CNXLaTeX: A LaTeX-based Syntax for Connexions Modules[*]

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

February 18, 2014

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

EdN:1 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]. (EdN: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] (EdN: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 [Koh13a] 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 CNXLaTeX Document

as well: `csection`, `csubsection` and `csubsubsection`. In contrast to regular LaTeX, 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 CNXML `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]). EdN:4

2.3 Mathematics

Mathematical formulae are integrated into text via the LaTeX 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 CNXML expects Content MathML as the representation format for mathematical formulae, while run-of-the-mill LaTeX only specifies the presentation (i.e. the two-dimensional layout of formulae). The LaTeXML converter can usually figure out some of the content MathML from regular LaTeX, in other cases, the author has to specify it e.g. using the infrastructure supplied by the `cmathml` package.

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

2.4 Statements

CNXML 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 CNXML, the `rule` element is used to represent a general assertion about the state of the world. The CNXLaTeX `rule`[5] environment is its CNXLaTeX coun- EdN:5

[4]EdNote: what are good values?

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

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 `cexamples`.

```
\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

CNXML or the `id` key of the corresponding environment in CNXLATEX). 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 CNXLATEX. 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]

EdN: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 CNXML, 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 CNXLATEX 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 metadata. It is specified using the `cnxabstract` environment. It does not take any arguments.

2.7 Exercises

`cexercise, cproblem, csolution` An exercise or problem in CONNEXIONS is specified by the `cexercise` environment, 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` EdN:7 For graphics we will use the `cfigure`[7] macro, which provides a non-floating environment 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 TRAC [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

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

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

```
\ProcessOptions
⟨/package⟩
```

We first make sure that the sref [Koh13b] and graphicx packages are loaded.

```
⟨*cls⟩
\RequirePackage{sref}
\RequirePackage{graphicx}
```

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

```
\DeclareOption{letterpaper}
   {\setlength\paperheight {11in}%
    \setlength\paperwidth  {8.5in}}
\DeclareOption{a4paper}
   {\setlength\paperheight {297mm}%
    \setlength\paperwidth  {210mm}}
\ExecuteOptions{letterpaper}
\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).

```
\input{size10.clo}
\pagenumbering{roman}
⟨/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.

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

```
use strict;
use LaTeXML::Package;
RequirePackage('metakeys');
```

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

```
RegisterNamespace('cnx'=>"http://cnx.rice.edu/cnxml");
RegisterNamespace('md'=>"http://cnx.rice.edu/mdml/0.4");
RegisterNamespace('bib'=>"http://bibtexml.sf.net/");
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.

```
DocType("cnx:document",
"-//CNX//DTD CNXML 0.5 plus LaTeXML//EN",
"../dtd/cnxml+ltxml.dtd",
'#default'=>"http://cnx.rice.edu/cnxml",
       'md'=>"http://cnx.rice.edu/mdml/0.4",
       'bib'=>"http://bibtexml.sf.net/",
       'm'=>"http://www.w3.org/1998/Math/MathML",
       '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] EdN:9

```
NewCounter('@itemizei',   'document',   idprefix=>'I');
NewCounter('@itemizeii',  '@itemizei',  idprefix=>'I');
NewCounter('@itemizeiii', '@itemizeii', idprefix=>'I');
NewCounter('@itemizeiv',  '@itemizeiii',idprefix=>'I');
NewCounter('@itemizev',   '@itemizeiv', idprefix=>'I');
NewCounter('@itemizevi',  '@itemizev',  idprefix=>'I');

NewCounter('enumi',  '@itemizei',   idprefix=>'i');
NewCounter('enumii', '@itemizeii',  idprefix=>'i');
NewCounter('enumiii','@itemizeiii', idprefix=>'i');
NewCounter('enumiv', '@itemizeiv',  idprefix=>'i');
# A couple of more levels, since we use these for ID's!
NewCounter('enumv',  '@itemizev',   idprefix=>'i');
NewCounter('enumvi', '@itemizevi',  idprefix=>'i');

DefMacro('\theenumi',           '\arabic{enumi}');
DefMacro('\theenumii',          '\alph{enumii}');
DefMacro('\theenumiii',         '\roman{enumiii}');
DefMacro('\theenumiv',          '\Alph{enumiv}');

NewCounter('equation', 'document', idprefix=>'E');
DefMacro('\theequation', '\arabic{equation}');
DefMacro('\textwidth','16cm');
```

And another thing that is now needed:

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

```
Let('\thedocument@ID','\@empty');
⟨/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`

```
⟨*cls⟩
\addmetakey{cnxmodule}{name}
\srefaddidkey{cnxmodule}{id}
\newenvironment{cnxmodule}[1][]{\metasetkeys{cnxmodule}{#1}}{}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('cnxmodule','name','Semiverbatim');
DefKeyVal('cnxmodule','id','Semiverbatim');
DefEnvironment('{document}','<cnx:document>#body</cnx:document>',
      beforeDigest=> sub { AssignValue(inPreamble=>0); },
      afterDigest=> sub { $_[0]->getGullet->flush; return; });
DefEnvironment('{cnxmodule} OptionalKeyVals:cnxmodule',
      "<cnx:name>&GetKeyVal('#1','name')</cnx:name>\n#body\n",
      afterDigestBegin => sub {
 AssignValue('cnxmodule_id',
     KeyVal($_[1]->getArg(1), 'id')->toString,
     'global');
      });#$
Tag('cnx:document', afterClose => sub {
     $_[1]->setAttribute('id', LookupValue('cnxmodule_id'));
   });
⟨/ltxml⟩
```

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

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

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

```
⟨*cls⟩
\newcounter{section}
\srefaddidkey{sectioning}{id}
\newenvironment{csection}[2][]%
{\stepcounter{section}\strut\\[1.5ex]\noindent%
{\Large\bfseries\arabic{section}.~{#2}}\\[1.5ex]
\metasetkeys{sectioning}{#1}}
{}
\newcounter{subsection}[section]
\newenvironment{csubsection}[2][]
{\refstepcounter{subsection}\strut\\[1ex]\noindent%
{\large\bfseries{\arabic{section}.\arabic{subsection}.~#2\\[1ex]}}%
\metasetkeys{sectioning}{#1}}%
{}
\newcounter{subsubsection}[subsection]
\newenvironment{csubsubsection}[2][]
{\refstepcounter{subsubsection}\strut\\[.5ex]\noindent
{\bfseries\arabic{section}.\arabic{subsection}.\arabic{subsubsecction}~#2\\[.5ex]}%
\metasetkeys{sectioning}{#1}}{}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('sectioning','id','Semiverbatim');
DefEnvironment('{csection}OptionalKeyVals:sectioning{}',
       "<cnx:section %&GetKeyVals(#1)>\n"
             . "?#2(<cnx:name>#2</cnx:name>\n)()"
             . "#body\n</cnx:section>\n");
DefEnvironment('{csubsection}OptionalKeyVals:sectioning{}',
       "<cnx:section %&GetKeyVals(#1)>\n"
             . "?#2(<cnx:name>#2</cnx:name>\n)()"
             . "#body\n</cnx:section>\n");
DefEnvironment('{csubsubsection}OptionalKeyVals:sectioning{}',
       "<cnx:section %&GetKeyVals(#1)>\n"
             . "?#2(<cnx:name>#2</cnx:name>\n)()"
             . "#body\n</cnx:section>\n");
⟨/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.

```
⟨*cls⟩
\srefaddidkey{para}{id}
\newenvironment{cpara}[1][]{\metasetkeys{para}{#1}}{\par}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('para','id','Semiverbatim');
DefEnvironment('{cpara} OptionalKeyVals:para','<cnx:para %&GetKeyVals(#1)>#body</cnx:para>');
sub number_para {
  my($document,$node,$whatsit)=@_;
  # Get prefix from first parent with an id.
  my(@parents)=$document->findnodes('ancestor::*[@id]',$node); # find 1st id'd parent.
  my $prefix= (@parents ? $parents[$#parents]->getAttribute('id')."." : '');
  # Get the previous number within parent; Worried about intervening elements around para's, bu
  my(@siblings)=$document->findnodes("preceding-sibling::cnx:para",$node);
  my $n=1;
  $n = $1+1  if(@siblings && $siblings[$#siblings]->getAttribute('id')=~/(\d+)$/);
  $node->setAttribute(id=>$prefix."p$n"); }
Tag('cnx:para',afterOpen=>\&number_para);
DefConstructor('\par',sub { $_[0]->maybeCloseElement('cnx:para'); },alias=>"\\par\n");
Tag('cnx:para', autoClose=>1, autoOpen=>1);
⟨/ltxml⟩
```

cnote

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

4.3 Mathematics

cequation

```
⟨*cls⟩
\srefaddidkey{equation}{id}
\newenvironment{cequation}[1][]%
{\metasetkeys{equation}{#1}\begin{displaymath}}
{\end{displaymath}}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('equation','id','Semiverbatim');
DefEnvironment('{cequation} OptionalKeyVals:equation',
      "<cnx:equation %&GetKeyVals(#1)>"
            . "<ltx:Math mode='display'>"
            . "<ltx:XMath>#body</ltx:XMath>"
            . "</ltx:Math></cnx:equation>",
      mode=>'display_math');
⟨/ltxml⟩
```

4.4 Rich Text

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

quote

```
⟨*cls⟩
\srefaddidkey{cquote}
\addmetakey{cquote}{type}
\addmetakey{cquote}{src}
\newenvironment{cquote}[1][]{%
\metasetkeys{cquote}{#1}\begin{center}\begin{minipage}{.8\textwidth}}{\end{minipage}\end{center
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('cquote','id','Semiverbatim');
DefKeyVal('cquote','type','Semiverbatim');
DefKeyVal('cquote','src','Semiverbatim');
DefEnvironment('{cquote} OptionalKeyVals:cquote',
               "<cnx:quote %&GetKeyVals(#1)>#body</cnx:quote>");
⟨/ltxml⟩
```

footnote

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

emph

```
⟨*ltxml⟩
DefConstructor('\emph{}',"<cnx:emphasis>#1</cnx:emphasis>");
⟨/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.

```
⟨*ltxml⟩
DefConstructor('\[',
       "<cnx:equation id='#id'>"
       . "<ltx:Math mode='display'>"
       .  "<ltx:XMath>"
       .   "#body"
       .  "</ltx:XMath>"
       . "</ltx:Math>"
       ."</cnx:equation>",
       beforeDigest=> sub{ $_[0]->beginMode('display_math'); },
       captureBody=>1,
       properties=> sub { RefStepID('equation') });
DefConstructor('\]'  ,"",beforeDigest=> sub{ $_[0]->endMode('display_math'); });
⟨/ltxml⟩
```

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

EdN:10

[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

```
⟨*ltxml⟩
DefConstructor('\[',
       "<cnx:equation id='#id'>"
       . "<ltx:Math mode='display'>"
       .  "<ltx:XMath>"
       .   "#body"
       .  "</ltx:XMath>"
       . "</ltx:Math>"
       ."</cnx:equation>",
       beforeDigest=> sub{ $_[0]->beginMode('display_math'); },
       captureBody=>1,
       properties=> sub { RefStepID('equation') });
DefConstructor('\]'  ,"",beforeDigest=> sub{ $_[0]->endMode('display_math'); });

DefMacro('\eqnarray',    '\@@eqnarray\@start@alignment');
DefMacro('\endeqnarray', '\@finish@alignment\end@eqnarray');
DefMacro('\csname eqnarray*\endcsname',    '\@@eqnarray*\@start@alignment');
DefMacro('\csname endeqnarray*\endcsname', '\@finish@alignment\end@eqnarray');
DefConstructor('\@@eqnarray OptionalMatch:* AlignmentBody:\end@eqnarray',
       sub {
 my($document,$star,$body,%props)=@_;
 $document->openElement('cnx:equation',refnum=>$props{refnum},id=>$props{id});
 $document->openElement('ltx:Math',mode=>'display');
 $document->openElement('ltx:XMath');
 constructAlignment($document,$body,attributes=>{name=>'eqnarray'});
 $document->closeElement('ltx:XMath');
 $document->closeElement('ltx:Math');
 $document->closeElement('cnx:equation'); },
       mode=>'display_math',
       beforeDigest=>sub { alignmentBindings('rcl'); },
       properties=> sub { ($_[1] ? RefStepID('equation') : RefStepCounter('equation')); },
       afterDigest=>sub {
 $_[1]->setProperty(body=>$_[1]->getArg(2));},# So we get TeX
       reversion=>'\begin{eqnarray#1}#2\end{eqnarray#1}');
⟨/ltxml⟩
```

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

```
⟨*cls⟩
\newcommand\litem[2][]{\item[#1]\label{#2}}
⟨/cls⟩
⟨*ltxml⟩
Tag('cnx:item', autoClose=>1);
DefConstructor('\item[]',"<cnx:item>?#1(<cnx:name>#1</cnx:name>)");
DefConstructor('\litem[]{}',"<cnx:item id='#2'>?#1(<cnx:name>#1</cnx:name>)");
DefConstructor('\itemize@item[]',
```

[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

```
      "<cnx:item id='#id'>?#1(<cnx:name>#1</cnx:name>)",
      properties=>sub{ RefStepItemCounter(); });
DefConstructor('\enumerate@item[]',
      "<cnx:item id='#id'>?#1(<cnx:name>#1</cnx:name>)",
      properties=>sub{ RefStepItemCounter(); });
DefConstructor('\description@item[]',
      "<cnx::item id='#id'>?#1(<cnx:name>#1</cnx:name>)",
      properties=>sub{ RefStepItemCounter(); });
AssignValue(itemlevel=>0);
DefEnvironment('{itemize}',
      "<cnx:list id='#id' type='itemize'>#body</cnx:list>",
      properties=>sub { beginItemize('itemize'); });
DefEnvironment('{enumerate}',
      "<cnx:list type='enumerate'  id='#id'>#body</cnx:list>",
      properties=>sub { beginItemize('enumerate'); });
DefEnvironment('{description}',
      "<cnx:list  type='description'  id='#id'>#body</cnx:list>",
      properties=>sub { beginItemize('description'); });
⟨/ltxml⟩
```

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

```
⟨*ltxml⟩
DefEnvironment('{center}','#body');
DefEnvironment('{minipage}{}','#body');
DefEnvironment('{small}','#body');
DefEnvironment('{footnotesize}','#body');
DefEnvironment('{tiny}','#body');
DefEnvironment('{scriptsize}','#body');
⟨/ltxml⟩
```

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

4.5 Statements

cexample

```
⟨*cls⟩
\srefaddidkey{example}
\addmetakey{example}{name}
\newenvironment{cexample}[1][]{\metasetkeys{example}{#1}
{\ifx\example@name\@empty\else\noindent\bfseries{\example@name}\fi}}
{}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('example','id','Semiverbatim');
DefEnvironment('{cexample}OptionalKeyVals:example',
               "<cnx:example %&GetKeyVals(#1)>#body</cnx:example>");
⟨/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

```
⟨*cls⟩
\newcounter{cexercise}
\srefaddidkey{cexercise}
\addmetakey{cexercise}{name}
\newenvironment{cexercise}[1][]{\metasetkeys{cexercise}{#1}
{\ifx\cexercise@name\@empty\else\stepcounter{cexercise}\noindent\bfseries{\cexercise@name~\arab
{}
\srefaddidkey{cproblem}
\newenvironment{cproblem}[1][]{\metasetkeys{cproblem}{#1}}{}{}
\srefaddidkey{csolution}
\newenvironment{csolution}[1][]{\metasetkeys{csolution}{#1}}{\par\noindent\bfseries{Solution}}{
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('cexercise','id','Semiverbatim');
DefKeyVal('cexercise','name','Semiverbatim');
DefEnvironment('{cexercise}OptionalKeyVals:exercise',
                 "<cnx:exercise ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>"
                 . "#body"
               . "</cnx:exercise>");
DefKeyVal('cproblem','id','Semiverbatim');
DefKeyVal('cproblem','name','Semiverbatim');
DefEnvironment('{cproblem}OptionalKeyVals:cproblem',
                 "<cnx:problem ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>"
            . "?&defined(&GetKeyVal(#1,'name'))(<cnx:name>&GetKeyVal(#1,'name')</cnx:name>\n)()"
                 . "#body"
               . "</cnx:problem>");
DefKeyVal('csolution','id','Semiverbatim');
DefKeyVal('csolution','name','Semiverbatim');
DefEnvironment('{csolution}OptionalKeyVals:cproblem',
                 "<cnx:solution ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>"
            . "?&defined(&GetKeyVal(#1,'name'))(<cnx:name>&GetKeyVal(#1,'name')</cnx:name>\n)()"
                 . "#body"
               . "</cnx:solution>");
⟨/ltxml⟩
```

`crule`

```
⟨*cls⟩
\srefaddidkey{rule}
\addmetakey{rule}{name}
\addmetakey{rule}{type}
\newenvironment{crule}[1][]{\metasetkeys{rule}{#1}%
{\noindent\bfseries{\rule@type:}\ifx\rule@name\@empty\else~(\rule@name)\fi}}%
```

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

```
{}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('rule','id','Semiverbatim');
DefKeyVal('rule','name','Semiverbatim');
DefKeyVal('rule','type','Semiverbatim');
DefEnvironment('{crule}OptionalKeyVals:rule',
                "<cnx:rule ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')() type='&Get
     . "?&defined(&GetKeyVal(#1,'name'))(<cnx:name>&GetKeyVal(#1,'name')</cnx:name>\n)()"
     . "\n#body\n"
     . "</cnx:rule>\n");
⟨/ltxml⟩
```

statement

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

proof

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

definition

```
⟨*cls⟩
\srefaddidkey{definition}
\addmetakey{definition}{term}
\addmetakey{definition}{seealso}
\newenvironment{definition}[1][]{\metasetkeys{definition}{#1}{\noindent\bfseries{Definition:}}}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('definition','id','Semiverbatim');
DefKeyVal('definition','term','Semiverbatim');
DefKeyVal('definition','seealso','Semiverbatim');
DefEnvironment('{definition}OptionalKeyVals:definition',
                "<cnx:definition ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>\n"
     . "?&defined(&GetKeyVal(#1,'term'))(<cnx:term>&GetKeyVal(#1,'term')</cnx:term>\n)()"
     . "\n#body\n"
     . "?&defined(&GetKeyVal(#1,'seealso'))(<cnx:seealso><cnx:term>&GetKeyVal(#1,'term')</cnx:t
     . "</cnx:definition>\n");
⟨/ltxml⟩
```

cmeaning

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

4.6 Conexxions

cnxn

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

link

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

cfigure The cfigure only gives us one of the possible instances of the <figure> element[13].[14] In LaTeX, we just pipe the size information through to includegraphics, in LaTeXML, we construct the CNXml structure[15]

EdN:13
EdN:14
EdN:15

```
⟨*cls⟩
```

[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?

```
\srefaddidkey{cfigure}
\addmetakey{cfigure}{type}
\addmetakey{cfigure}{caption}
\newcounter{figure}
\newcommand\cfigure[3][]{% cnx_keys, graphicx_keys, path
\begin{center}%
\includegraphics[#2]{#3}%
\metasetkeys{cfigure}{#1}\sref@target%
\ifx\cfigure@caption\@empty\else
\par\noindent Figure\refstepcounter{figure} {\arabic{figure}}: \cfigure@caption%
\protected@edef\@currentlabel{\arabic{figure}}%
\sref@label@id{Figure \thefigure}\fi
\end{center}}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('cfigure','id','Semiverbatim');
DefKeyVal('cfigure','name','Semiverbatim');
DefKeyVal('cfigure','type','Semiverbatim');
DefKeyVal('cfigure','caption','Semiverbatim');
DefConstructor('\cfigure OptionalKeyVals:cfigure Semiverbatim Semiverbatim',
               "<cnx:figure ?&defined(&GetKeyVal(#1,'id'))(id='&GetKeyVal(#1,'id')')()>"
          . "?&defined(&GetKeyVal(#1,'name'))(<cnx:name>&GetKeyVal(#1,'name')</cnx:name>\n)()"
               . "<cnx:media type='&GetKeyVal(#1,'type')' src='#3'/>"
          . "?&defined(&GetKeyVal(#1,'caption'))(<cnx:caption>&GetKeyVal(#1,'caption')</cnx:cap
             . "</cnx:figure>");
⟨/ltxml⟩
```

ccite

```
⟨*cls⟩
\addmetakey{ccite}{src}
\newcommand\ccite[2][]{\metasetkeys{ccite}{#1}\emph{#2}}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('ccite','src','Semiverbatim');
DefConstructor('\ccite OptionalKeyVals:ccite {}','<cnx:cite %&GetKeyVals(#1)>#2</cnx:cite>');
⟨/ltxml⟩
```

term

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

4.7 Metadata

metadata

```
⟨*cls⟩
```

```
\addmetakey{metadata}{version}
\addmetakey{metadata}{created}
\addmetakey{metadata}{revised}
\newsavebox{\metadatabox}
\newenvironment{metadata}[1][]%
{\noindent\hfill\begin{lrbox}{\metadatabox}
\begin{minipage}{.8\textwidth}%
{\Large\bfseries CNX Module: \cnx@name\hfill\strut}\\[2ex]}%
{\end{minipage}\end{lrbox}\fbox{\usebox\metadatabox}\hfill}
% \newenvironment{metadata}[1][]%
% {\noindent\strut\hfill\begin{lrbox}{\metadatabox}\begin{minipage}{10cm}%
% {\strut\hfill\Large\bfseries CNX Module: \cnx@name\hfill\strut}\\[2ex]}%
% {\end{minipage}\end{lrbox}\fbox{\usebox\metadatabox}\hfill\strut\\[3ex]}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('metadata','version','Semiverbatim');
DefKeyVal('metadata','created','Semiverbatim');
DefKeyVal('metadata','revised','Semiverbatim');
DefEnvironment('{metadata}OptionalKeyVals:metadata',
       "<cnx:metadata>\n"
     . "<md:version>&GetKeyVal('#1','version')</md:version>\n"
     . "<md:created>&GetKeyVal('#1','created')</md:created>\n"
     . "<md:revised>&GetKeyVal('#1','revised')</md:revised>\n"
     . "#body\n"
     . "</cnx:metadata>");
⟨/ltxml⟩
```

authorlist

```
⟨*cls⟩
\newenvironment{authorlist}{{\bfseries{Authors}:~}}{\\[1ex]}
⟨/cls⟩
⟨*ltxml⟩
DefEnvironment('{authorlist}',"<md:authorlist>#body</md:authorlist>");
⟨/ltxml⟩
```

maintainerlist

```
⟨*cls⟩
\newenvironment{maintainerlist}{{\bfseries{Maintainers}:~}}{\\[1ex]}
⟨/cls⟩
⟨*ltxml⟩
DefEnvironment('{maintainerlist}',"<md:maintainerlist>#body</md:maintainerlist>");
⟨/ltxml⟩
```

cnxauthor

```
⟨*cls⟩
\srefaddidkey{auth}
\addmetakey{auth}{honorific}
\addmetakey{auth}{firstname}
\addmetakey{auth}{other}
```

```
\addmetakey{auth}{surname}
\addmetakey{auth}{lineage}
\addmetakey{auth}{email}
\newcommand\cnxauthor[1][]{\metasetkeys{auth}{#1}\auth@first~\auth@sur,}
⟨/cls⟩
⟨*ltxml⟩
DefKeyVal('auth','id','Semiverbatim');
DefKeyVal('auth','firstname','Semiverbatim');
DefKeyVal('auth','surname','Semiverbatim');
DefKeyVal('auth','email','Semiverbatim');
DefConstructor('\cnxauthor OptionalKeyVals:auth',
       "<md:author id='&GetKeyVal('#1','id')'>\n"
     . "?&defined(&GetKeyVal(#1,'honorific'))(<md:honorific>&GetKeyVal('#1','honorific')</md:ho
     . "?&defined(&GetKeyVal(#1,'firstname'))(<md:firstname>&GetKeyVal('#1','firstname')</md:fi
     . "?&defined(&GetKeyVal(#1,'other'))(<md:other>&GetKeyVal('#1','other')</md:other>\n)()"
     . "?&defined(&GetKeyVal(#1,'surname'))(<md:surname>&GetKeyVal('#1','surname')</md:surname>
             . "?&defined(&GetKeyVal(#1,'lineage'))(<md:lineage>&GetKeyVal('#1','lineage')</md:
             . "?&defined(&GetKeyVal(#1,'email'))(<md:email>&GetKeyVal('#1','email')</md:email>
     . "</md:author>\n");
⟨/ltxml⟩
```

maintainer

```
⟨*cls⟩
\newcommand\maintainer[1][]{\metasetkeys{auth}{#1}\auth@first~\auth@sur,}
⟨/cls⟩
⟨*ltxml⟩
DefConstructor('\maintainer OptionalKeyVals:auth',
       "<md:maintainer id='&GetKeyVal('#1','id')'>\n"
     . "?&defined(&GetKeyVal(#1,'honorific'))(<md:honorific>&GetKeyVal('#1','honorific')</md:ho
     . "?&defined(&GetKeyVal(#1,'firstname'))(<md:firstname>&GetKeyVal('#1','firstname')</md:fi
     . "?&defined(&GetKeyVal(#1,'other'))(<md:other>&GetKeyVal('#1','other')</md:other>\n)()"
     . "?&defined(&GetKeyVal(#1,'surname'))(<md:surname>&GetKeyVal('#1','surname')</md:surname>
             . "?&defined(&GetKeyVal(#1,'lineage'))(<md:lineage>&GetKeyVal('#1','lineage')</md:
             . "?&defined(&GetKeyVal(#1,'email'))(<md:email>&GetKeyVal('#1','email')</md:email>
     . "</md:maintainer>\n");
⟨/ltxml⟩
```

keywordlist

```
⟨*cls⟩
\newenvironment{keywordlist}{\bfseries{Keywords}:~}{\\[1ex]}
⟨/cls⟩
⟨*ltxml⟩
DefEnvironment('{keywordlist}',"<md:keywordlist>\n#body\n</md:keywordlist>");
⟨/ltxml⟩
```

keyword

```
⟨*cls⟩
\newcommand\keyword[1]{#1,}
⟨/cls⟩
```

```
⟨*ltxml⟩
DefConstructor('\keyword {}',"<md:keyword>#1</md:keyword>");
⟨/ltxml⟩
```

cnxabstract

```
⟨*cls⟩
\newenvironment{cnxabstract}%
{\par\noindent\strut\hfill\begin{minipage}{10cm}{\bfseries{Abstract}:~}}%
{\end{minipage}\hfill}
⟨/cls⟩
⟨*ltxml⟩
DefEnvironment('{cnxabstract} OptionalKeyVals:cnxabstract',
       "<md:abstract>\n#body\n</md:abstract>\n");
1;
⟨/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.