 DefKeyVal('cquote','src','Semiverbatim');
188 DefEnvironment('{cquote} OptionalKeyVals:cquote',
189                "<cnx:quote %&GetKeyVals(#1)>#body</cnx:quote>");
190 ⟨/ltxml⟩
```

footnote

```
191 ⟨∗ltxml⟩
192 DefConstructor('\footnote[]{}',"<cnx:note type='foot'>#2</cnx:note>");
193 ⟨/ltxml⟩
```

emph

```
194 ⟨∗ltxml⟩
195 DefConstructor('\emph{}',"<cnx:emphasis>#1</cnx:emphasis>");
196 ⟨/ltxml⟩
```

displaymath, eqnarray We redefine the abbreviate display math envionment and the `eqnarray` and `eqnarray*` environments to use the CNXML equation tags, everything else stays the same.

```
197 ⟨∗ltxml⟩
198 DefConstructor('\[',
199        "<cnx:equation id='#id'>"
200      . "<ltx:Math mode='display'>"
201      .  "<ltx:XMath>"
202      .   "#body"
203      .  "</ltx:XMath>"
204      . "</ltx:Math>"
205      ."</cnx:equation>",
206        beforeDigest=> sub{ $_[0]->beginMode('display_math'); },
207        captureBody=>1,
208        properties=> sub { RefStepID('equation') });
209 DefConstructor('\]'  ,"",beforeDigest=> sub{ $_[0]->endMode('display_math'); });
210 ⟨/ltxml⟩
```

displaymath We redefine the abbreviate display math envionment to use the CNXML equation tags, everything else stays the same.[10]

```
211 ⟨∗ltxml⟩
212 DefConstructor('\[',
213       "<cnx:equation id='#id'>"
214     . "<ltx:Math mode='display'>"
215     .  "<ltx:XMath>"
216     .   "#body"
217     .  "</ltx:XMath>"
218     . "</ltx:Math>"
219     ."</cnx:equation>",
220       beforeDigest=> sub{ $_[0]->beginMode('display_math'); },
221       captureBody=>1,
222       properties=> sub { RefStepID('equation') });
223 DefConstructor('\]'  ,"",beforeDigest=> sub{ $_[0]->endMode('display_math'); });
224
225 DefMacro('\eqnarray',     '\@@eqnarray\@start@alignment');
226 DefMacro('\endeqnarray', '\@finish@alignment\end@eqnarray');
227 DefMacro('\csname eqnarray*\endcsname',    '\@@eqnarray*\@start@alignment');
228 DefMacro('\csname endeqnarray*\endcsname', '\@finish@alignment\end@eqnarray');
229 DefConstructor('\@@eqnarray OptionalMatch:* AlignmentBody:\end@eqnarray',
230       sub {
231  my($document,$star,$body,%props)=@_;
232  $document->openElement('cnx:equation',refnum=>$props{refnum},id=>$props{id});
233  $document->openElement('ltx:Math',mode=>'display');
234  $document->openElement('ltx:XMath');
235  constructAlignment($document,$body,attributes=>{name=>'eqnarray'});
236  $document->closeElement('ltx:XMath');
237  $document->closeElement('ltx:Math');
238  $document->closeElement('cnx:equation'); },
239       mode=>'display_math',
240       beforeDigest=>sub { alignmentBindings('rcl'); },
241       properties=> sub { ($_[1] ? RefStepID('equation') : RefStepCounter('equation')); },
242       afterDigest=>sub {
243  $_[1]->setProperty(body=>$_[1]->getArg(2));},# So we get TeX
244       reversion=>'\begin{eqnarray#1}#2\end{eqnarray#1}');
245 ⟨/ltxml⟩
```

displaymath We redefine the abbreviate display math envionment to use the CNXML equation tags, everything else stays the same.[11]

```
246 ⟨∗cls⟩
247 \newcommand\litem[2][]{\item[#1]\label{#2}}
248 ⟨/cls⟩
```

---

[10]EDNOTE: check LaTeX.ltxml frequently and try to keep in sync, it would be good, if the code in LaTeXML.ltxml could be modularized, so that the cnx/ltx namespace differences could be relegated to config options

[11]EDNOTE: check LaTeX.ltxml frequently and try to keep in sync, it would be good, if the code in LaTeXML.ltxml could be modularized, so that the cnx/ltx namespace differences could be relegated to config options

```
249 ⟨∗ltxml⟩
250 Tag('cnx:item', autoClose=>1);
251 DefConstructor('\item[]',"<cnx:item>?#1(<cnx:name>#1</cnx:name>)");
252 DefConstructor('\litem[]{}',"<cnx:item id='#2'>?#1(<cnx:name>#1</cnx:name>)");
253 DefConstructor('\itemize@item[]',
254        "<cnx:item id='#id'>?#1(<cnx:name>#1</cnx:name>)",
255        properties=>sub{ RefStepItemCounter(); });
256 DefConstructor('\enumerate@item[]',
257        "<cnx:item id='#id'>?#1(<cnx:name>#1</cnx:name>)",
258        properties=>sub{ RefStepItemCounter(); });
259 DefConstructor('\description@item[]',
260        "<cnx::item id='#id'>?#1(<cnx:name>#1</cnx:name>)",
261        properties=>sub{ RefStepItemCounter(); });
262 AssignValue(itemlevel=>0);
263 DefEnvironment('{itemize}',
264        "<cnx:list id='#id' type='itemize'>#body</cnx:list>",
265        properties=>sub { beginItemize('itemize'); });
266 DefEnvironment('{enumerate}',
267        "<cnx:list type='enumerate'  id='#id'>#body</cnx:list>",
268        properties=>sub { beginItemize('enumerate'); });
269 DefEnvironment('{description}',
270        "<cnx:list  type='description'  id='#id'>#body</cnx:list>",
271        properties=>sub { beginItemize('description'); });
272 ⟨/ltxml⟩
```

The next set of commands and environments are largely presentational, so we just skip them.

```
273 ⟨∗ltxml⟩
274 DefEnvironment('{center}','#body');
275 DefEnvironment('{minipage}{}','#body');
276 DefEnvironment('{small}','#body');
277 DefEnvironment('{footnotesize}','#body');
278 DefEnvironment('{tiny}','#body');
279 DefEnvironment('{scriptsize}','#body');
280 ⟨/ltxml⟩
```

```
281 ⟨∗ltxml⟩
282 DefConstructor('\ref Semiverbatim', "<cnx:cnxn target='#1'>&LookupValue('LABEL@#1')</cnx:cnxn>"
283 ⟨/ltxml⟩
```

4.5   Statements

cexample
```
284 ⟨∗cls⟩
285 \srefaddidkey{example}
286 \addmetakey{example}{name}
287 \newenvironment{cexample}[1][]{\metasetkeys{example}{#1}
288 {\ifx\example@name\@empty\else\noindent\bfseries{\example@name}\fi}}
289 {}
290 ⟨/cls⟩
```

```
291 ⟨∗ltxml⟩
292 DefKeyVal('example','id','Semiverbatim');
293 DefEnvironment('{cexample}OptionalKeyVals:example',
294               "<cnx:example %&GetKeyVals(#1)>#body</cnx:example>");
295 ⟨/ltxml⟩
```

cexercise The `cexercise`, `cproblem` and `csolution` environments are very simple to set up for LaTeX. For the LaTeXML side, we simplify matters considerably for the moment by restricting the possibilities we have on the CNXML side: We assume that the content is just one `<cnx:para>` element for the `<cnx:problem>` and `<cnx:solution>` elements.[12]

EdN:12

```
296 ⟨∗cls⟩
297 \newcounter{cexercise}
298 \srefaddidkey{cexercise}
299 \addmetakey{cexercise}{name}
300 \newenvironment{cexercise}[1][]{\metasetkeys{cexercise}{#1}
301 {\ifx\cexercise@name\@empty\else\stepcounter{cexercise}\noindent\bfseries{\cexercise@name~\arab
302 {}
303 \srefaddidkey{cproblem}
304 \newenvironment{cproblem}[1][]{\metasetkeys{cproblem}{#1}}{}{}
305 \srefaddidkey{csolution}
306 \newenvironment{csolution}[1][]{\metasetkeys{csolution}{#1}}{\par\noindent\bfseries{Solution}}{
307 ⟨/cls⟩
308 ⟨∗ltxml⟩
309 DefKeyVal('cexercise','id','Semiverbatim');
310 DefKeyVal('cexercise','name','Semiverbatim');
311 DefEnvironment('{cexercise}OptionalKeyVals:exercise',
312               "<cnx:exercise ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>"
313             . "#body"
314           . "</cnx:exercise>");
315 DefKeyVal('cproblem','id','Semiverbatim');
316 DefKeyVal('cproblem','name','Semiverbatim');
317 DefEnvironment('{cproblem}OptionalKeyVals:cproblem',
318               "<cnx:problem ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>"
319         . "?&defined(&GetKeyVal(#1,'name'))(<cnx:name>&GetKeyVal(#1,'name')</cnx:name>\n)()"
320             . "#body"
321           . "</cnx:problem>");
322 DefKeyVal('csolution','id','Semiverbatim');
323 DefKeyVal('csolution','name','Semiverbatim');
324 DefEnvironment('{csolution}OptionalKeyVals:cproblem',
325               "<cnx:solution ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>"
326         . "?&defined(&GetKeyVal(#1,'name'))(<cnx:name>&GetKeyVal(#1,'name')</cnx:name>\n)()"
327             . "#body"
328           . "</cnx:solution>");
329 ⟨/ltxml⟩
```

crule

---

[12]EDNOTE: relax this when we have automated the generation of cnx:para elements

```
330 ⟨*cls⟩
331 \srefaddidkey{rule}
332 \addmetakey{rule}{name}
333 \addmetakey{rule}{type}
334 \newenvironment{crule}[1][]{\metasetkeys{rule}{#1}%
335 {\noindent\bfseries{\rule@type:}\ifx\rule@name\@empty\else~(\rule@name)\fi}}%
336 {}
337 ⟨/cls⟩
338 ⟨*ltxml⟩
339 DefKeyVal('rule','id','Semiverbatim');
340 DefKeyVal('rule','name','Semiverbatim');
341 DefKeyVal('rule','type','Semiverbatim');
342 DefEnvironment('{crule}OptionalKeyVals:rule',
343               "<cnx:rule ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')() type='&Get
344     . "?&defined(&GetKeyVal(#1,'name'))(<cnx:name>&GetKeyVal(#1,'name')</cnx:name>\n)()"
345     . "\n#body\n"
346     . "</cnx:rule>\n");
347 ⟨/ltxml⟩
```

statement

```
348 ⟨*cls⟩
349 \srefaddidkey{statement}
350 \newenvironment{statement}[1][]{\metasetkeys{statement}{#1}}{}
351 ⟨/cls⟩
352 ⟨*ltxml⟩
353 DefKeyVal('statement','id','Semiverbatim');
354 DefEnvironment('{statement} OptionalKeyVals:statement','<cnx:statement %&GetKeyVals(#1)>#body</
355 ⟨/ltxml⟩
```

proof

```
356 ⟨*cls⟩
357 \srefaddidkey{proof}
358 \newenvironment{proof}[1][]{\metasetkeys{proof}{#1}}{}
359 ⟨/cls⟩
360 ⟨*ltxml⟩
361 DefKeyVal('proof','id','Semiverbatim');
362 DefEnvironment('{proof}OptionalKeyVals:proof','<cnx:proof %&GetKeyVals(#1)>#body</cnx:proof>');
363 ⟨/ltxml⟩
```

definition

```
364 ⟨*cls⟩
365 \srefaddidkey{definition}
366 \addmetakey{definition}{term}
367 \addmetakey{definition}{seealso}
368 \newenvironment{definition}[1][]{\metasetkeys{definition}{#1}{\noindent\bfseries{Definition:}}}
369 ⟨/cls⟩
370 ⟨*ltxml⟩
371 DefKeyVal('definition','id','Semiverbatim');
372 DefKeyVal('definition','term','Semiverbatim');
```

```
373 DefKeyVal('definition','seealso','Semiverbatim');
374 DefEnvironment('{definition}OptionalKeyVals:definition',
375         "<cnx:definition ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>\n"
376       . "?&defined(&GetKeyVal(#1,'term'))(<cnx:term>&GetKeyVal(#1,'term')</cnx:term>\n)()"
377       . "\n#body\n"
378       . "?&defined(&GetKeyVal(#1,'seealso'))(<cnx:seealso><cnx:term>&GetKeyVal(#1,'term')</cnx:t
379       . "</cnx:definition>\n");
380 ⟨/ltxml⟩
```

cmeaning

```
381 ⟨*cls⟩
382 \srefaddidkey{meaning}
383 \newenvironment{cmeaning}[1][]{\metasetkeys{meaning}{#1}}{}
384 ⟨/cls⟩
385 ⟨*ltxml⟩
386 DefKeyVal('meaning','id','Semiverbatim');
387 DefEnvironment('{cmeaning}OptionalKeyVals:meaning','<cnx:meaning %&GetKeyVals(#1)>#body</cnx:me
388 ⟨/ltxml⟩
```

4.6 Conexxions

cnxn

```
389 ⟨*cls⟩
390 \addmetakey{cnxn}{document}
391 \addmetakey{cnxn}{target}
392 \addmetakey{cnxn}{strength}
393 \newcommand\cnxn[2][]% keys, link text
394 {\metasetkeys{cnxn}{#1}{\underline{#2}}\footnote{{\ttfamily\@ifx\cnxn@document\@empty\cnxn@docu
395 \newcommand\@makefntext[1]{\parindent 1em\noindent\hb@xt@1.8em{\hss\@makefnmark}#1}
396 ⟨/cls⟩
397 ⟨*ltxml⟩
398 DefKeyVal('cnxn','document','Semiverbatim');
399 DefKeyVal('cnxn','target','Semiverbatim');
400 DefKeyVal('cnxn','strength','Semiverbatim');
401 DefConstructor('\cnxn OptionalKeyVals:cnxn {}','<cnx:cnxn %&GetKeyVals(#1)>#1</cnx:cnxn>');
402 ⟨/ltxml⟩
```

link

```
403 ⟨*cls⟩
404 \addmetakey{link}{src}
405 \newcommand\link[2][]{\metasetkeys{link}{#1}\underline{#2}}
406 ⟨/cls⟩
407 ⟨*ltxml⟩
408 DefKeyVal('link','src','Semiverbatim');
409 DefConstructor('\link OptionalKeyVals:link {}','<cnx:link %&GetKeyVals(#1)>#2</cnx:link>');
410 ⟨/ltxml⟩
```

cfigure   The cfigure only gives us one of the possible instances of the <figure> ele-

ment[13].[14] In LaTeX, we just pipe the size information through to includegraphics,
in LaTeXML, we construct the CNXML structure[15]

```
411 ⟨∗cls⟩
412 \srefaddidkey{cfigure}
413 \addmetakey{cfigure}{type}
414 \addmetakey{cfigure}{caption}
415 \newcounter{figure}
416 \newcommand\cfigure[3][]{% cnx_keys, graphicx_keys, path
417 \begin{center}%
418 \includegraphics[#2]{#3}%
419 \metasetkeys{cfigure}{#1}\sref@target%
420 \ifx\cfigure@caption\@empty\else
421 \par\noindent Figure\refstepcounter{figure} {\arabic{figure}}: \cfigure@caption%
422 \protected@edef\@currentlabel{\arabic{figure}}%
423 \sref@label@id{Figure \thefigure}\fi
424 \end{center}}
425 ⟨/cls⟩
426 ⟨∗ltxml⟩
427 DefKeyVal('cfigure','id','Semiverbatim');
428 DefKeyVal('cfigure','name','Semiverbatim');
429 DefKeyVal('cfigure','type','Semiverbatim');
430 DefKeyVal('cfigure','caption','Semiverbatim');
431 DefConstructor('\cfigure OptionalKeyVals:cfigure Semiverbatim Semiverbatim',
432             "<cnx:figure ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>"
433           . "?&defined(&GetKeyVal(#1,'name'))(<cnx:name>&GetKeyVal(#1,'name')</cnx:name>\n)()"
434             . "<cnx:media type='&GetKeyVal(#1,'type')' src='#3'/>"
435           . "?&defined(&GetKeyVal(#1,'caption'))(<cnx:caption>&GetKeyVal(#1,'caption')</cnx:cap
436             . "</cnx:figure>");
437 ⟨/ltxml⟩
```

ccite

```
438 ⟨∗cls⟩
439 \addmetakey{ccite}{src}
440 \newcommand\ccite[2][]{\metasetkeys{ccite}{#1}\emph{#2}}
441 ⟨/cls⟩
442 ⟨∗ltxml⟩
443 DefKeyVal('ccite','src','Semiverbatim');
444 DefConstructor('\ccite OptionalKeyVals:ccite {}','<cnx:cite %&GetKeyVals(#1)>#2</cnx:cite>');
445 ⟨/ltxml⟩
```

term

```
446 ⟨∗cls⟩
447 \newcommand\term[1]{{\bfseries\underline{#1}}}
448 ⟨/cls⟩
449 ⟨∗ltxml⟩
450 DefConstructor('\term[]{}',"<cnx:term>#2</cnx:term>");
```

---

[13]EDNOTE: extend that

[14]EDNOTE: do more about required and optional keys in arguments.

[15]EDNOTE: what do we do with the graphicx information about size,... CSS?

451 ⟨/ltxml⟩

4.7 Metadata

452 ⟨∗cls⟩
453 \addmetakey{metadata}{version}
454 \addmetakey{metadata}{created}
455 \addmetakey{metadata}{revised}
456 \newsavebox{\metadatabox}
457 \newenvironment{metadata}[1][]%
458 {\noindent\hfill\begin{lrbox}{\metadatabox}
459 \begin{minipage}{.8\textwidth}%
460 {\Large\bfseries CNX Module: \cnx@name\hfill\strut}\\[2ex]}%
461 {\end{minipage}\end{lrbox}\fbox{\usebox\metadatabox}\hfill}
462 % \newenvironment{metadata}[1][]%
463 % {\noindent\strut\hfill\begin{lrbox}{\metadatabox}\begin{minipage}{10cm}%
464 % {\strut\hfill\Large\bfseries CNX Module: \cnx@name\hfill\strut}\\[2ex]}%
465 % {\end{minipage}\end{lrbox}\fbox{\usebox\metadatabox}\hfill\strut\\[3ex]}
466 ⟨/cls⟩
467 ⟨∗ltxml⟩
468 DefKeyVal('metadata','version','Semiverbatim');
469 DefKeyVal('metadata','created','Semiverbatim');
470 DefKeyVal('metadata','revised','Semiverbatim');
471 DefEnvironment('{metadata}OptionalKeyVals:metadata',
472   "<cnx:metadata>\n"
473   . "<md:version>&GetKeyVal('#1','version')</md:version>\n"
474   . "<md:created>&GetKeyVal('#1','created')</md:created>\n"
475   . "<md:revised>&GetKeyVal('#1','revised')</md:revised>\n"
476   . "#body\n"
477   . "</cnx:metadata>");
478 ⟨/ltxml⟩

479 ⟨∗cls⟩
480 \newenvironment{authorlist}{{\bfseries{Authors}:~}}{\\[1ex]}
481 ⟨/cls⟩
482 ⟨∗ltxml⟩
483 DefEnvironment('{authorlist}',"<md:authorlist>#body</md:authorlist>");
484 ⟨/ltxml⟩

485 ⟨∗cls⟩
486 \newenvironment{maintainerlist}{{\bfseries{Maintainers}:~}}{\\[1ex]}
487 ⟨/cls⟩
488 ⟨∗ltxml⟩
489 DefEnvironment('{maintainerlist}',"<md:maintainerlist>#body</md:maintainerlist>");
490 ⟨/ltxml⟩

cnxauthor

491 ⟨∗cls⟩
492 \srefaddidkey{auth}
493 \addmetakey{auth}{honorific}
494 \addmetakey{auth}{firstname}
495 \addmetakey{auth}{other}
496 \addmetakey{auth}{surname}
497 \addmetakey{auth}{lineage}
498 \addmetakey{auth}{email}
499 \newcommand\cnxauthor[1][]{\metasetkeys{auth}{#1}\auth@first~\auth@sur,}
500 ⟨/cls⟩
501 ⟨∗ltxml⟩
502 DefKeyVal('auth','id','Semiverbatim');
503 DefKeyVal('auth','firstname','Semiverbatim');
504 DefKeyVal('auth','surname','Semiverbatim');
505 DefKeyVal('auth','email','Semiverbatim');
506 DefConstructor('\cnxauthor OptionalKeyVals:auth',
507   "<md:author id='&GetKeyVal('#1','id')'>\n"
508   . "?&defined(&GetKeyVal(#1,'honorific'))(<md:honorific>&GetKeyVal('#1','honorific')</md:ho
509   . "?&defined(&GetKeyVal(#1,'firstname'))(<md:firstname>&GetKeyVal('#1','firstname')</md:fi
510   . "?&defined(&GetKeyVal(#1,'other'))(<md:other>&GetKeyVal('#1','other')</md:other>\n)()"
511   . "?&defined(&GetKeyVal(#1,'surname'))(<md:surname>&GetKeyVal('#1','surname')</md:surname>
512   . "?&defined(&GetKeyVal(#1,'lineage'))(<md:lineage>&GetKeyVal('#1','lineage')</md:
513   . "?&defined(&GetKeyVal(#1,'email'))(<md:email>&GetKeyVal('#1','email')</md:email>
514   . "</md:author>\n");
515 ⟨/ltxml⟩

maintainer

516 ⟨∗cls⟩
517 \newcommand\maintainer[1][]{\metasetkeys{auth}{#1}\auth@first~\auth@sur,}
518 ⟨/cls⟩
519 ⟨∗ltxml⟩
520 DefConstructor('\maintainer OptionalKeyVals:auth',
521   "<md:maintainer id='&GetKeyVal('#1','id')'>\n"
522   . "?&defined(&GetKeyVal(#1,'honorific'))(<md:honorific>&GetKeyVal('#1','honorific')</md:ho
523   . "?&defined(&GetKeyVal(#1,'firstname'))(<md:firstname>&GetKeyVal('#1','firstname')</md:fi
524   . "?&defined(&GetKeyVal(#1,'other'))(<md:other>&GetKeyVal('#1','other')</md:other>\n)()"
525   . "?&defined(&GetKeyVal(#1,'surname'))(<md:surname>&GetKeyVal('#1','surname')</md:surname>
526   . "?&defined(&GetKeyVal(#1,'lineage'))(<md:lineage>&GetKeyVal('#1','lineage')</md:
527   . "?&defined(&GetKeyVal(#1,'email'))(<md:email>&GetKeyVal('#1','email')</md:email>
528   . "</md:maintainer>\n");
529 ⟨/ltxml⟩

keywordlist

530 ⟨∗cls⟩
531 \newenvironment{keywordlist}{\bfseries{Keywords}:~}{\\[1ex]}
532 ⟨/cls⟩
533 ⟨∗ltxml⟩
534 DefEnvironment('{keywordlist}',"<md:keywordlist>\n#body\n</md:keywordlist>");
535 ⟨/ltxml⟩

21

keyword

```
536 ⟨∗cls⟩
537 \newcommand\keyword[1]{#1,}
538 ⟨/cls⟩
539 ⟨∗ltxml⟩
540 DefConstructor('\keyword {}',"<md:keyword>#1</md:keyword>");
541 ⟨/ltxml⟩
```

cnxabstract

```
542 ⟨∗cls⟩
543 \newenvironment{cnxabstract}%
544 {\par\noindent\strut\hfill\begin{minipage}{10cm}{\bfseries{Abstract}:~}}%
545 {\end{minipage}\hfill}
546 ⟨/cls⟩
547 ⟨∗ltxml⟩
548 DefEnvironment('{cnxabstract} OptionalKeyVals:cnxabstract',
549      "<md:abstract>\n#body\n</md:abstract>\n");
550 1;
551 ⟨/ltxml⟩
```

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.

23

References

[Koh15a] Michael Kohlhase. `metakeys.sty`: *A generic framework for extensible Metadata in LATEX*. Tech. rep. Comprehensive TEX Archive Network (CTAN), 2015. URL: `http://www.ctan.org/tex-archive/macros/latex/contrib/stex/metakeys/metakeys.pdf`.

[Koh15b] Michael Kohlhase. `sref.sty`: *Semantic Crossreferencing in LATEX*. Tech. rep. Comprehensive TEX Archive Network (CTAN), 2015. URL: `http://www.ctan.org/tex-archive/macros/latex/contrib/stex/sref/sref.pdf`.

[sTeX] *KWARC/sTeX*. URL: `https://svn.kwarc.info/repos/stex` (visited on 05/15/2015).