The MMT Manual

Florian Rabe
Jacobs University Bremen

November 18, 2013

This documentation is outdated and subsumed by the one in `../html/index.html`.

1 XML Syntax and MMT URIs

The MMT language and its semantics have been described in depth in [RK13]. Here we will focus on describing the XML syntax to someone who already has a general understanding of MMT.

The semantics of notations is described in Sect. **??**.

1.1 URIs

We define three data types for addressing MMT elements: the type *MMTURI* of MMT URIs and the types *Name* and *QName* of unqualified and qualified MMT names. They are defined by the following grammar:

MMT URI	$MMTURI$	::=	$N \mid M \mid S$
Namespace URI	N	::=	URI, no query, no fragment
Module URI	M	::=	$N?QName \mid ?/QName$
Symbol URI	S	::=	$M?QName \mid ??QName \mid ??/QName$
Unqualified name	$Name$	::=	$pchar^+$
Qualified name	$QName$	::=	$Name(/Name)^*$
	URI, $pchar$		see RFC 3986 [BLFM05]

Namespaces N have no semantics and only serve to disambiguate toplevel declarations. Modules are any kind of named resource that introduces a scope within which other resources (i.e., symbols) are introduced such as signatures, theories, ontologies. Modules may be nested, in which case they have qualified names. Symbols are any kind of named atomic resource such as constants, functions, predicates, sorts, axioms, theorems. The names of symbols may be qualified as well, which MMT uses to form qualified names for symbols induced by named imports.

We will use Q and R to range over qualified names. In a URI $N[?Q[?R]]$, N, Q, and R are called the namespace, module name, and symbol name, respectively. A URI of this form is called **absolute** if N is an absolute URI. Otherwise, it is called **relative** or an ***MMTURI* reference**. Every absolute MMT URI has exactly one of the following three forms: N, $N?Q$, or $N?Q?R$ where N is an absolute URI. Every relative MMT URI has exactly one of the following seven forms: n, $n?q$, $n?q?r$, $?/q$, $?/q?r$, $??r$, or $??/r$ where n is a relative (possibly empty) URI.

Note that a *pchar* may be any character permitted in a URI except for "/", "?", "#" "[", "]", and "%". Furtermore, all percent-encoded characters are permitted.

Example 1. In this example, we abbreviate `http://cds.omdoc.org/algebra/algebra.omdoc` with `A`.

`0/algebra/algebra.omdoc?monoid?unit` is an absolute symbol URI. It refers to the symbol `unit` declared in the module `monoid` declared in the document `A`.

The /-character in the module part separates submodules. `A?monoid/latex` refers to the module `latex` declared within the module `A?monoid` (e.g., a module containing notations to render monoids in Latex syntax).

The /-character in the symbol part separates named imports, called structures in MMT. Let `A?group?mon` refer to an import `mon` declared within the module `A?group` that imports the module `A?monoid`. Then `A?group?mon/unit` refers to the symbol `A?monoid?unit` imported via this import. In general, if the symbol part of an MMT URI has n components, then the first $n-1$ must be the names of named imports.

The resolution of an MMT URI reference u against an absolute base URI U is defined as follows:

1. u is of the form n, $n?q$, or $n?q?r$: u is resolved relative to the namespace N of U. (A possible module or symbol name in U are ignored.) If N' is the result of resolving n against N according to RFC 3986, then the resulting MMT URI is N', $N'?q$, or $N'?q?r$, respectively.
   Note that in the special case where n is empty, this implies $N' = n$. (Beware that software packages for the URI data type such as in Java 1.5 might implement the obsolete RFC 2396, where empty d was resolved in the same way as ..)

2. u is of the form $?/q$ or $?/q?r$: u is resolved relative to the namespace N and module name Q of U. (A possible symbol name in U is ignored. It is an error if U has no module name.) The resolution is $N?Q/q$ or $N?Q/q?r$, respectively.

3. u is of the form $??r$ or $??/r$ and $U = N?Q?R$. (It is an error if U is a module or document URI.) The resolution is $N?Q?r$ or $N?Q?R/r$, respectively.

Example 2. Assume a base URI `http://cds.omdoc.org/algebra/algebra.omdoc?group?mon`. The following table gives examples of resolutions of relative URIs for each of the above six cases. Here we abbreviate `http://cds.omdoc.org` with `O`.

URI	Resolution
`mathml.omdoc`	`O/algebra/mathml.omdoc`
`?group`	`O/algebra/algebra.omdoc?group`
`../logics/fol/fol.omdoc?fol?and`	`O/logics/fol/fol.omdoc?fol?and`
`?/latex`	`O/algebra/algebra.omdoc?group/latex`
`?/latex?circ`	`O/algebra/algebra.omdoc?group/latex?circ`
`??/unit`	`O/algebra/algebra.omdoc?group?mon/unit`

In addition to the above grammars, we introduce the following convention: *MMTURI*s that contain less than two occurrences of ?, can also be written with 2 ?s by appending ?. In other words, $n??$ and $N?Q?$ abbreviate N and $N?Q$, respectively.

Thus, (recalling that no ?-character may occur in URIs that have no query component) every absolute MMT URI can be written uniquely as a ?-separated triple. The components of this triple are a URI and two /-separated lists of strings.

Relationship with URIs Every absolute/relative *MMTURI* is also a legal absolute/relative URI. In particular, if we consider an *MMTURI* $N?Q?R$ as a URI, then $Q?R$ is its query component. Moreover, the *MMTURI* resolution of $n?q?r$ against $D?Q?R$ is identical to the usual resolution of relative URIs.

The situation is more complicated for those relative *MMTURI*s that begin with ?/ or ??. Here, the resolution must be implemented separately. This is unavoidable: In order to subsume common practices regarding XML namespaces, the module and symbol name must be put into the query component; but URIs do not permit relative resolution within the query component.

Relationship between OpenMath identifiers and *MMTURI*s. Every absolute *MMTURI* is a triple of namespace, module name, and symbol name. This corresponds directly to the cdbase-cd-name triple in OpenMath identifiers [BCC+04]. Note that OpenMath use the fragment component when forming URIs from OpenMath identifiers; *MMTURI*s avoid this because it would preclude efficient retrieval of individual symbols.

1.2 Document Level Elements

Label: `omdoc` (a document unit)

Attributes:

name	type	value
`name`	*MMTURI*	the optional name of the unit
`base`	*MMTURI*	the base URI for the unit's content, relative to base URI given by parent, empty by default

Children: `omdoc* & xref* & module*`

Comment: If this occurs as the root of a document that has a URL, then the name must be omitted or be equal to the last segment of that URL's path.

Label: `dref` (a reference to an external document unit)

Attributes:

name	type	value
`target`	*MMTURI*	the referenced document

Children:

Label: `mref` (a reference to an external document unit)

Attributes:

name	type	value
`target`	*MMTURI*	the referenced module

Children:

1.3 Module Level Elements

Label: `theory` (a theory)

Attributes:

name	type	value
`base`	*MMTURI*	the base URI of the module, relative to the base URI given by the parent, empty by default
`name`	*Name*	the name of the view
`meta`	*MMTURI*	the URI of the optional meta-theory, relative to base URI given by parent

Children: `(include* & symbol*) | definition{object}`

Comment: Here, a symbol can be a constant or a structure.

Comment: Depending on the children, we speak of *declared* and *defined* theories. In the latter case, the definiens is a theory expression.

Label: `view` (a theory morphism, postulated link)

Attributes:

name	type	value
`base`	*MMTURI*	the base URI of the module, relative to the document base, empty by default
`name`	*Name*	the name of the view
`from`	*MMTURI*	the domain of the view, relative to base URI given by parent
`to`	*MMTURI*	the codomain of the view, relative to base URI given by parent

Children: `(include* & symbol*) | definition{object}`

Comment: The symbols are assignments, i.e., their name must be the same as that of a corresponding symbol in the domain, their types are predetermined, and their definientia required.

Comment: Depending on the children, we speak of *declared* and *defined* views. In the latter case, the definiens is a morphism expression.

Label: `style` (a named set of notations)

Attributes:

name	type	value
`base`	*MMTURI*	the base URI of the module, relative to the document base, empty by default
`name`	*Name*	the name of the style
`for`	*MMTURI*	the base URI used for `for` attributes, relative to the module's base URI, empty by default
`defaults`	`use` | `ignore`	defaults to `use`, determines the treatment of default notations given in theories

Children: `(include* & notation*) | definition{notset}`

Comment: Definitions are actually not supported yet and only added for symmetry.

All module level elements are named. The base URI of a module defaults to the document URI. However, a differing URI may be provided with the `base` attribute of the module or an ancestor. The latter should only be used in generated documents because it prevents reference by location.

A module with name n and base URI B is addressable via the module URI $B?n$. Module URIs (but not module names) must be unique within a file.

A document unit with name n whose parent is addressable as D is addressable as D/n. Document unit names must be unique within a document unit.

1.4 Symbol Level Elements

Some symbol level elements are named (includes and notations optionally so). If a symbol with name n occurs in a module with URI M, the symbol is addressable via the URI $M?n$.

Label: `include` (inclusion of a theory/view/style into the containing theory/view/styles)

Attributes:

name	type	value
`from`	*MMTURI*	the included module, relative to containing module

Children:

Label: `constant` (e.g., a sort, function, predicate, judgment, or proof rule)

Attributes:

name	type	value
`name`	*Name*	the name of the constant

Children: `alias{name} & type{object}? & definition{object}? & notation{notation}? & role{string}`

Comment: Constants can occur both in theories and views. In the latter case, their type is predetermined and must be omitted, and their definiens is required.

Label: `structure` (named instantiation of another theory, definitional link)

Attributes:

name	type	value
`name`	*Name*	the name of the structure
`from`	*MMTURI*	the domain of the structure, relative to containing theory

Children: `(include* & symbol*) | definition{object}`

Comment: A can be an assignment to a constant or an assignment to a structure.

Comment: Depending on the children, we speak of *declared* and *defined* structures. In the latter case, the definiens is a morphism expression.

Every notation has a role. The permitted values of `role` are given in Sect. **??**. There are two ways to give notations, direct and via parameters:

Label: `notation` (a notation)

Attributes:

name	type	value
`name`	*Name*	the optional name of the notation
`for`	*MMTURI*	the optional URI to which the notation applies, relative to base URI of style
`role`	string	the simple role to which the notation applies
`wrap`	`true | false`	a flag specifying whether the notation is merged with more specific ones
`precedence`	$\mathbb{Z}^*$	the optional output precedence

Children: `pres`

Comment: A notation without a name has no URI. `precedence` may only be given if the role is bracketable.

Label: `notation` (a notation)

Attributes:

name	type	value
`name`	*Name*	the optional name of the notation
`for`	*MMTURI*	the optional URI to which the notation applies, relative to base URI of style
`role`	string	the simple role to which the notation applies
`precedence`	$\mathbb{Z}^*$	the optional output precedence
`fixity`	string	the optional fixity: `pre | post | in | inter | bind`
`application-style`	string	the optional application style: `math | lc`
`associativity`	string	the optional associativity: `none | left | right`
`implicit`	$\mathbb{N}$	the number of implicit arguments

Children:

Comment: A notation without a name has no URI. `precedence` may only be given if the role is bracketable.

1.5 Object Level Elements

The type `objects` represents Mmt terms. These are given as OpenMath objects wrapped in an OMOBJ element. Furthermore, morphism application of μ to ω is encoded in OpenMath using the special theory MMT with base MMT:

```
<OMA>
  <OMS base="MMT" module="mmt" name="morphism−application"/>
  μ
  ω
```

```
</OMA>
```

Module Expressions Objects may denote MMT theories and morphisms. Besides OMS elements referring to MMT theories, theory expressions can arise from, e.g., instantiation, union, and pushout. Besides OMS elements referring to MMT views and structures, morphism expressions can arise from, e.g., instantiation, identity, composition, unit, pushout. We use *MMT* to abbreviate http://omdoc.org/mmt.

Link $D?Q$:

```
<OMS base="D" module="Q"/>
```

Composition $\mu_1 \ldots \mu_n$:

```
<OMA>
  <OMS base="MMT" module="mmt" name="composition"/>
  μ1
  ...
  μn
</OMA>
```

Identity id_T:

```
<OMA>
  <OMS base="MMT" module="mmt" name="identity"/>
  T
</OMA>
```

Theories are encoded like links.

1.6 Resolving MMT URIs

Document level The scheme and authority of a URI D must resolve to some root directory. Then the path of D is resolved relative to that directory. For the resolution of paths, we treat the paths ending in / and those not ending in / as identical.

A path P is resolved relative to the directory D as follows:

- P is empty, then resolve to D.
- If $P = p/P'$ and p is a subdirectory of D, then resolve P' relative to D/p.
- If $P = p/P'$ and p is a file in D, then resolve P' relative to the content of that file (which must be an `omdoc` element).

A path P is resolved relative to an `omdoc` element O as follows:

- P is empty, then resolve to O.
- If $P = p/P'$ and p is the value of a `name` attribute of an `omdoc` child of O, then resolve P' relative to that child.

Note that this makes the file structure transparent in the following sense. Assume a directory D with files $n_1, \ldots, n_r$ containing the respective `omdoc` element O_i. Let O be the file containing

```
<omdoc>
  <omdoc name="n1">
    O1
  </omdoc>
  ...
  <omdoc name="nr">
    Or
  </omdoc>
</omdoc>
```

Then the resolutions of a path relative to D and O are the same.

We can implement this transparency with a standard apache web server: In every directory D add a file `index.omdoc` containing O as defined above and add a directive in the `.htaccess` file to serve `index.omdoc` as the directory listing. (If D should happen to contain a file `index.omdoc` already, we can pick any other file name for it.) Then apache's path resolution will correspond to the above definition for all paths that do not end in /.

The connection to directory listings is stressed if we use this alternative – semantically equivalent – definition of O:

```
<omdoc>
  <xref target="n_1"/>
  ...
  <xref target="n_r"/>
</omdoc>
```

Note that it is always legal to split an `omdoc` file into directories. However, the opposite is only legal if module names are unique across the directory.

Module Level A module level URI $D?Q$ is resolved by resolving D to an `omdoc` element O and then resolving Q relative to it as follows:

- If $Q = q$ is a single segment, then resolve to the module with name q in O. Here, the modules in O are the module level children of `omdoc`-descendants of O.
- If $Q = q/Q'$, then resolve Q' relative to the `omdoc`-wrapped resolution q.

Here we assume that there are no `base` attributes set in O.

Symbol Level A module level URI $D?Q?R$ is resolved by resolving $D?Q$ to a module level element M and then resolving R relative to it as follows:

- If $R = r$ is a single segment, then resolve to the symbol with name r in M. Here, the symbols in M are the symbol level children of `omdoc`-descendants of M.
- If $R = r/R'$, then resolve R' relative to the domain of the structure with name r in M and translate the result along said structure.

Note that the structure of nested `omdoc` elements is transparent to the resolution of modules and symbols. Thus, the uniqueness of module and symbol names, which is necessary to make resolution well-defined, nested `omdoc` elements must be disregarded.

2 Text Syntax

2.1 Module Level

Intuitively, modules are the named declarations that introduce scopes containing other named declarations (other modules or symbols).

2.2 Symbol Level

Intuitively, symbols are the atomic named declarations contained in modules.

2.3 Object Level

Intuitively, objects are the unnamed components occurring in named declarations. This includes in particular the actual MMT objects that occur as the types and definientia. But for the purposes of the text syntax, it also all other components such as notations and roles.

2.3.1 Text Notations

Currently, the best available write-up is in [GIR13].

2.3.2 Pragmatic Syntax

Currently, the best available write-up is in [GIR13].

2.3.3 Parsing

Currently, the best available write-up is in [GIR13].

2.4 Structural Delimiters

Within this document, we display them as D_{US}, D_{RS}, D_{GS}, and D_{FS}.

Editor Support jEdit supports setting up abbreviations that make entering any character easy. For example, one can use the following in the `mmt` section of jEdit's `abbrevs` file:

US | D_{US}
GS | D_{GS}
RS | D_{RS}
FS | D_{FS}

In fact, the `abbrevs` file provided with MMT sets up more sophisticated abbreviations that expand to declaration templates and in particular include the delimiters such as

```
constant \|  : $D_{US}$ = $D_{US}$ # $D_{US}$$D_{RS}$
```

ViM supports the insertion of control characters via the following hotkeys:

- group separator (displayed as ^]) - ctrl+shift+]
- record separator (^^) - ctrl+shift+^
- unit separator (^_) - ctrl+shift+_

Note that in command mode one can look up the ASCII code of a character by pressing `ga`.

Font Support To display MMT's special delimiters in text interfaces, it is advisable to use a font that has reasonable glyphs for them. GNU unifont is an example.

3 Archives

An MMT archive [HIJ+11] organizes connected documents in a file system structure. This is inspired by and similar to the project view in software engineering, specifically in Java. MMT provides archive-level functions for building and indexing document collections.

Archives can be manipulated via the API functions or via the shell (see Sect. **??**).

Dimensions MMT supports the following dimensions, which occur as toplevel folders in an archive. The following dimensions contain the same tree of subfolders and files.

- `source`: source files in any language, in particular MMT text syntax
- `compiled`: the files in MMT XML syntax
- `narration`: similar to the ones in `compiled` but containing only the narrative outline, i.e., all MMT modules are replaced with `mref` elements
- `notation`: index of notations
- `relational`: relational index
- `mws`: MathWebSearch [KŞ06] index

The names of the folders `source` and `compiled` may be changed by using the `compilation` key in the manifest (see below).

The `content` dimension contains one file for every MMT module. The directory structure follows that of the MMT namespaces. In particular, the folder immediate below `content` is named by the scheme and the authority of the namespace URI. (`..` is used as a separator instead of `://` to avoid character restrictions of operating systems.) Below these folders, one subfolder is used for every path segment.

The `scala` dimension contains one file for every file in the `content` directory. It contains the Scala code extract from the MMT modules as described in [KMR13].

Manifest Archives must contain a file `META-INT/MANIFEST.MF`. Syntactically, this file consists of line-wise `key:value` pairs. Except for colons in keys, all characters are allowed. However, beginning and trailing whitespace in keys and values is ignored. Empty lines and lines starting with `//` are ignored. Unless otherwise mentioned, keys should occur only once and later occurrences overwrite previous values.

The following keys have a predefined meaning and are used by MMT:

- `id`: The identifier of the archive.
- `compilation`: A string of the form `format_1 @ folder_1 -> ... -> format_n @ folder_n`. This identifies a compilation chain used for the compilation of the archive. `format_i` defines the format of the files in `folder_i`. `folder_1` and `folder_n` identify the folders storing the `source` and `compiled` dimensions; consequently, `format_1` is the source format of the archive, and `format_n` should be `omdoc`.
- `narration-base`: This is a URI that gives the base URI for the `source` documents and all dimensions with the same folder structure. It also doubles as the URI of the whole archive.
- `source-base`: This is a URI that gives the base URI for all modules in the `source` documents.

Build Processes MMT provides to build the dimensions of an archive, starting from the source:

compile `compiled` is generated from `source`.

content (indexing) `content`, `narration`, `relational`, and `notation` are generated generically from `compiled`.

extract (code generation) `scala` is generated from `content`.

mar (packaging) `id.mar` is generated by packaging all dimensions.

The compilation works as follows. According to the compilation chain defined in the manifest, compilers are called that read the files in `folder_i` and write files into `folder_i+1`. For each step in the chain, the appropriate compiler is chosen according to the `format_i`. The available compilers are provided as extensions (see Sect. **??**).

References

[BCC+04] S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaetano, and M. Kohlhase. The Open Math Standard, Version 2.0. Technical report, The Open Math Society, 2004. See http://www.openmath.org/standard/om20.

[BLFM05] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform resource identifier (URI): Generic syntax. RFC 3986, Internet Engineering Task Force (IETF), 2005.

[GIR13] D. Ginev, M. Iancu, and F. Rabe. Integrating Content and Narration-Oriented Document Formats. see http://kwarc.info/frabe/Research/GIR_mmtlatex_13.pdf, 2013.

[HIJ+11] F. Horozal, A. Iacob, C. Jucovschi, M. Kohlhase, and F. Rabe. Combining Source, Content, Presentation, Narration, and Relational Representation. In J. Davenport, W. Farmer, F. Rabe, and J. Urban, editors, *Intelligent Computer Mathematics*, pages 212–227. Springer, 2011.

[KMR13] M. Kohlhase, F. Mance, and F. Rabe. A Universal Machine for Biform Theory Graphs. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *Intelligent Computer Mathematics*, pages 82–97. Springer, 2013.

[KŞ06] M. Kohlhase and I. Şucan. A Search Engine for Mathematical Formulae. In T. Ida, J. Calmet, and D. Wang, editors, *Artificial Intelligence and Symbolic Computation*, pages 241–253. Springer, 2006.

[RK13] F. Rabe and M. Kohlhase. A Scalable Module System. *Information and Computation*, 230(0):1–54, 2013.