# Literate Sources for Content Dictionaries: a Progress Report

Lars Hellström

Division of Applied Mathematics, The School of Education, Culture and Communication, Mälardalen University, Box 883, 721 23 Västerås, Sweden; lars.hellstrom@residenset.net

**Abstract**

At OM2013, the author suggested and sketched a system that would use LaTeX documents as Literate Programming sources for content dictionaries. This paper reports on the progress that has since been made with this system. One important milestone that has been reached is that valid `.ocd` files with `CDDefinition`s, `FMP`s, `CMP`s, and `Example`s are being generated.

## 1 Overview

In [1], the idea of applying literate programming techniques to the task of creating OpenMath content dictionaries was raised, various possibilities were analysed, and a concrete design was sketched. The primary characteristic of this design is that all machine-readable material of the content dictionary is encoded using special-purpose markup in a LaTeX document. Typesetting said document using standard LaTeX will on the one hand produce a printed representation of the information so encoded, and on the other hand generate those XML documents that formally define the content dictionary. In March 2014, development had reached the point where the system could generate an `.ocd` file that would validate against the `omcd2.rng` schema, and in June it was used for an original content dictionary submitted to the OpenMath 2014 workshop.

Concretely, the system consists of a LaTeX 2$_\varepsilon$ package called `openmathcd`. This package defines a number of commands and environments that constitute the special-purpose markup for encoding content dictionaries. Among these are the `OpenMathCD` environment, which encloses the material for a content dictionary, and the `CDDefinition` environment, which encloses the material for one symbol within a content dictionary. There are also numerous commands and environments for encoding OpenMath objects.

In more detail, what the package currently can do is:

- generate every kind of element allowed in an `.ocd` file,
- generate FMPs and Examples with embedded OpenMath objects,
- typeset OpenMath objects (with or without simultaneously writing them to file).

Some envisioned things it cannot yet do, but which would only take a small amount of programming, are:

- generate `OME`, `OMB`, `OMF`, `OMSTR`, `OMR`, or `OMFOREIGN` elements (so far, there has been no need for them in the content dictionaries generated),
- generate `.sts` files.

One envisioned thing which may require a bit more thought, mostly to design a sensible user interface, is:

- generate file(s) defining notation for symbols.

# 2 Details

A file archive containing the current state as of 2014-06-07 of the openmathcd package—`.dtx` sources as well as a ready-to-use `.sty` files—can be downloaded from

> `http://www.mdh.se/polopoly_fs/1.60502!/Menu/general/column-content/attachment/list4.zip`

This archive also contains an example document (`list4.tex`) and the content dictionary generated from it (`list4.ocd`), which is the submitted original content dictionary mentioned above.

## 2.1 Object markup

The openmathcd markup for objects is patterned after the XML encoding for these, but with some basic adjustments to fit LaTeX syntax. Markup for compound objects are environments, whereas basic objects are expressed as commands. The base grammar for an ⟨*omel*⟩ may be stated as

⟨*omel*⟩ ⟶ `\OMV{`⟨*name*⟩`}`
| `\OMI{`⟨*optional sign*⟩⟨*digits*⟩`}`
| `\OMS[`⟨*cdbase*⟩`]{`⟨*cd*⟩`}{`⟨*name*⟩`}`
| `\begin{OMA}` ⟨*omel*⟩$^+$ `\end{OMA}`
| `\begin{OMBIND}` ⟨*omel*⟩ `\begin{OMBVAR}` ⟨*omel*⟩$^+$ `\end{OMBVAR}` ⟨*omel*⟩ `\end{OMBIND}`
| `\begin{OMATTR}` `\begin{OMATP}` $\big($⟨*omel*⟩ ⟨*omel*⟩$\big)^+$ `\end{OMATP}` ⟨*omel*⟩ `\end{OMATTR}`

which is fewer characters than the XML encoding for the leaf ⟨*omel*⟩s, but a few more for the compound ones; the primary gain is not in providing a significantly more compact encoding, but rather in switching from a format known chiefly by computer scientists (XML) to a format known by most mathematicians (LaTeX).

The LaTeX code fragments conforming to this grammar for an ⟨*omel*⟩ may be used to two ends, which typically happen in parallel: they may be transformed to valid XML encoding OpenMath objects written to a generated file, and they may be typeset to become part of the printed material in the document. For typesetting, there are currently two styles available: XML code style, which will be used inside an `OMOBJ` environment, and a "semiformula" style, which will be used inside `sfOMOBJ` and `semiformulae` environments. The `OMOBJ` and `sfOMOBJ` will cause the ⟨*omel*⟩ to be written to a generated file (where appropriate), whereas the `semiformulae` environment is more for facilitating discussions of OpenMath objects. Examples of such 'discussions' can be nonformalised proofs of mathematical theorems where details in the formal encoding of something as an OpenMath object are important.

A practical extension of the markup, which saves quite some typing, is the `OMAS` environment. Technically, it extends the above grammar for ⟨*omel*⟩ with the alternative

> `\begin{OMAS}[`⟨*cdbase*⟩`]{`⟨*cd*⟩`}{`⟨*name*⟩`}` ⟨*omel*⟩$^*$ `\end{OMAS}`

that (as far as encoding an OpenMath object is concerned) is equivalent to

> `\begin{OMA} \OMS[`⟨*cdbase*⟩`]{`⟨*cd*⟩`}{`⟨*name*⟩`}` ⟨*omel*⟩$^*$ `\end{OMA}`

Using this, the formula $2 + 2 = 4$ may be encoded as the ⟨*omel*⟩

```
\begin{OMAS}{relation1}{eq}
  \begin{OMAS}{arith1}{plus}
    \OMI{2} \OMI{2}
  \end{OMAS}
  \OMI{4}
\end{OMAS}
```

and an OMOBJ environment typesets that as

```
<OMOBJ>
  <OMA> <OMS cd="relation1" name="eq"/>
    <OMA> <OMS cd="arith1" name="plus"/>
      <OMI>2</OMI>
      <OMI>2</OMI>
    </OMA>
    <OMI>4</OMI>
  </OMA>
</OMOBJ>
```

whereas the sfOMOBJ environment may typeset it as

$$\mathbf{application}(\text{relation1.eq}, \mathbf{application}(\text{arith1.plus}, 2, 2), 4)$$

(although what should be the defaults in this latter style is at the time of writing very much in flux). The generated XML code is in both cases the same as that typeset by the OMOBJ environment, except that the generated code also has an xmlns="http://www.openmath.org/OpenMath" attribute on the OMOBJ element.

LaTeX parses ⟨*omel*⟩s by executing them, so the standard range of LaTeX programming tricks are available for further streamlining of markup.

## 2.2 Funny characters

One of the great challenges when generating code, especially code that may embed arbitrary strings (as is the case with for example CMPs), is to make sure that all characters are correctly encoded. The difficulty level increases even more when the source format has its own syntax rules that are different from those of the target format; an incomplete translation could result in a situation where users would have to know and counteract idiosyncracies of both the source and the target format. But thanks to using the harmless LaTeX package (included in the above archive) for handling character strings, users of openmathcd need only worry about handling LaTeX syntax, and may even use LaTeX markup for accented letters (which for mathematicians may be less confusing than locating them on the keyboard).

Generated XML files are always pure ASCII because that is all TeX can do portably, but the character set supported by openmathcd is full Unicode; numerical character entities are used extensively in the generated XML files, which will be well-formed. (Getting LaTeX to *typeset* unusual characters as appropriate glyphs can however be nontrivial.) openmathcd does not check that names only contain valid characters, but that is a trivial matter to verify using XML validation.

## 2.3 Canned strings

One design goal has been that users should not have to write long strings that are anyway fixed beforehand. One class of such strings are the XML namespaces, which are inserted automatically where needed. The longest canned string is however the copyright licence; the single command \StandardOMLicence will insert the full 29 lines of the standard licence (wrapped up in a CDComment element) into the generated file(s).

It would be a minor modification to also put cdbase and/or version attributes on each generated OMOBJ. The author would be interested to hear arguments for or against.

# 3 Moving forward

## 3.1 The importance of brevity

The XML encoding of an OpenMath object can be hard to read because the information is very spread out; there can be a lot of text between the name of a function and the name of the variable it is being applied to. The semiformula style is much closer to ordinary mathematical formulae, but they too do not achieve the same togetherness of formula elements as ordinary mathematical formulae do. One reason for this might be that many of the tokens in semiformulae are still too long to allow the eye to behold groups of them as units; reducing common tokens to single glyphs could overcome this.

Changing the long **application** token to a simple @ makes a significant difference (because it is *very* common), but then it is instead the names of the symbols which stand out as being long. When written semiformulae (or something very similar to them) have been hand-crafted, such as for example in [2], it is typical that also symbols (particularly the common ones) are given single glyph presentations: $\forall$ for `quant1#forall`, $=$ for `relation1#eq`, etc. Doing this for an explicit set of declared symbols is within the realm of what LaTeX macros can achieve, so it should probably be added as a feature to openmathcd.

## 3.2 Relation to standard enhancement

In several cases, it is hard to tell exactly how to further develop the openmathcd markup, because the correct direction depends on how the OpenMath standard will evolve. Some open tickets in the OpenMath Trac database,[1] and aspects of openmathcd they would affect, are:

| ticket | title | affects |
|---|---|---|
| #5 | FMP type=defining | `FMP` environment arguments |
| #128 | Make CDSignatures or Signature use cdbase | STS generation |
| #138 | CD's CDBase declaration is mandatory | `\CDBase` command |
| #139 | Symbol's default cdbase not specified correctly | `OMOBJ` attributes |
| #144 | Add Notation Definitions to OpenMath | Notation specification |
| #152 | Revising the Simple Type System | STS generation |

It should however be observed that even a partial resolution of some of these issues—for example defining a partial notation definition system, or defining a system abstractly even if not with a formal syntax—would be a great help, as it could allow development to take a few steps forward.

# References

[1] Lars Hellström. Literate sources for content dictionaries. Paper 22 in *MathUI, OpenMath, PLMMS and ThEdu Workshops and Work in Progress at the Conference on Intelligent Computer Mathematics*, CEUR Workshop Proceedings **1010**, 2013. `http://ceur-ws.org/Vol-1010/paper-22.pdf`

[2] Fulya Horozal, Michael Kohlhase, and Florian Rabe. Extending OpenMath with Sequences, pp. 58–72 in: *Intelligent Computer Mathematics, Work-in-Progress Proceedings*, Technical Reports of University of Bologna UBLCS-2011-04, 2011. `http://kwarc.info/frabe/Research/HKR_sequences_11.pdf`

[1] `https://trac.mathweb.org/OM3/query`