Mathematical Markup Language (MathML) Version 3.0

W3C Working Draft 04 June 2009

This version: http://www.w3.org/TR/2009/WD-MathML3-20090604/
Latest MathML 3 version: http://www.w3.org/TR/MathML3/
Latest MathML Recommendation: http://www.w3.org/TR/MathML/
Previous version:
http://www.w3.org/TR/2008/WD-MathML3-20081117/
Editors: David Carlisle (NAG)
Patrick Ion (Mathematical Reviews, American Mathematical Society)
Robert Miner (Design Science, Inc.)
Principal Authors: Ron Ausbrooks, Stephen Buswell, David Carlisle, Giorgi Chavchanidze, Stéphane Dalmas, Stan Devitt, Angel Diaz, Sam Dooley, Roger Hunter, Patrick Ion, Michael Kohlhase, Azzeddine Lazrek, Paul Libbrecht, Bruce Miller, Robert Miner, Murray Sargent, Bruce Smith, Neil Soiffer, Robert Sutor, Stephen Watt

In addition to the HTML version, this document is also available in these non-normative formats: XHTML+MathML version and PDF version.

Abstract

This specification defines the Mathematical Markup Language, or MathML. MathML is an XML application for describing mathematical notation and capturing both its structure and content. The goal of MathML is to enable mathematics to be served, received, and processed on the World Wide Web, just as HTML has enabled this functionality for text.

This specification of the markup language MathML is intended primarily for a readership consisting of those who will be developing or implementing renderers or editors using it, or software that will communicate using MathML as a protocol for input or output. It is *not* a User's Guide but rather a reference document.

MathML can be used to encode both mathematical notation and mathematical content. About thirty-eight of the MathML tags describe abstract notational structures, while another about one hundred and seventy provide a way of unambiguously specifying the intended meaning of an expression. Additional chapters discuss how the MathML content and presentation elements interact, and how MathML renderers might be implemented and should interact with browsers. Finally, this document addresses the issue of special characters used for mathematics, their handling in MathML, their presence in Unicode, and their relation to fonts.

While MathML is human-readable, in all but the simplest cases, authors use equation editors, conversion programs, and other specialized software tools to generate MathML. Several versions of such MathML tools exist, and more, both freely available software and commercial products, are under development.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document is a W3C Public Working Draft produced by the W3C Math Working Group as part of the W3C Math Activity. The goals of the W3C Math Working Group are discussed in the W3C Math WG Charter (revised July 2006). A list of participants in the W3C Math Working Group is available.

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This sixth Public Working Draft specifies a new version of the the Mathematical Markup Language, MathML 3.0 which is at present under active development. The Math WG hopes this draft will permit informed feedback. There is a description of some considerations underlying this work in the W3C Math WG's public Roadmap [roadmap]. Feedback should be sent to the Public W3C Math mailing list .

The MathML 2.0 (Second Edition) specification has been a W3C Recommendation since 2001. After its recommendation, a W3C Math Interest Group collected reports of experience with the deployment of MathML and identified issues with MathML that might be ameliorated. The rechartering of a Math Working Group allows the revision to MathML 3.0 in the light of that experience, of other comments on the markup language, and of recent changes in specifications of the W3C and in the technological context. MathML 3.0 does not signal any change in the overall design of MathML. The major additions in MathML 3 are support for bidirectional layout, better linebreaking and explicit positioning, elementary math notations, and a new strict content MathML vocabulary with well-defined semantics generated from formal content dictionaries. The MathML 3 Specification has also been restructured.

Public discussion of MathML and issues of support through the W3C for mathematics on the Web takes place on the public mailing list of the Math Working Group (list archives). To subscribe send an email to www-math-request@w3.org with the word `subscribe` in the subject line.

Please report errors in this document to www-math@w3.org.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

The basic chapter structure of this document is based on the earlier MathML 2.0 Recommendation [MathML2]. That MathML 2.0 itself was a revision of the earlier W3C Recommendation MathML 1.01 [MathML1]; MathML 3.0 is a revision of the W3C Recommendation MathML 2.0. It differs from it in that all previous chapters have been updated, some new elements and attributes added and some deprecated. This Public Working Draft differs in structure from the initial Public Working Draft as renewed efforts to separate the formal from the explanatory have resulted in eight chapters not seven. Much has been moved to separate documents containing Primer material, material on Characters and Entities and on the MathML DOM. First Working Drafts of these documents will be published soon. A current list of open issues, pointing into the relevant places in the draft, follows the Table of Contents.

The present draft is an incremental one making public some of the results of Math Working Group work in recent months. The biggest difference this time is in Chapter 4, although there have been smaller ameliorations throughout the specification. A more detailed description of changes from the previous Recommendation follows.

- With the second Working Draft, much of the non-normative explication that formerly was found in Chapters 1 and 2, and many examples from elsewhere in the previous MathML specifications, were removed from the MathML3 specification and planned to be incorporated into a MathML Primer being prepared as a separate document. It is expected this will help

the use of this formal MathML3 specification as a reference document in implementations, and offer the new user better help in understanding MathML's deployment. The remaining content of Chapters 1 and 2 is being edited to reflect the changes elsewhere in the document, and in the rapidly evolving Web environment. Some of their text used to go back to early days of the Web and XML, and its explanations are now commonplace.

- Chapter 3, on presentation-oriented markup, in this draft adds new material on linebreaking and on markup for elementary math notations. Material introduced in the last draft revising the `mpadded` and `maction` elements has been further revised as a result of active discussion. In addition, the layout of schemata such as that for long division have been carefully revised with an eye to the demands mathematics as an international language. This has resulted in the introduction of new `mstack`, `mlongdiv` and other associated elements. Earlier work, as recorded in the W3C Note Arabic mathematical notation, has allowed clarification of the relationship with bidirectional text and examples with RTL text have been added.
- Chapter 4, on content-oriented markup, contains major changes and additions in this Working Draft. The meaning of the actual content remains as before in principle, but a lot of work has been done on expressing it better. A few new elements have been added.
- Chapter 5 is being refined as its purpose has been further clarified to deal with the mixing of markup languages. This chapter deals, in particular, with interrelations of parts of the MathML specification, especially with presentation and content markup.
- Chapter 6 is a new addition which deals with the issues of interaction of MathML with a host environment. This chapter deals with interrelations of the MathML specification with XML and HTML, but in the context of deployment on the Web. In particular there is a discussion of the interaction of CSS with MathML.
- Chapter 7 replaces the previous Chapter 6, and has been rewritten and reorganized to reflect the new situation in regard to Unicode, and the changed W3C context with regard to named character entities. The new W3C specification of Entity Definitions for Characters in XML, which incorporates those used for mathematics is becoming a public working draft [Entities]. It is expected that some new ancillary tables will be provided that reflect requests the Math WG has received.
- The Appendices, of which there are eight shown, have been reworked. Appendix A now contains the new RelaxNG schema for MathML3 as well as discussion of MathML3 DTD issues. Appendix B addresses media types associated with MathML and implicitly constitutes a request for the registration of three new ones, as is now standard for work from the W3C. Appendix C contains a new simplified and reconsidered Operator Dictionary. Appendices D, E, F, G and H contain similar non-normative material to that in the previous specification, now appropriatley updated.
- A fuller discussion of the document's evolution can be found in Appendix F. and MathML 3.0.

Contents

Chapter 1

Introduction

1.1 Mathematics and its Notation

A distinguishing feature of mathematics is the use of a complex and highly evolved system of two-dimensional symbolic notations. As J. R. Pierce has written in his book on communication theory, mathematics and its notations should not be viewed as one and the same thing [Pierce1961]. Mathematical ideas can exist independently of the notations that represent them. However, the relation between meaning and notation is subtle, and part of the power of mathematics to describe and analyze derives from its ability to represent and manipulate ideas in symbolic form. The challenge before a Mathematics Markup Language (MathML) in enabling mathematics on the World Wide Web is to capture both notation and content (that is, its meaning) in such a way that documents can utilize the highly evolved notational forms of written and printed mathematics, and the new potential for interconnectivity in electronic media.

Mathematical notations evolve constantly as people continue to innovate in ways of approaching and expressing ideas. Even the commonplace notations of arithmetic have gone through an amazing variety of styles, including many defunct ones advocated by leading mathematical figures of their day [Cajori1928]. Modern mathematical notation is the product of centuries of refinement, and the notational conventions for high-quality typesetting are quite complicated. For example, variables and letters which stand for numbers are usually typeset today in a special mathematical italic font subtly distinct from the usual text italic; this seems to have been introduced in Europe in the late 1500 CE. Spacing around symbols for operations such as +, -, $\times$ and / is slightly different from that of text, to reflect conventions about operator precedence that have evolved over centuries. Entire books have been devoted to the conventions of mathematical typesetting, from the alignment of superscripts and subscripts, to rules for choosing parenthesis sizes, and on to specialized notational practices for subfields of mathematics. The manuals describing the nuances of present-day computer typesetting and composition systems can run to hundreds of pages.

Notational conventions in mathematics, and in printed text in general, guide the eye and make printed expressions much easier to read and understand. Though we usually take them for granted, we, as modern readers, rely on a numerous conventions such as paragraphs, capital letters, font families and cases, and even the device of decimal-like numbering of sections such as we are using in this document. Such notational conventions are perhaps even more important for electronic media, where one must contend with the difficulties of on-screen reading. But appropriate standards coupled with computers enable a broadening of access to mathematics beyond the world of print.

It is remarkable how widespread the current conventions of mathematical notations have become. The general two-dimensional layout, and most of the same symbols, are used in all modern mathematical communications, whether the participants are, say, European, writing left-to-right, or Middle-Eastern, writing right-to-left. Of course, conventions for the symbols used, particularly those naming functions

and variables, may tend to favor a local language and script. The largest variation from the most common is a form used in some Arabic-speaking communities which lays out the entire mathematical notation from right-to-left, roughly in mirror image of the European tradition.

However, there is more to putting mathematics on the Web than merely finding ways of displaying traditional mathematical notation in a Web browser. The Web represents a fundamental change in the underlying metaphor for knowledge storage, a change in which *interconnection* plays a central role. It has become important to find ways of communicating mathematics which facilitate automatic processing, searching and indexing, and reuse in other mathematical applications and contexts. With this advance in communication technology, there is an opportunity to expand our ability to represent, encode, and ultimately to communicate our mathematical insights and understanding with each other. We believe that MathML as specified below is an important step in developing mathematics on the Web.

1.2 Origins and Goals

1.2.1 Design Goals of MathML

In order to meet the diverse needs of the scientific community, MathML has been designed from the beginning with the following ultimate goals in mind.

MathML should ideally:

- Encode mathematical material suitable for teaching and scientific communication at all levels.
- Encode both mathematical notation and mathematical meaning.
- Facilitate conversion to and from other mathematical formats, both presentational and semantic. Output formats should include:
  - graphical displays
  - speech synthesizers
  - input for computer algebra systems
  - other mathematics typesetting languages, such as TEX
  - plain text displays, e.g. VT100 emulators
  - iternational print media, including braille

  Recognized that conversion to and from other notational systems or media may entail loss of information in the process.
- Allow the passing of information intended for specific renderers and applications.
- Support efficient browsing of lengthy expressions.
- Provide for extensibility.
- Be well suited to templates and other common techniques for editing formulas.
- Be legible to humans, and simple for software to generate and process.

No matter how successfully MathML achieves its goals as a markup language, it is clear that MathML is only useful if it is implemented well. The W3C Math Working Group identified long ago a short list of additional implementation goals. These goals attempt to describe concisely the minimal functionality MathML rendering and processing software should try to provide.

- MathML expressions in HTML (and XHTML) pages should render properly in popular Web browsers, in accordance with reader and author viewing preferences, and at the highest quality possible given the capabilities of the platform.
- HTML (and XHTML) documents containing MathML expressions should print properly and at high-quality printer resolutions.
- MathML expressions in Web pages should be able to react to user gestures, such those as with a mouse, and to coordinate communication with other applications through the browser.

- Mathematical expression editors and converters should be developed to facilitate the creation of Web pages containing MathML expressions.

The extent to which these goals are ultimately met depends on the cooperation and support of browser vendors, and other software developers. The W3C Math Working Group has continued to work with other working groups of the W3C, and outside the W3C, to ensure that the needs of the scientific community will be met in the future. MathML 2 and its implementations showed considerable progress in this area over the situation that obtained at the time of the MathML 1.0 Recommendation (April 1998) [MathML1]. MathML3 and the developing Web are expected to allow much more.

1.3 Overview

MathML is designed as an 'XML Application', that is, it uses XML markup for describing mathematics. A special aspect of MathML is that there are two main strains of markup: Presentation markup, discussed in Chapter 3, is used to display mathematical expressions; and Content markup, discussed in Chapter 4, is used to convey the mathematical meaning. Content markup is specified in particular detail. This specification makes use of a format called Content Dictionaries, which is also an application of XML. This format has been developed by the OpenMath Society, [OpenMath2004] with the dictionaries being used by this specification involving joint development by the OpenMath Society and the W3C Math Working Group.

Fundamentals common to both strains of markup are covered in Chapter 2, while the means for combing these strains, as well as external markup, into single MathML objects are discussed in Chapter 5. How MathML interacts with applications is covered in Chapter 6. Finally, a discussion of the special symbols, and issues regarding characters, entities and fonts, is given in Chapter 7.

1.4 A First Example

As a simple but instructive illustration of what the markup of MathML has become let us consider the quadratic formula.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

MathML offers two flavors of markup of this formula. The first is the style which emphasizes the actual presentation of a formula, the two-dimensional layout in which the symbols are arranged. So for this case we have the following.

```
<mrow>
  <mi>x</mi>
  <mo>=</mo>
  <mfrac>
    <mrow>
      <mrow>
        <mo>-</mo>
        <mi>b</mi>
      </mrow>
      <mo>&PlusMinus;</mo>
      <msqrt>
```

```
        <mrow>
          <msup>
            <mi>b</mi>
            <mn>2</mn>
          </msup>
          <mo>-</mo>
          <mrow>
            <mn>4</mn>
            <mo>&InvisibleTimes;</mo>
            <mi>a</mi>
            <mo>&InvisibleTimes;</mo>
            <mi>c</mi>
          </mrow>
        </mrow>
      </msqrt>
    </mrow>
    <mrow>
      <mn>2</mn>
      <mo>&InvisibleTimes;</mo>
      <mi>a</mi>
    </mrow>
  </mfrac>
</mrow>
```

Consider the superscript 2 in this formula. It is a commonplace that this represents the squaring operation here, but the meaning of a superscript actually depends on the context. A letter with a superscript can be used to signify a particular component of a vector, or maybe the superscript just labels a different type of some structure. Similarly two letters written one just after the other could signify two variables multiplied together, as they do in the quadratic formula, or they could be two letters making up the name of a single variable. What is called Content Markup in MathML allows closer specification of the mathematical meaning of many common formulas. The quadratic formula given in this style of markup is as follows.

```
<apply><eq/>
  <apply><plus/>
    <apply><power/>
      <ci>x</ci>
      <cn>2</cn>
    </apply>
    <apply><times/>
      <cn>4</cn>
      <ci>x</ci>
    </apply>
    <cn>4</cn>
  </apply>
  <cn>0</cn>
</apply>
```

Chapter 2

MathML Fundamentals

2.1 MathML Syntax and Grammar

2.1.1 General Considerations

MathML is an application of [XML], Extensible Markup Language, and as such it is governed by the rules of XML syntax. XML syntax is a notation for rooted labeled planar trees. Planarity means that the children of a node may be viewed as given a natural order and MathML depends on this.

The basic 'syntax' of MathML is thus defined by XML. Upon this, we layer a 'grammar', being the rules for allowed elements, the order in which they can appear, and how they may be contained within each other, as well as additional syntactic rules for the values of attributes. These rules are defined by this specification, and formalized by a RelaxNG schema [RELAX-NG]. The RelaxNG Schema is normative, but a DTD (Document Type Definition) and an XML Schema [XMLSchemas] are provided for continuity (they were normative for MathML2). See Appendix A.

As an XML vocabulary, MathML's character set must consist of legal characters as specified by Unicode [Unicode]. The use of Unicode characters for mathematics is discussed in Chapter 7.

The following sections discuss the general aspects of the MathML grammar as well as describe the syntaxes used for attribute values.

2.1.2 MathML and Namespaces

An XML namespace [Namespaces] is a collection of names identified by a URI. The URI for the MathML namespace is:

`http://www.w3.org/1998/Math/MathML`

To declare a namespace, one uses an `xmlns` attribute, or an attribute with an `xmlns` prefix. When the `xmlns` attribute is used alone, it sets the default namespace for the element on which it appears, and for any child elements. For example:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
<mrow>...</mrow>
</math>
```

When the `xmlns` attribute is used as a prefix, it declares a prefix which can then be used to explicitly associate other elements and attributes with a particular namespace. When embedding MathML within XHTML, one might use:

```
<body xmlns:m="http://www.w3.org/1998/Math/MathML">
...
<m:math><m:mrow>...</m:mrow></m:math>
...
</body>
```

2.1.3 Children versus Arguments

Many MathML elements require a specific number of children or attach a particular meaning to child elements in certain positions. When children of a given MathML element are subject to these conditions, we will often refer to them as *arguments* instead of merely as children, in order to emphasize this somewhat mathematical relationship. For elements that act as 'containers', the arguments correspond directly to children. This is the case for most presentation elements and some content elements such as `set`. In other cases, such as the content element `apply`, it is clearer to refer to the second child of the `apply` as being the 'first argument' of the operator; that operator itself being the first child of the `apply`. Other cases are presentation elements that conceptually accept only a single argument, but for convenience accept any number of children; then we infer an `mrow` containing those children which acts as the argument to the element in question; See Section 3.1.3.1.

In the detailed discussions of element syntax given with each element throughout the MathML specification, the correspondence of children with arguments, the number of arguments required and their order, as well as other constraints on the content are given. This information is also tabulated for the presentation elements Section 3.1.3.

2.1.4 MathML and Rendering

MathML presentation elements only suggest (i.e., do not require) specific ways of rendering in order to allow for medium-dependent rendering and for individual preferences of style.

Nevertheless, some parts of this specification describe suggested visual rendering rules in some detail; in those descriptions it is often assumed that the model of rendering used supports the concepts of a well-defined 'current rendering environment' which, in particular, specifies a 'current font', a 'current display' (for pixel size) and a 'current baseline'. The 'current font' provides certain metric properties and an encoding of glyphs.

2.1.5 MathML Attribute Values

MathML elements take attributes with values that further specialize the meaning or effect of the element. Attribute names are shown in a `monospaced` font. The meaning of each attribute and its allowable values are described, throughout this document, along with the specification of each element. The syntax for allowable values use the syntax explained in this section.

When otherwise allowed by the specification for each attribute, MathML attribute values may contain any legal characters specified by the XML recommendation. See Chapter 7 for further clarification.

2.1.5.1 Syntax notations used in the MathML specification

To describe the MathML-specific syntax of permissible attribute values, the following conventions and notations are used for most attributes in the present document.

Notation	What it matches
decimal-digit	a decimal digit from the range U+0030 to U+0039
hexadecimal-digit	a hexadecimal (base 16) digit from the range U+0030 to U+0039 or U+0041 to U+0046
unsigned-integer	a string of *decimal-digit*s, representing a non-negative integer.
positive-integer	a string of *decimal-digit*s, but not consisting solely of "0"s (U+0030), representing a positive integer.
integer	a string of *decimal digit*s, optionally starting with '-' (U+002D)
unsigned-number	a decimal integer or rational number (a string of *digit*s, with up to one decimal point represented by U+002E), no sign is allowed.
number	a decimal integer or rational number, optionally starting with '-' (U+002D)
character	a single non-whitespace character
string	an arbitrary character string
length	a length, as explained below, Section 2.1.5.2
color	a color, as explained below, Section 2.1.5.3
id	an identifier, unique within the document; must satisfy the NAME syntax of the XML recommendation [XML]
idref	an identifier referring to another element within the document; must satisfy the NAME syntax of the XML recommendation [XML]
URI	a Uniform Resource Identifier, [RFC3986]
italicized word	values as explained in the text for each attribute
literal	non-italicized words should appear literally in the attribute value
quoted symbol	that same symbol, literally present in the attribute value (e.g. "+" or '+')

The 'types' described above, except for *string*, may be combined into composite patterns using the following operators. They are shown in order of precedence from highest to lowest precedence:

Notation	What it matches
(*form*)	same as *form*
[*form*]	an optional instance of *form*
form *	zero or more instances of *form*
form +	one or more instances of *form*
f_1 f_2 ... f_n	one instance of each form f_i, in sequence, perhaps separated by whitespace
f_1 | f_2 | ... | f_n	any one of the specified forms f_i

When an attribute value is composed of multiple instances of the above types (eg. (*length* *)), adjacent values must be separated by whitespace (see Section 2.1.7); whitespace is typically not allowed within the values of types, (e.g., between a - and number). This separating whitespace is, however, optional in the case of (*character* *).

Since some applications are inconsistent about normalization of whitespace, for maximum interoperability it is advisable to use only a single whitespace character for separating parts of a value. Moreover, leading and trailing whitespace in attribute values should be avoided.

For most numerical attributes, only those in a subset of the expressible values are sensible; values outside this subset are not errors, unless otherwise specified, but rather are rounded up or down (at the discretion of the renderer) to the closest value within the allowed subset. The set of allowed values may depend on the renderer, and is not specified by MathML.

If a numerical value within an attribute value syntax description is declared to allow a minus sign ('-'), e.g., `number` or `integer`, it is not a syntax error when one is provided in cases where a negative value is not sensible. Instead, the value should be handled by the processing application as described in the preceding paragraph. An explicit plus sign ('+') is not allowed as part of a numerical value except when

it is specifically listed in the syntax (as a quoted '+' or "+"), and its presence can change the meaning of the attribute value (as documented with each attribute which permits it).

Editor's note:P. IonThe presence or not of an explicit + in attribute values is a place we should be in accord with HTML's conventions, in particular HTML5's, if at all possible.

2.1.5.2 Length Valued Attributes

Most presentation elements have attributes that accept values representing lengths to be used for size, spacing or similar properties. The syntax of a length is specified as

Type	Syntax
length	*number* | *number unit* | *namedspace*

There is no space between the number and unit.

The possible units and *namedspaces*, along with their interpretations, are shown below. Note that although the units and their meanings are taken from CSS, the syntax of lengths is not identical. A few MathML elements have length attributes that accept additional keywords; these are specified in the description of those specific elements.

When a length is given as a number without a unit it represents a multiple of the default value. Similarly, a trailing "%" represents a percent of the default value. The default value, or how it is obtained, is listed in the table of attributes for each element. (See also Section 2.1.5.4)

In some cases, the range of acceptable values for a particular attribute may be restricted; implementations are free to round up or down to the closest allowable value.

The possible units in MathML are:

Unit	Description
`em`	an em (font-relative unit traditionally used for horizontal lengths)
`ex`	an ex (font-relative unit traditionally used for vertical lengths)
`px`	pixels, or size of a pixel in the current display
`in`	inches (1 inch = 2.54 centimeters)
`cm`	centimeters
`mm`	millimeters
`pt`	points (1 point = 1/72 inch)
`pc`	picas (1 pica = 12 points)
`%`	percentage of the default value

Some additional aspects of units are discussed further under Additional Notes, below.

The following constants, *namedspaces*, may also be used where a length is needed; they are typically used for spacing or padding between tokens: `"veryverythinmathspace"` (1/18em), `"verythinmathspace"` (2/18em), `"thinmathspace"` (3/18em), `"mediummathspace"` (4/18em), `"thickmathspace"` (5/18em), `"verythickmathspace"` (6/18em), `"veryverythickmathspace"` (7/18em), as well as the negatives `"negativeveryverythinmathspace"`, `"negativeverythinmathspace"`, `"negativethinmathspace"`, `"negativemediummathspace"`, `"negativethickmathspace"`, `"negativeverythickmathspace"` and `"negativeveryverythickmathspace"`. Suggested default values for these constants are shown above in parentheses; the actual spacing used is implementation specific.

Additional notes about units

Lengths are only used in MathML for presentation, and presentation will ultimately involve rendering in or on some medium. For visual media, the display context is assumed to have certain properties

available to the rendering agent. A px corresponds to a pixel on the display, to the extent that that is meaningful. The resolution of the display device will affect the correspondence of pixels to the units in, cm, mm, pt and pc.

Moreover, the display context will also provide a default for the font size; the parameters of this font determine the initial values used to interpret the units em and ex, and thus indirectly the sizes of namedspaces. Since these units track the display context, and in particular, the user's preferences for display, the relative units em and ex are generally to be preferred over absolute units such as px or cm.

Two additional aspects of relative units must be clarified, however. First, some elements such as Section 3.4 or mfrac, implicitly switch to smaller font sizes for some of their arguments. Similarly, mstyle can be used to explicitly change the current font size. In such cases, the effective values of an em or ex inside those contexts will be different than outside. The second point is that the effective value of an em or ex used for an attribute value can be affected by changes to the current font size. Thus, attributes that affect the current font size, such as mathsize, mathvariant and scriptlevel, must be processed before evaluating other length valued attributes.

If, and how, lengths might affect non-visual media is left up to the implementors.

2.1.5.3 Color Valued Attributes

The color, or background color, of presentation elements may be specified as a *color* using the following syntax:

Type	Syntax
color	*#RGB* | *#RRGGBB* | *html-color-name*

A color is specified either by '#' followed by hexadecimal values for the red, green, and blue components, with no intervening whitespace, or by an *html-color-name*. The color components can be either 1-digit or 2-digit, but must all have the same number of digits; the component ranges from 0 (component not present) to FF (component fully present). Note that #123 corresponds to #102030.

Color values can also be specified as an *html-color-name*, one of the color-name keywords defined in [HTML4] ("aqua", "black", "blue", "fuchsia", "gray", "green", "lime", "maroon", "navy", "olive", "purple", "red", "silver", "teal", "white", and "yellow"). Note that the color name keywords are not case-sensitive, unlike most keywords in MathML attribute values, for compatibility with CSS and HTML.

When a *color* is applied to an element, it is the color in which the content of tokens is rendered. Additionally, when inherited from mstyle or from the environment in which the complete MathML expression is embedded, it controls the color of all other drawing due to MathML elements, including the lines or radical signs that can be drawn in rendering mfrac, mtable, or msqrt.

When used to specify a background color, the keyword "transparent" is also allowed. The suggested MathML visual rendering rules do not define the precise extent of the region whose background is affected by using the background attribute on mstyle, except that, when mstyle's content does not have negative dimensions and its drawing region is not overlapped by other drawing due to surrounding negative spacing, this region should lie behind all the drawing done to render the content of the mstyle, but should not lie behind any of the drawing done to render surrounding expressions. The effect of overlap of drawing regions caused by negative spacing on the extent of the region affected by the background attribute is not defined by these rules.

2.1.5.4 Default values of attributes

Default values for MathML attributes are, in general, given along with the detailed descriptions of specific elements in the text. Default values shown in plain text in the tables of attributes for an element are literal, but when italicized are descriptions of how default values can be computed.

Default values described as *inherited* are taken from the rendering environment, as described in Section 3.3.4, or in some cases (which are described individually) taken from the values of other attributes of surrounding elements, or from certain parts of those values. The value used will always be one which could have been specified explicitly, had it been known; it will never depend on the content or attributes of the same element, only on its environment. (What it means when used may, however, depend on those attributes or the content.)

Default values described as *automatic* should be computed by a MathML renderer in a way which will produce a high-quality rendering; how to do this is not usually specified by the MathML specification. The value computed will always be one which could have been specified explicitly, had it been known, but it will usually depend on the element content and possibly on the context in which the element is rendered.

Other italicized descriptions of default values which appear in the tables of attributes are explained individually for each attribute.

The single or double quotes which are required around attribute values in an XML start tag are not shown in the tables of attribute value syntax for each element, but are shown around example attribute values in the text.

Note that, in general, there is no value which can be given explicitly for a MathML attribute which will simulate the effect of not specifying the attribute at all for attributes which are *inherited* or *automatic*. Giving the words ‘inherited’ or ‘automatic’ explicitly will not work, and is not generally allowed. Furthermore, even for presentation attributes for which a specific default value is documented here, the `mstyle` element (Section 3.3.4) can be used to change this for the elements it contains.

Note also that the defaults being discussed describe the behavior of MathML applications when an attribute is not supplied; they do not indicate a value that will be filled in by the XML parser, as is sometimes done by DTD-based specifications.

2.1.6 Attributes Shared by all MathML Elements

In addition to the attributes described specifically for each element, the following attributes are also allowed on *all* MathML elements.

Name	values	default
id	*id*	*none*
Establishes an unique identifier associated with the element to support linking, cross-references and parallel markup. See `xref` and Section 5.4.		
xref	*idref*	*none*
References another element within the document. See `id` and Section 5.4.		
class	*string*	*none*
Associates the element with a set of style classes for use with [XSLT] and [CSS2]. Typically this would be a space separated sequence of words, but this is not specified by MathML. See Section 6.5 for discussion of the interaction of MathML and CSS.		
style	*string*	*none*
Associates style information with the element for use with [XSLT] and [CSS2]. This typically would be an inline CSS style, but this is not specified by MathML. See Section 6.5 for discussion of the interaction of MathML and CSS.		
href	*URI*	*none*
Can be used to establish the element as a hyperlink to the specfied *URI*.		

Note that MathML 2 had no direct support for linking, and instead followed the W3C Recommendation 'XML Linking Language' [XLink] in defining links using the `xlink:href` attribute. This has changed, and MathML 3 now uses an `href` attribute. However, particular compound document formats may specify the use of XML Linking with MathML elements, so user agents that support XML Linking should continue to support the use of the `xlink:href` attribute with MathML 3 as well.

Every MathML element, because of a legacy from MathML 1.0, also accepts the deprecated attribute `other` (Section 2.3.3) which was conceived for passing non-standard attributes without violating the MathML DTD. MathML renderers are only required to process this attribute if they respond to any attributes which are not standard in MathML. However, the use of `other` is strongly discouraged when there are already alternate ways within MathML of passing specific information.

See also Section 3.2.2 for a list of MathML attributes which can be used on most presentation token elements.

2.1.7 Collapsing Whitespace in Input

In MathML, as in XML, 'whitespace' means simple spaces, tabs, newlines, or carriage returns, i.e., characters with hexadecimal Unicode codes `U+0020`, `U+0009`, `U+000A`, or `U+000D`, respectively.

MathML ignores whitespace occurring outside token elements. Non-whitespace characters are not allowed there. Whitespace occurring within the content of token elements is 'trimmed' from the ends, i.e., all whitespace at the beginning and end of the content is removed. Whitespace internal to content of MathML elements is 'collapsed' canonically, i.e., each sequence of 1 or more whitespace characters is replaced with one space character (U+0020, sometimes called a blank character).

For example, `<mo> ( </mo>` is equivalent to `<mo>(</mo>`, and

```
<mtext>
  Theorem
  1:
</mtext>
```

is equivalent to `<mtext>Theorem 1:</mtext>`.

Authors wishing to encode whitespace characters at the start or end of the content of a token, or in

sequences other than a single space, without having them ignored, must use   or other non-marking characters that are not trimmed. For example, compare

```
<mtext>
 Theorem
  1:
</mtext>
```

with

```
<mtext>
 Theorem  1:
</mtext>
```

When the first example is rendered, there is no whitespace before 'Theorem', one space between 'Theorem' and '1:', and no whitespace after '1:'. In the second example, a single space is rendered before 'Theorem', two spaces are rendered before '1:', and there is no whitespace after the '1:'.

Note that the xml:space attribute does not apply in this situation since XML processors pass whitespace in tokens to a MathML processor; it is the MathML processing rules which specify that whitespace is trimmed and collapsed.

For whitespace occurring outside the content of the token elements mi, mn, mo, ms, mtext, ci, cn and annotation, an mspace element should be used, as opposed to an mtext element containing only 'whitespace' entities.

2.2 The Top-Level math Element

MathML specifies a single top-level or root math element, which encapsulates each instance of MathML markup within a document. All other MathML content must be contained in a math element; equivalently, every valid, complete MathML expression must be contained in <math> tags. The math element must always be the outermost element in a MathML expression; it is an error for one math element to contain another. These considerations also apply when sub-expressions are passed between applications, such as for cut-and-paste operations; See Section 6.3

The math element can contain an arbitrary number of children schemata. The children schemata render by default as if they were contained in an mrow element.

2.2.1 Attributes

In addition to the attributes specified in Section 2.1.6, the math element accepts:

Name	values	default
display	block | inline	inline
specifies whether the enclosed MathML expression should be rendered as a separate vertical block (in display style) or inline, aligned with adjacent text. When `display="block"`, `displaystyle` is initialized to `"true"`, whereas `display="block"` initializes it to `"false"`; in both cases `scriptlevel` is initialized to 0. When this attribute is missing, a rendering agent is free to initialize the state as appropriate to the context. See Section 3.1.6.		
dir	ltr | rtl	ltr
specifies the overall directionality `ltr` (Left To Right) or `rtl` (Right To Left) of layout. See Section 3.1.5 for further discussion.		
maxwidth	*length*	*available width*
specifies the maximum width to be used for linebreaking. The default is the maximum width available in the surrounding environment. If that value cannot be determined, the renderer should assume an infinite rendering width.		
overflow	linebreak | scroll | elide | truncate | scale	linebreak
specifies the preferred handing in cases where an expression is too long to fit in the allowed width. See the discussion below.		
altimg	*URI*	*none*
provides a URI referring to an image to display as a fall-back for user agents that do not support embedded MathML.		
altimg-width	*length*	*width of* `altimg`
specifies the width to display `altimg`, scaling the image if necessary; See `altimg-height`.		
altimg-height	*length*	*height of* `altimg`
specifies the height to display `altimg`, scaling the image if necessary; if only one of the attributes `altimg-width` and `altimg-height` are given, the scaling should preserve the image's aspect ratio; if neither attribute is given, the image should be shown at its natural size.		
altimg-valign	*length*	0ex
specifies the vertical alignment of the image. A positive value of `valign` shifts the bottom of the image below the current baseline, while a negative value raises it above. By default, the bottom of the image aligns to the baseline.		
alttext	*string*	*none*
provides a textual alternative as a fall-back for user agents that do not support embedded MathML or images.		
cdgroup	*URI*	*none*
The URI specifies a CD group file that acts as a catalogue of CD bases for locating OpenMath content dictionaries of `csymbol`, `annotation`, and `annotation-xml` elements in this `math` element; see Section 4.2.3. When no `cdgroup` attribute is explicitly specified, the document format embedding this `math` element may provide a method for determing CD bases. Otherwise the system must determine a CD base, in the absense of specific information `http://www.openmath.org/cd` is assumed as the CD base for all `csymbol` elements `annotation`, and `annotation-xml`. This is the CD base for the collection of standard CDs maintained by the OpenMath Society.		

In cases where size negotiation is not possible or fails (for example in the case of an expression that is too long to fit in the allowed width), the `overflow` attribute is provided to suggest a processing method to the renderer. Allowed values are:

Value	Meaning
linebreak	The expression will be broken across several lines. See Section 3.1.7 for further discussion.
scroll	The window provides a viewport into the larger complete display of the mathematical expression. Horizontal or vertical scrollbars are added to the window as necessary to allow the viewport to be moved to a different position.
elide	The display is abbreviated by removing enough of it so that the remainder fits into the window. For example, a large polynomial might have the first and last terms displayed with '+ ... +' between them. Advanced renderers may provide a facility to zoom in on elided areas.
truncate	The display is abbreviated by simply truncating it at the right and bottom borders. It is recommended that some indication of truncation is made to the viewer.
scale	The fonts used to display the mathematical expression are chosen so that the full expression fits in the window. Note that this only happens if the expression is too large. In the case of a window larger than necessary, the expression is shown at its normal size within the larger window.

Issue (control): Should there be a way to specify some sort of control over how line breaks are chosen (e.g., before or after an infix operator, or if the infix operator is duplicated)?

Issue (control): Should there be a way to specify some sort of indenting style?

2.2.2 Deprecated Attributes

The following attributes of `math` are deprecated

Name	values	default
macros	*URI* *	*none*
intended to provide a way of pointing to external macro definition files. Macros are not part of the MathML specification, and much of the desired functionality can be accommodated by XSL transformations [XSLT].		
mode	display | inline	inline
specified whether the enclosed MathML expression should be rendered in a display style or an in-line style. This attribute is deprecated in favor of the `display` attribute.		

2.3 Conformance

Information is nowadays commonly generated, processed and rendered by software tools. The exponential growth of the Web is fueling the development of advanced systems for automatically searching, categorizing, and interconnecting information. In addition, there are increasing numbers of Web services, some of which offer technically based materials and activities. Thus, although MathML can be written by hand and read by humans, whether machine-aided or just with much concentration, the future of MathML is largely tied to the ability to process it with software tools.

There are many different kinds of MathML processors: editors for authoring MathML expressions, translators for converting to and from other encodings, validators for checking MathML expressions, computation engines that evaluate, manipulate or compare MathML expressions, and rendering engines that produce visual, aural or tactile representations of mathematical notation. What it means to support MathML varies widely between applications. For example, the issues that arise with a validating parser are very different from those for an equation editor.

In this section, guidelines are given for describing different types of MathML support, and for making clear the extent of MathML support in a given application. Developers, users and reviewers are

encouraged to use these guidelines in characterizing products. The intention behind these guidelines is to facilitate reuse by and interoperability of MathML applications by accurately setting out their capabilities in quantifiable terms.

The W3C Math Working Group maintains MathML Compliance Guidelines. Consult this document for future updates on conformance activities and resources.

Editor's note:P. IonThe Compliance Guidelines mentioned above is still that for MathML2 and requires updating.

2.3.1 MathML Conformance

A valid MathML expression is an XML construct determined by the MathML Relax_NG Schema together with the additional requirements given in this specification.

We shall use the phrase 'a MathML processor' to mean any application that can accept, produce, or 'roundtrip' a valid MathML expression. Perhaps the simplest example of an application that might round-trip a MathML expression might be an editor that writes a new file even though no modifications are made.

Three forms of MathML conformance are specified:

1. A MathML-input-conformant processor must accept all valid MathML expressions, and faithfully translate all MathML expressions into application-specific form allowing native application operations to be performed.
2. A MathML-output-conformant processor must generate valid MathML, faithfully representing all application-specific data.
3. A MathML-roundtrip-conformant processor must preserve MathML equivalence. Two MathML expressions are 'equivalent' if and only if both expressions have the same interpretation (as stated by the MathML Schema and specification) under any circumstances, by any MathML processor. Equivalence on an element-by-element basis is discussed elsewhere in this document.

Beyond the above definitions, the MathML specification makes no demands of individual processors. In order to guide developers, the MathML specification includes advisory material; for example, there are many suggested rendering rules throughout Chapter 3. However, in general, developers are given wide latitude in interpreting what kind of MathML implementation is meaningful for their own particular application.

To clarify the difference between conformance and interpretation of what is meaningful, consider some examples:

1. In order to be MathML-input-conformant, a validating parser needs only to accept expressions, and return 'true' for expressions that are valid MathML. In particular, it need not render or interpret the MathML expressions at all.
2. A MathML computer-algebra interface based on content markup might choose to ignore all presentation markup. Provided the interface accepts all valid MathML expressions including those containing presentation markup, it would be technically correct to characterize the application as MathML-input-conformant.
3. An equation editor might have an internal data representation that makes it easy to export some equations as MathML but not others. If the editor exports the simple equations as valid MathML, and merely displays an error message to the effect that conversion failed for the others, it is still technically MathML-output-conformant.

2.3.1.1 MathML Test Suite and Validator

As the previous examples show, to be useful, the concept of MathML conformance frequently involves a judgment about what parts of the language are meaningfully implemented, as opposed to parts that are merely processed in a technically correct way with respect to the definitions of conformance. This requires some mechanism for giving a quantitative statement about which parts of MathML are meaningfully implemented by a given application. To this end, the W3C Math Working Group has provided a test suite.

The test suite consists of a large number of MathML expressions categorized by markup category and dominant MathML element being tested. The existence of this test suite makes it possible, for example, to characterize quantitatively the hypothetical computer algebra interface mentioned above by saying that it is a MathML-input-conformant processor which meaningfully implements MathML content markup, including all of the expressions in the content markup section of the test suite.

Developers who choose not to implement parts of the MathML specification in a meaningful way are encouraged to itemize the parts they leave out by referring to specific categories in the test suite.

For MathML-output-conformant processors, information about currently available tools to validate MathML is maintained at MathML validator. Developers of MathML-output-conformant processors are encouraged to verify their output using this validator.

Customers of MathML applications who wish to verify claims as to which parts of the MathML specification are implemented by an application are encouraged to use the test suites as a part of their decision processes.

2.3.1.2 Deprecated MathML 1.x and MathML 2.x Features

MathML 2.0 contains a number of features of earlier MathML which are now deprecated. The following points define what it means for a feature to be deprecated, and clarify the relation between deprecated features and current MathML conformance.

1. In order to be MathML-output-conformant, authoring tools may not generate MathML markup containing deprecated features.
2. In order to be MathML-input-conformant, rendering and reading tools must support deprecated features if they are to be in conformance with MathML 1.x or MathML 2.x. They do not have to support deprecated features to be considered in conformance with MathML 3.0. However, all tools are encouraged to support the old forms as much as possible.
3. In order to be MathML-roundtrip-conformant, a processor need only preserve MathML equivalence on expressions containing no deprecated features.

2.3.1.3 MathML Extension Mechanisms and Conformance

MathML 2.0 defined three basic extension mechanisms: The `mglyph` element provides a way of displaying glyphs for non-Unicode characters, and glyph variants for existing Unicode characters; the `maction` element uses attributes from other namespaces to obtain implementation-specific parameters; and content markup makes use of the `definitionURL` attribute, as well as Content Dictionaries and the cd attribute, to point to external definitions of mathematical semantics.

These extension mechanisms are important because they provide a way of encoding concepts that are beyond the scope of MathML 3.0 as presently explicitly specified, which allows MathML to be used for exploring new ideas not yet susceptible to standardization. However, as new ideas take hold, they may become part of future standards. For example, an emerging character that must be represented by an `mglyph` element today may be assigned a Unicode codepoint in the future. At that time, representing

the character directly by its Unicode codepoint would be preferable. This transition into Unicode has already taken place for hundreds of characters used for mathematics.

Because the possibility of future obsolescence is inherent in the use of extension mechanisms to facilitate the discussion of new ideas, MathML can reasonably make no conformance requirements concerning the use of extension mechanisms, even when alternative standard markup is available. For example, using an `mglyph` element to represent an 'x' is permitted. However, authors and implementors are strongly encouraged to use standard markup whenever possible. Similarly, maintainers of documents employing MathML 3.0 extension mechanisms are encouraged to monitor relevant standards activity (e.g., Unicode, OpenMath, etc) and to update documents as more standardized markup becomes available.

2.3.2 Handling of Errors

If a MathML-input-conformant application receives input containing one or more elements with an illegal number or type of attributes or child schemata, it should nonetheless attempt to render all the input in an intelligible way, i.e., to render normally those parts of the input that were valid, and to render error messages (rendered as if enclosed in an `merror` element) in place of invalid expressions.

MathML-output-conformant applications such as editors and translators may choose to generate `merror` expressions to signal errors in their input. This is usually preferable to generating valid, but possibly erroneous, MathML.

2.3.3 Attributes for unspecified data

The MathML attributes described in the MathML specification are necessary for presentation and content markup. Ideally, the MathML attributes should be an open-ended list so that users can add specific attributes for specific renderers. However, this cannot be done within the confines of a single XML DTD or in a Schema. Although it can be done using extensions of the standard DTD, say, some authors will wish to use non-standard attributes to take advantage of renderer-specific capabilities while remaining strictly in conformance with the standard DTD.

To allow this, the MathML 1.0 specification [MathML1] allowed the attribute `other` on all elements, for use as a hook to pass on renderer-specific information. In particular, it was intended as a hook for passing information to audio renderers, computer algebra systems, and for pattern matching in future macro/extension mechanisms. The motivation for this approach to the problem was historical, looking to PostScript, for example, where comments are widely used to pass information that is not part of PostScript.

In the next period of evolution of MathML the development of a general XML namespace mechanism seemed to make the use of the `other` attribute obsolete. In MathML 2.0, the `other` attribute is deprecated in favor of the use of namespace prefixes to identify non-MathML attributes. The `other` attribute remains deprecated in MathML 3.0.

For example, in MathML 1.0, it was recommended that if additional information was used in a renderer-specific implementation for the `maction` element (Section 3.7.1), that information should be passed in using the `other` attribute:

```
<maction actiontype="highlight" other="color='#ff0000'"> expression </maction>
```

From MathML 2.0 onwards, a `color` attribute from another namespace would be used:

```
<body xmlns:my="http://www.example.com/MathML/extensions">
...
<maction actiontype="highlight" my:color="#ff0000"> expression </maction>
```

```
...
</body>
```

Note that the intent of allowing non-standard attributes is *not* to encourage software developers to use this as a loophole for circumventing the core conventions for MathML markup. Authors and applications should use non-standard attributes judiciously.

Chapter 3

Presentation Markup

3.1 Introduction

This chapter specifies the 'presentation' elements of MathML, which can be used to describe the layout structure of mathematical notation.

3.1.1 What Presentation Elements Represent

Presentation elements correspond to the 'constructors' of traditional mathematical notation — that is, to the basic kinds of symbols and expression-building structures out of which any particular piece of traditional mathematical notation is built. Because of the importance of traditional visual notation, the descriptions of the notational constructs the elements represent are usually given here in visual terms. However, the elements are medium-independent in the sense that they have been designed to contain enough information for good spoken renderings as well. Some attributes of these elements may make sense only for visual media, but most attributes can be treated in an analogous way in audio as well (for example, by a correspondence between time duration and horizontal extent).

MathML presentation elements only suggest (i.e. do not require) specific ways of rendering in order to allow for medium-dependent rendering and for individual preferences of style. This specification describes suggested visual rendering rules in some detail, but a particular MathML renderer is free to use its own rules as long as its renderings are intelligible.

The presentation elements are meant to express the syntactic structure of mathematical notation in much the same way as titles, sections, and paragraphs capture the higher-level syntactic structure of a textual document. Because of this, for example, a single row of identifiers and operators, such as '$x + a / b$', will often be represented not just by one `mrow` element (which renders as a horizontal row of its arguments), but by multiple nested `mrow` elements corresponding to the nested sub-expressions of which one mathematical expression is composed — in this case,

```
<mrow>
  <mi> x </mi>
  <mo> + </mo>
  <mrow>
    <mi> a </mi>
    <mo> / </mo>
    <mi> b </mi>
  </mrow>
</mrow>
```

Similarly, superscripts are attached not just to the preceding character, but to the full expression constituting their base. This structure allows for better-quality rendering of mathematics, especially when

details of the rendering environment such as display widths are not known to the document author; it also greatly eases automatic interpretation of the mathematical structures being represented.

Certain MathML characters are used to name operators or identifiers that in traditional notation render the same as other symbols, such as `&DifferentialD;`, `&ExponentialE;`, or `&ImaginaryI;`, or operators that usually render invisibly, such as `&InvisibleTimes;`, `&InvisiblePlus;`, `&ApplyFunction;`, or `&InvisibleComma;`. These are distinct notational symbols or objects, as evidenced by their distinct spoken renderings and in some cases by their effects on linebreaking and spacing in visual rendering, and as such should be represented by the appropriate specific entity references. For example, the expression represented visually as '$f(x)$' would usually be spoken in English as 'f of x' rather than just 'f x'; this is expressible in MathML by the use of the `&ApplyFunction;` operator after the 'f', which (in this case) can be aurally rendered as 'of'.

The complete list of MathML entities is described in [Entities].

3.1.2 Terminology Used In This Chapter

It is strongly recommended that, before reading the present chapter, one read Section 2.1 on MathML syntax and grammar, which contains important information on MathML notations and conventions. In particular, in this chapter it is assumed that the reader has an understanding of basic XML terminology described in Section 2.1.3, and the attribute value notations and conventions described in Section 2.1.5.

The remainder of this section introduces MathML-specific terminology and conventions used in this chapter.

3.1.2.1 Types of presentation elements

The presentation elements are divided into two classes. *Token elements* represent individual symbols, names, numbers, labels, etc. In general, tokens can have only characters as content. The only exceptions are the vertical alignment element `malignmark`, `mglyph`. *Layout schemata* build expressions out of parts, and can have only elements as content (except for whitespace, which they ignore). There are also a few empty elements used only in conjunction with certain layout schemata.

All individual 'symbols' in a mathematical expression should be represented by MathML token elements. The primary MathML token element types are identifiers (e.g. variables or function names), numbers, and operators (including fences, such as parentheses, and separators, such as commas). There are also token elements for representing text or whitespace that has more aesthetic than mathematical significance, and for representing 'string literals' for compatibility with computer algebra systems. Note that although a token element represents a single meaningful 'symbol' (name, number, label, mathematical symbol, etc.), such symbols may be comprised of more than one character. For example `sin` and `24` are represented by the single tokens `<mi>sin</mi>` and `<mn>24</mn>` respectively.

In traditional mathematical notation, expressions are recursively constructed out of smaller expressions, and ultimately out of single symbols, with the parts grouped and positioned using one of a small set of notational structures, which can be thought of as 'expression constructors'. In MathML, expressions are constructed in the same way, with the layout schemata playing the role of the expression constructors. The layout schemata specify the way in which sub-expressions are built into larger expressions. The terminology derives from the fact that each layout schema corresponds to a different way of 'laying out' its sub-expressions to form a larger expression in traditional mathematical typesetting.

3.1.2.2 Terminology for other classes of elements and their relationships

The terminology used in this chapter for special classes of elements, and for relationships between elements, is as follows: The *presentation elements* are the MathML elements defined in this chapter.

These elements are listed in Section 3.1.9. The *content elements* are the MathML elements defined in Chapter 4.

A MathML *expression* is a single instance of any of the presentation elements with the exception of the empty elements `none` or `mprescripts`, or is a single instance of any of the content elements which are allowed as content of presentation elements (described in Section 5.3.2). A *sub-expression* of an expression E is any MathML expression that is part of the content of E, whether *directly* or *indirectly*, i.e. whether it is a 'child' of E or not.

Since layout schemata attach special meaning to the number and/or positions of their children, a child of a layout schema is also called an *argument* of that element. As a consequence of the above definitions, the content of a layout schema consists exactly of a sequence of zero or more elements that are its arguments.

3.1.3 Required Arguments

Many of the elements described herein require a specific number of arguments (always 1, 2, or 3). In the detailed descriptions of element syntax given below, the number of required arguments is implicitly indicated by giving names for the arguments at various positions. A few elements have additional requirements on the number or type of arguments, which are described with the individual element. For example, some elements accept sequences of zero or more arguments — that is, they are allowed to occur with no arguments at all.

Note that MathML elements encoding rendered space *do* count as arguments of the elements in which they appear. See Section 3.2.7 for a discussion of the proper use of such space-like elements.

3.1.3.1 Inferred <mrow>s

The elements listed in the following table as requiring 1* argument (`msqrt`, `mstyle`, `merror`, `menclose`, `mpadded`, `mphantom`, `mtd`, and `math`) conceptually accept a single argument, but actually accept any number of children. If the number of children is 0, or is more than 1, they treat their contents as a single *inferred* `mrow` formed from all their children, and treat this `mrow` as the argument. Although the `math` element is not a presentation element, it is listed below for completeness.

For example,

```
<mtd>
</mtd>
```

is treated as if it were

```
<mtd>
  <mrow>
  </mrow>
</mtd>
```

and

```
<msqrt>
  <mo> - </mo>
  <mn> 1 </mn>
</msqrt>
```

is treated as if it were

```
<msqrt>
  <mrow>
```

```
    <mo> - </mo>
    <mn> 1 </mn>
  </mrow>
</msqrt>
```

This feature allows MathML data not to contain (and its authors to leave out) many `mrow` elements that would otherwise be necessary.

3.1.3.2 Table of argument requirements

For convenience, here is a table of each element's argument count requirements, and the roles of individual arguments when these are distinguished. An argument count of 1* indicates an inferred `mrow` as described above.

Element	Required argument count	Argument roles (when these differ by position)
`mrow`	0 or more	
`mfrac`	2	*numerator denominator*
`msqrt`	1*	
`mroot`	2	*base index*
`mstyle`	1*	
`merror`	1*	
`mpadded`	1*	
`mphantom`	1*	
`mfenced`	0 or more	
`menclose`	1*	
`msub`	2	*base subscript*
`msup`	2	*base superscript*
`msubsup`	3	*base subscript superscript*
`munder`	2	*base underscript*
`mover`	2	*base overscript*
`munderover`	3	*base underscript overscript*
`mmultiscripts`	1 or more	*base* (*subscript superscript*)* [`<mprescripts/>` (*presubscript presuperscript*)*]
`mtable`	0 or more rows	0 or more `mtr` or `mlabeledtr` elements
`mlabeledtr`	1 or more	a label and 0 or more `mtd` elements
`mtr`	0 or more	0 or more `mtd` elements
`mtd`	1*	
`mstack`	1 or more	
`mlongdiv`	1 or more	
`msgroup`	1 or more	
`msr`	1 or more	
`mscarry`	1*	
`maction`	1 or more	depend on `actiontype` attribute
`math`	1*	

3.1.4 Elements with Special Behaviors

Certain MathML presentation elements exhibit special behaviors in certain contexts. Such special behaviors are discussed in the detailed element descriptions below. However, for convenience, some of the most important classes of special behavior are listed here.

Certain elements are considered space-like; these are defined in Section 3.2.7. This definition affects some of the suggested rendering rules for `mo` elements (Section 3.2.5).

Certain elements, e.g. `msup`, are able to embellish operators that are their first argument. These elements are listed in Section 3.2.5, which precisely defines an 'embellished operator' and explains how this affects the suggested rendering rules for stretchy operators.

3.1.5 Directionality

In the notations familiar to most readers, both the overall layout and the textual symbols are arranged from left to right (LTR). Yet, as alluded to in the introduction, mathematics written in Hebrew, or in locales such as Morocco or Persia, the overall layout is used unchanged, but the embedded symbols (often Hebrew or Arabic) are written right to left (RTL). Moreover, in most of the Arabic speaking world, the notation is arranged entirely RTL; thus a superscript is still raised, but it follows the base on the left, rather than the right.

MathML 3.0 therefore recognizes two distinct directionalities: the directionality of the text and symbols within token elements, and the overall directionality represented by Layout Schemata. These two facets are dicussed below.

3.1.5.1 Overall Directionality of Mathematics Formulas

The overall directionality for a formula, basically the direction of the Layout Schemata, is specified by the `dir` attribute on the containing `math` element (see Section 2.2). The default is `ltr`. When `dir='rtl'` is used, the layout is simply the mirror image of the conventional European layout. That is, shifts up or down are unchanged, but the progression in laying out is from right to left. Sub- and superscripts appear to the left of the base; the surd for a root appears at the right, with the bar continuing over the base to the left.

The overall directionality may also be switched for individual subformula by using the `dir` attribute on `mrow` elements. When not specified, all `mrow` elements inherit the directionality of the container.

3.1.5.2 Bidirectional Layout in Token Elements

The text directionality comes into play for the MathML token elements that can contain text (`mtext`, `mo`, `mi`, `mn` and `ms`), and is determined by the Unicode properties of that text. A token element containing exclusively LTR or RTL characters is displayed straightforwardly in the given direction. When a mixture of directions is involved used, such as RTL Arabic and LTR numbers, the Unicode bidirectional algorithm [Bidi] is applied. This algorithm specifies how runs of characters with the same direction are processed and how the runs are (re)ordered. The base, or initial, direction is given by the overall directionality described above (Section 3.1.5.1), and affects how weakly directional characters are treated and how runs are nested.

The important thing to notice is that the Bidi algorithm is applied independently to the contents of each token element; each token element is an independent run of characters. This is in contrast to the application of Bidi to HTML, where the algorithm applies to the entire sequence of characters within each block level element.

Other features of Unicode and scripts that should be respected are 'mirroring' and 'glyph shaping'. Some Unicode characters are marked as being mirrored when presented in a RTL context, that is, the character is drawn as if it were mirrored, or replaced by a corresponding character. Thus an opening parenthesis, '(', in RTL will display as ')'. Conversely, the solidus (/ U+002F), is *not* marked as mirrored. Thus, an Arabic author that desires the slash to be reversed in an inline division should explicitly use reverse solidus (\ U+005C), or an alternative such as the mirroring DIVISION SLASH (U+2215).

Additionally, caligraphic scripts such as Arabic blend, or connect, sequences of characters together, changing their appearance. As this can have an significant impact on readability, as well as aesthetics, it is important to apply such shaping if possible. Glyph shaping, like directionality, applies to each token element's contents individually.

Issue (unicode-properties): We need to check on the status of various characters added to support Arabic, and also check that the directionality and mirroring properties are correct. (eg summation and similar)

Please note that for the transfinite cardinals represented by Hebrew characters, the codepoints U+2135-U+2138 (ALEF SYMBOL, BET SYMBOL, GIMEL SYMBOL, DALET SYMBOL) should be used. These are strong left-to-right.

3.1.6 Displaystyle and Scriptlevel

So-called 'displayed' formula, those appearing on a line by themselves, typically make more generous use of vertical space than inline formula which should blend into the adjacent text without intruding into neighboring lines. For example, in a displayed summation, the limits are placed above and below the summation symbol, while when it appears inline the limits would appear in the sub and superscript position. For similar reasons, sub- and superscripts, nested fractions and other constructs typically display in a smaller size than the main part of the formula. MathML implicitly associates with every presentation node a `displaystyle` and `scriptlevel` reflecting whether a more expansive vertical layout applies and the level of scripting in the current context.

These values are initialized by the `math` element according to the `display` attribute. They are automatically adjusted by the various script and limit schemata elements, and the elements `mfrac`, and `mroot`, which typically set `displaystyle` false and increment `scriptlevel` for some or all of their arguments. (See the description for each element for the specific rules used.) They also may be set explicitly via the `displaystyle` and `scriptlevel` attributes on the `mstyle` element, or the `displaystyle` attribute of `mtable`. In all other cases, they are inherited from the node's parent.

The `displaystyle` affects the amount of vertical space used to lay out a formula: when true, the more spacious layout of displayed equations is used, whereas when false a more compact layout of inline formula is used. This primarily affects the interpretation of the `largeop` and `movablelimits` attributes of the `mo` element. However, more sophisticated renderers are free to use this attribute to render more or less compactly.

The main effect of `scriptlevel` is to control the font size. Typically, the higher the `scriptlevel`, the smaller the font size. (Non-visual renderers can respond to the font size in an analogous way for their medium.) Whenever the `scriptlevel` is changed, whether automatically or explicitly, the current font size is multiplied by the value of `scriptsizemultiplier` to the power of the *change* in `scriptlevel`. However, changes to the font size due to `scriptlevel` changes should never reduce the size below `scriptminsize`, to prevent scripts becoming unreadably small. The default `scriptsizemultiplier` is approximately the square root of 1/2, whereas `scriptminsize` defaults to 8 points; these values may be changed on `mstyle`; see Section 3.3.4. Note that the `scriptlevel` attribute of `mstyle` allows arbitrary values of `scriptlevel` to be obtained, including negative values which result in increased font sizes.

The changes to the font size due to `scriptlevel` should be viewed as being imposed from 'outside' the node. This means that the effect of `scriptlevel` is applied before an explicit `mathsize` (See Section 3.2.2) on a token child of `mfrac`. Thus, the `mathsize` effectively overrides the effect of `scriptlevel`. However, that change to `scriptlevel` changes the current font size, which affects the meaning of an `"em"` length (See Section 2.1.5.2), and so the `scriptlevel` still may have an effect in

such cases. Note also that since mathsize is not constrained by scriptminsize, such direct changes to font size can result in scripts smaller than scriptminsize.

Note that direct changes to current font size, whether by CSS or by the mathsize attribute (See Section 3.2.2), have no effect on the value of scriptlevel.

TEX's \displaystyle, \textstyle, \scriptstyle, and \scriptscriptstyle correspond to displaystyle and scriptlevel as "true" and "0", "false" and "0", "false" and "1", and "false" and "2", respectively. Thus, math's display="block" correponds to \displaystyle, while display="inline" correponds to \textstyle.

3.1.7 Linebreaking of Expressions

3.1.7.1 Control of Linebreaks

MathML provides support for both automatic and manual (forced) linebreaking of expressions, to break excessively long expressions into several lines. All such linebreaks take place within mrow (including inferred mrow; See Section 3.1.3.1), or mfenced. The breaks themselves take place at operators (mo), and also, for backwards compatibility, at mspace.

Automatic linebreaking occurs when the containing math element has overflow="linebreak" and the display engine determines that there is not enough space available to display the entire formula. The available width must therefore be known to the renderer. Like font properties, one is assumed to be inherited from the environment in which the MathML element lives. If no width can be determined, an infinite width should be assumed. Inside of a mtable, each column has some width. This width may be specified as an attribute or determined by the contents. This width should be used as the linewrapping width for linebreaking, and each entry in an mtable is linewrapped as needed.

Forced linebreaks are specified by using linebreak="newline" on a mo or mspace element. Both automatic and manual linebreaking can occur within the same formula.

Automatic linebreaking of subexpressions of mfrac, msqrt, mroot and menclose and the various script elements is not required. Renderers are free to ignore forced breaks within those elements if they choose.

Attributes on mo and possibily on mspace elements control linebreaking and indentation of the following line. The aspects of linebreaking that can be controlled are:

- *Where* — attributes determine the desirability of a linebreak at a specific operator or space, in particular whether a break is required or inhibited. These can only be set on mo and mspace elements. (See Section 3.2.5.2)
- *Operator Display/Position* — when a linebreak occurs, determines whether the operator will appear at the end of the line, at the beginning of the next line, or in both positions; and how much vertical space should be added after the linebreak. These attributes can be set on mo elements or inherited from mstyle or math elements. (See Section 3.2.5.2)
- *Indentation* — determines the indentation of the line following a linebreak, including indenting so that the next line aligns with some point in a previous line. These attributes can be set on mo and mspace elements or inherited from mstyle or math elements. (See Section 3.2.5.2)

3.1.7.2 Automatic Linebreaking Algorithm (Informative)

One method of linebreaking that works reasonably well is sometimes referred to as a "best-fit" algorithm. It works by computing a "penalty" for each potential break point on a line. The break point with

the smallest penalty is chosen and the algorithm then works on the next line. Three useful factors in a penalty calculation are:

1. How much of the line width (after subtracting of the indent) is unused? The more unused, the higher the penalty.
2. How deeply nested is the breakpoint in the expression tree? The expression tree's depth is roughly similar to the nesting depth of `mrows`. The more deeply nested the break point, the higher the penalty.
3. If the next line is not the last line, and if the indentingstyle uses information about the linebreak point to determine how much to indent, then the amount of room left for linebreaking on the next line (ie, linebreaks that leave very little room to draw the next line result in a higher penalty).
4. Whether `"linebreak"` has been specified: `"nobreak"` effectively sets the penalty to infinity, `"badbreak"` increases the penalty, `"goodbreak"` decreases the penalty, and `"newline"` effectively sets the penalty to 0.

This algorithm takes time proportional to the number of tokens elements times the number of lines.

3.1.8 Warning about fine-tuning of presentation

Several elements and attributes of MathML are expressly designed to support fine-tuning of presentation for use-cases that wish to exert precise control of the layout and presentation of math. However, given the variability in MathML agents, the variability of the fonts available on different platforms, and particularly given the freedom given to agents to layout the mathematics according to their own requirements (See Section 3.1), it must be pointed out that such fine-tuning can often lead to a lack of portability. Specifically, the overuse of these controls may yeild a 'perfect' layout on one platform, but give much worse presentation on others. The following sections clarify the kinds of problems that can occur.

3.1.8.1 Warning: nonportability of 'tweaking'

A likely temptation for the use of the `mpadded` and `mspace` elements (and perhaps also `mphantom` and `mtext`) will be for an author to improve the spacing generated by a specific renderer by slightly modifying it in specific expressions, i.e. to 'tweak' the rendering.

Authors are strongly warned that *different MathML renderers may use different spacing rules* for computing the relative positions of rendered symbols in expressions that have no explicit modifications to their spacing; if renderer B improves upon renderer A's spacing rules, explicit spacing added to improve the output quality of renderer A may produce very poor results in renderer B, very likely worse than without any 'tweaking' at all.

Even when a specific choice of renderer can be assumed, its spacing rules may be improved in successive versions, so that the effect of tweaking in a given MathML document may grow worse with time. Also, when style sheet mechanisms are extended to MathML, even one version of a renderer may use different spacing rules for users with different style sheets.

Therefore, it is suggested that MathML markup never use `mpadded` or `mspace` elements to tweak the rendering of specific expressions, unless the MathML is generated solely to be viewed using one specific version of one MathML renderer, using one specific style sheet (if style sheets are available in that renderer).

In cases where the temptation to improve spacing proves too strong, careful use of `mpadded`, `mphantom`, or the alignment elements (Section 3.5.5) may give more portable results than the direct insertion of

extra space using `mspace` or `mtext`. Advice given to the implementors of MathML renderers might be still more productive, in the long run.

3.1.8.2 *Warning: spacing should not be used to convey meaning*

MathML elements that permit 'negative spacing', namely `mspace`, `mpadded`, and `mtext`, could in theory be used to simulate new notations or 'overstruck' characters by the visual overlap of the renderings of more than one MathML sub-expression.

This practice is *strongly discouraged in all situations*, for the following reasons:

- it will give different results in different MathML renderers (so the warning about 'tweaking' applies), especially if attempts are made to render glyphs outside the bounding box of the MathML expression;
- it is likely to appear much worse than a more standard construct supported by good renderers;
- such expressions are almost certain to be uninterpretable by audio renderers, computer algebra systems, text searches for standard symbols, or other processors of MathML input.

More generally, any construct that uses spacing to convey mathematical meaning, rather than simply as an aid to viewing expression structure, is discouraged. That is, the constructs that are discouraged are those that would be interpreted differently by a human viewer of rendered MathML if all explicit spacing was removed.

Consider using the `mglyph` element for cases such as this. If such spacing constructs are used in spite of this warning, they should be enclosed in a `semantics` element that also provides an additional MathML expression that can be interpreted in a standard way. See Section 5.1 for further discussion.

The above warning also applies to most uses of rendering attributes to alter the meaning conveyed by an expression, with the exception of attributes on `mi` (such as `mathvariant`) used to distinguish one variable from another.

3.1.9 Summary of Presentation Elements

3.1.9.1 *Token Elements*

`mi`	identifier
`mn`	number
`mo`	operator, fence, or separator
`mtext`	text
`mspace`	space
`ms`	string literal
`mglyph`	accessing glyphs for characters from MathML
`msline`	horizontal line inside of `mstack`

3.1.9.2 General Layout Schemata

`mrow`	group any number of sub-expressions horizontally
`mfrac`	form a fraction from two sub-expressions
`msqrt`	form a square root (radical without an index)
`mroot`	form a radical with specified index
`mstyle`	style change
`merror`	enclose a syntax error message from a preprocessor
`mpadded`	adjust space around content
`mphantom`	make content invisible but preserve its size
`mfenced`	surround content with a pair of fences
`menclose`	enclose content with a stretching symbol such as a long division sign.

3.1.9.3 Script and Limit Schemata

`msub`	attach a subscript to a base
`msup`	attach a superscript to a base
`msubsup`	attach a subscript-superscript pair to a base
`munder`	attach an underscript to a base
`mover`	attach an overscript to a base
`munderover`	attach an underscript-overscript pair to a base
`mmultiscripts`	attach prescripts and tensor indices to a base

3.1.9.4 Tables and Matrices

`mtable`	table or matrix
`mlabeledtr`	row in a table or matrix with a label or equation number
`mtr`	row in a table or matrix
`mtd`	one entry in a table or matrix
`maligngroup` and `malignmark`	alignment markers

3.1.9.5 Elementary Math Layout

`mstack`	columns of aligned characters
`mlongdiv`	similar to msgroup, with the addition of a divisor and result
`msgroup`	a group of rows in an mstack that are shifted by similar amounts
`msrow`	a row in an mstack
`mscarries`	row in an mstack that whose contents represent carries or borrows
`mscarry`	one entry in an mscarries

3.1.9.6 Enlivening Expressions

`maction`	bind actions to a sub-expression

3.2 Token Elements

Token elements in presentation markup are broadly intended to represent the smallest units of mathematical notation which carry meaning. Tokens are roughly analogous to words in text. However, because of the precise, symbolic nature of mathematical notation, the various categories and properties of

token elements figure prominently in MathML markup. By contrast, in textual data, individual words rarely need to be marked up or styled specially.

Frequently tokens consist of a single character denoting a mathematical symbol. Other cases, e.g. function names, involve multi-character tokens. Further, because traditional mathematical notation makes wide use of symbols distinguished by their typographical properties (e.g. a Fraktur 'g' for a Lie algebra, or a bold 'x' for a vector), care must be taken to insure that styling mechanisms respect typographical properties which carry meaning. Consequently, characters, tokens, and typographical properties of symbols are closely related to one another in MathML.

3.2.1 MathML characters in token elements

Character data in MathML markup is only allowed to occur as part of the content of token elements. The only exception is whitespace between elements, which is ignored. Token elements can contain any sequence of zero or more Unicode characters. In particular, tokens with empty content are allowed, and should typically render invisibly, with no width except for the normal extra spacing for that kind of token element. The exceptions to this are the empty elements `mspace`, `mglyph` and `msline`. The width of these elemnts depend upon their attribute values.

MathML characters can be either represented directly as Unicode character data, or indirectly via numeric or character entity references. See Chapter 7 for a discussion of the advantages and disadvantages of numeric character references versus entity references, and [Entities] for a full list of the entity names available.

New mathematical "characters" that arise, or non-standard glyphs for existing MathML characters, may be represented by means of the `mglyph` element.

Apart from the `mglyph` element, the `malignmark` element is the only other element allowed in the content of tokens. See Section 3.5.5 for details.

Token elements (other than `mspace`, `mglyph` and `msline`) should be rendered as their content (i.e. in the visual case, as a closely-spaced horizontal row of standard glyphs for the characters in their content). Rendering algorithms should also take into account the mathematics style attributes as described below, and modify surrounding spacing by rules or attributes specific to each type of token element.

3.2.1.1 Alphanumeric symbol characters

A large class of mathematical symbols are single letter identifiers typically used as variable names in formulas. Different font variants of a letter are treated as separate symbols. For example, a Fraktur 'g' might denote a Lie algebra, while a Roman 'g' denotes the corresponding Lie group. These letter-like symbols are traditionally typeset differently than the same characters appearing in text, using different spacing and ligature conventions. These characters must also be treated specially by style mechanisms, since arbitrary style transformations can change meaning in an expression.

For these reasons, Unicode contains more than nine hundred Math Alphanumeric Symbol characters corresponding to letter-like symbols. These characters are in the Secondary Multilingual Plane (SMP). See [Entities] for more information. As valid Unicode data, these characters are permitted in MathML, and as tools and fonts for them become widely available, we anticipate they will be the predominant way of denoting letter-like symbols.

MathML also provides an alternative encoding for these characters using only Basic Multilingual Plane (BMP) characters together with markup. MathML defines a correspondence between token elements with certain combinations of BMP character data and the `mathvariant` attribute and tokens containing

SMP Math Alphanumeric Symbol characters. Processing applications that accept SMP characters are required to treat the corresponding BMP and attribute combinations identically. This is particularly important for applications that support searching and/or equality testing.

The next section discusses the `mathvariant` attribute in more detail, and a complete technical description of the corresponding characters is given in Section 7.5.

3.2.2 Mathematics style attributes common to token elements

MathML includes four *mathematics style* attributes. These attributes are valid on all presentation token elements, and on no other elements except `mstyle`. The attributes are:

Name	values	default
mathvariant	normal | bold | italic | bold-italic | double-struck | bold-fraktur | script | bold-script | fraktur | sans-serif | bold-sans-serif | sans-serif-italic | sans-serif-bold-italic | monospace | initial | tailed | looped | stretched	normal (*except on* `<mi>`)
Specifies the logical class of the token. Note that this class is more than styling, it typically conveys semantic intent; see the discussion below.		
mathsize	small | normal | big | *length*	*inherited*
Specifies the size to display the token content. The values `"small"` and `"big"` choose a size smaller or larger than the current font size, but leave the exact proportions unspecified; `"normal"` is allowed for completeness, but since it is equivalent to `"100%"` or `"1em"`, it has no effect.		
mathcolor	*color*	*inherited*
Specifies the color to display the token content.		
mathbackground	*color* | transparent	transparent
Specifies the color for the background behind the display of the token content.		

The mathematics style attributes define logical classes of token elements. Each class is intended to correspond to a collection of typographically-related symbolic tokens that have a meaning within a given math expression, and therefore need to be visually distinguished and protected from inadvertent document-wide style changes which might change their meanings.

When MathML rendering takes place in an environment where CSS is available, the mathematics style attributes can be viewed as predefined selectors for CSS style rules. See Section 6.5 for discussion of the interaction of MathML and CSS. Also, see [MathMLforCSS] for discussion of rendering MathML by CSS and a sample CSS style sheet. When CSS is not available, it is up to the internal style mechanism of the rendering application to visually distinguish the different logical classes. Most MathML renderers will probably want to rely on some degree to additional, internal style processing algorithms. In particular, the `mathvariant` attribute does not follow the CSS inheritance model; the default value is `"normal"` (non-slanted) for all tokens except for `mi` with single-character content. See Section 3.2.3 for details.

Renderers have complete freedom in mapping mathematics style attributes to specific rendering properties. However, in practice, the mathematics style attribute names and values suggest obvious typographical properties, and renderers should attempt to respect these natural interpretations as far as possible. For example, it is reasonable to render a token with the `mathvariant` attribute set to `"sans-serif"` in Helvetica or Arial. However, rendering the token in a Times Roman font could be seriously misleading and should be avoided.

It is important to note that only certain combinations of character data and `mathvariant` attribute values make sense. For example, there is no clear cut rendering for a 'fraktur' alpha, or a 'bold italic'

Kanji character. By design, the only cases that have an unambiguous interpretation are exactly the ones that correspond to SMP Math Alphanumeric Symbol characters, which are enumerated in Section 7.5. The `mathvariant` values `"initial"`, `"tailed"`, `"looped"` and `"stretched"` are expected to apply only to Arabic characters. In all other cases, it is suggested that renderers ignore the value of the `mathvariant` attribute if it is present. Similarly, authors should refrain from using the `mathvariant` attribute with characters that do not have SMP counterparts, since renderings may not be useful or predictable. In the very rare case that it is necessary to specify a font variant for other characters or symbols within an equation, external styling mechanisms such as CSS are generally preferable, but see Section 6.5 for caveats.

Token elements also accept the attributes listed in Section 2.1.6.

Since MathML expressions are often embedded in a textual data format such as XHTML, the surrounding text and the MathML must share rendering attributes such as font size, so that the renderings will be compatible in style. For this reason, most attribute values affecting text rendering are inherited from the rendering environment, as shown in the 'default' column in the table above. (In cases where the surrounding text and the MathML are being rendered by separate software, e.g. a browser and a plug-in, it is also important for the rendering environment to provide the MathML renderer with additional information, such as the baseline position of surrounding text, which is not specified by any MathML attributes.) Note, however, that MathML doesn't specify the mechanism by which style information is inherited from the rendering environment.

If the requested `mathsize` of the current font is not available, the renderer should approximate it in the manner likely to lead to the most intelligible, highest quality rendering. Note that many MathML elements automatically change the font size in some of their children; see the discussion in Section 3.1.6.

3.2.2.1 Deprecated style attributes on token elements

The MathML 1.01 style attributes listed below are deprecated in MathML 2 and 3. These attributes were aligned to CSS, but in rendering environments that support CSS, it is preferable to use CSS directly to control the rendering properties corresponding to these attributes, rather than the attributes themselves. However as explained above, direct manipulation of these rendering properties by whatever means should usually be avoided. As a general rule, whenever there is a conflict between these deprecated attributes and the corresponding attributes (Section 3.2.2), the former attributes should be ignored.

The deprecated attributes are:

Name	values	default
fontfamily	*string*	*inherited*
Should be the name of a font that may be available to a MathML renderer, or a CSS font specification; See Section 6.5 and CSS[CSS2] for more information. Deprecated in favor of `mathvariant`.		
fontweight	normal | bold	*inherited*
Specified the font weight for the token. Deprecated in favor of `mathvariant`.		
fontstyle	normal | italic	normal (*except on* `<mi>`)
Specified the font style to use for the token. Deprecated in favor of `mathvariant`.		
fontsize	*length*	*inherited*
Specified the size for the token. Deprecated in favor of `mathsize`.		
color	*color*	*inherited*
Specified the color for the token. Deprecated in favor of `mathcolor`.		

3.2.3 Identifier <mi>

3.2.3.1 Description

An mi element represents a symbolic name or arbitrary text that should be rendered as an identifier. Identifiers can include variables, function names, and symbolic constants. A typical graphical renderer would render an mi element as the characters in its content, with no extra spacing around the characters (except spacing associated with neighboring elements).

Not all 'mathematical identifiers' are represented by mi elements — for example, subscripted or primed variables should be represented using msub or msup respectively. Conversely, arbitrary text playing the role of a 'term' (such as an ellipsis in a summed series) can be represented using an mi element, as shown in an example in Section 3.2.6.4.

It should be stressed that mi is a presentation element, and as such, it only indicates that its content should be rendered as an identifier. In the majority of cases, the contents of an mi will actually represent a mathematical identifier such as a variable or function name. However, as the preceding paragraph indicates, the correspondence between notations that should render like identifiers and notations that are actually intended to represent mathematical identifiers is not perfect. For an element whose semantics is guaranteed to be that of an identifier, see the description of ci in Chapter 4.

3.2.3.2 Attributes

mi elements accept the attributes listed in Section 3.2.2, but in one case with a different default value:

Name	values	default
mathvariant	normal | bold | italic | bold-italic | double-struck | bold-fraktur | script | bold-script | fraktur | sans-serif | bold-sans-serif | sans-serif-italic | sans-serif-bold-italic | monospace | initial | tailed | looped | stretched	*(depends on content; described below)*
Specifies the logical class of the token. The default is "normal" (non-slanted) unless the content is a single character, in which case it would be "italic".		

Note that the deprecated fontstyle attribute defaults in the same way as mathvariant, depending on the content.

Note that for purposes of determining equivalences of Math Alphanumeric Symbol characters (See Section 7.5 and Section 3.2.1.1) the value of the mathvariant attribute should be resolved first, including the special defaulting behavior described above.

3.2.3.3 Examples

```
<mi> x </mi>
<mi> D </mi>
<mi> sin </mi>
<mi mathvariant='script'> L </mi>
<mi></mi>
```

An mi element with no content is allowed; <mi></mi> might, for example, be used by an 'expression editor' to represent a location in a MathML expression which requires a 'term' (according to conventional syntax for mathematics) but does not yet contain one.

Identifiers include function names such as 'sin'. Expressions such as 'sin x' should be written using the character U+2061 (which also has the entity names `&af;` and `&ApplyFunction;`) as shown below; see also the discussion of invisible operators in Section 3.2.5.

```
<mrow>
  <mi> sin </mi>
  <mo> &ApplyFunction; </mo>
  <mi> x </mi>
</mrow>
```

Miscellaneous text that should be treated as a 'term' can also be represented by an `mi` element, as in:

```
<mrow>
  <mn> 1 </mn>
  <mo> + </mo>
  <mi> ... </mi>
  <mo> + </mo>
  <mi> n </mi>
</mrow>
```

When an `mi` is used in such exceptional situations, explicitly setting the `mathvariant` attribute may give better results than the default behavior of some renderers.

The names of symbolic constants should be represented as `mi` elements:

```
<mi> &pi; </mi>
<mi> &ImaginaryI; </mi>
<mi> &ExponentialE; </mi>
```

3.2.4 Number <mn>

3.2.4.1 Description

An `mn` element represents a 'numeric literal' or other data that should be rendered as a numeric literal. Generally speaking, a numeric literal is a sequence of digits, perhaps including a decimal point, representing an unsigned integer or real number. A typical graphical renderer would render an `mn` element as the characters of its content, with no extra spacing around them (except spacing from neighboring elements such as `mo`). `mn` elements are typically rendered in an unslanted font.

The mathematical concept of a 'number' can be quite subtle and involved, depending on the context. As a consequence, not all mathematical numbers should be represented using `mn`; examples of mathematical numbers that should be represented differently are shown below, and include complex numbers, ratios of numbers shown as fractions, and names of numeric constants.

Conversely, since `mn` is a presentation element, there are a few situations where it may desirable to include arbitrary text in the content of an `mn` that should merely render as a numeric literal, even though that content may not be unambiguously interpretable as a number according to any particular standard encoding of numbers as character sequences. As a general rule, however, the `mn` element should be reserved for situations where its content is actually intended to represent a numeric quantity in some fashion. For an element whose semantics are guaranteed to be that of a particular kind of mathematical number, see the description of `cn` in Chapter 4.

3.2.4.2 Attributes

`mn` elements accept the attributes listed in Section 3.2.2.

3.2.4.3 Examples

```
<mn> 2 </mn>
<mn> 0.123 </mn>
<mn> 1,000,000 </mn>
<mn> 2.1e10 </mn>
<mn> 0xFFEF </mn>
<mn> MCMLXIX </mn>
<mn> twenty one </mn>
```

3.2.4.4 Numbers that should not be written using <mn> alone

Many mathematical numbers should be represented using presentation elements other than mn alone; this includes complex numbers, ratios of numbers shown as fractions, and names of numeric constants. Examples of MathML representations of such numbers include:

```
<mrow>
  <mn> 2 </mn>
  <mo> + </mo>
  <mrow>
    <mn> 3 </mn>
    <mo> &InvisibleTimes; </mo>
    <mi> &ImaginaryI; </mi>
  </mrow>
</mrow>
<mfrac> <mn> 1 </mn> <mn> 2 </mn> </mfrac>
<mi> &pi; </mi>
<mi> &ExponentialE; </mi>
```

3.2.5 Operator, Fence, Separator or Accent <mo>

3.2.5.1 Description

An mo element represents an operator or anything that should be rendered as an operator. In general, the notational conventions for mathematical operators are quite complicated, and therefore MathML provides a relatively sophisticated mechanism for specifying the rendering behavior of an mo element. As a consequence, in MathML the list of things that should 'render as an operator' includes a number of notations that are not mathematical operators in the ordinary sense. Besides ordinary operators with infix, prefix, or postfix forms, these include fence characters such as braces, parentheses, and 'absolute value' bars, separators such as comma and semicolon, and mathematical accents such as a bar or tilde over a symbol. We will use the term "operator" in this chapter to refer to operators in this broad sense.

Typical graphical renderers show all mo elements as the characters of their content, with additional spacing around the element determined by its attributes and further described below. Renderers without access to complete fonts for the MathML character set may choose not to render an mo element as precisely the characters in its content in some cases. For example, <mo> ≤ </mo> might be rendered as <= to a terminal. However, as a general rule, renderers should attempt to render the content of an mo element as literally as possible. That is, <mo> ≤ </mo> and <mo> <= </mo> should render differently. The first one should render as a single character representing a less-than-or-equal-to sign, and the second one as the two-character sequence <=.

Operators, in the general sense used here, are subject to essentially the same rendering attributes and rules; subtle distinctions in the rendering of these classes of symbols, when they exist, are supported

using the boolean attributes `fence`, `separator` and `accent`, which can be used to distinguish these cases.

A key feature of the `mo` element is that its default attribute values are set on a case-by-case basis from an 'operator dictionary' as explained below. In particular, default values for `fence`, `separator` and `accent` can usually be found in the operator dictionary and therefore need not be specified on each `mo` element.

Note that some mathematical operators are represented not by `mo` elements alone, but by `mo` elements 'embellished' with (for example) surrounding superscripts; this is further described below. Conversely, as presentation elements, `mo` elements can contain arbitrary text, even when that text has no standard interpretation as an operator; for an example, see the discussion 'Mixing text and mathematics' in Section 3.2.6. See also Chapter 4 for definitions of MathML content elements that are guaranteed to have the semantics of specific mathematical operators.

Note also that linebreaking, as discussed in Section 3.1.7, usually takes place at operators (either before or after, depending on local conventions). Thus, `mo` accepts attributes to encode the desirability of breaking at a particular operator, as well as attributes describing the treatment of the operator and indentation in case the a linebreak is made at that operator.

3.2.5.2 Attributes

`mo` elements accept the attributes listed in Section 3.2.2 and the additional attributes listed here. Since the display of operators is so critical in mathematics, the `mo` element accepts a large number of attributes; these are described in the next three subsections.

Most attributes get their default values from an enclosing `mstyle` element, `math` element, or from the Section 3.2.5.7, as described later in this section. When a value that is listed as "inherited" is not explicitly given on an `mo`, `mstyle` element, `math` element, or found in the operator dictionary for a given `mo` element, the default value shown in parentheses is used. The attributes may also appear on any ancestor of the `math` element, if permitted by the containing document, to provide defaults for all contained `math` elements. In such cases, the attributes would be in the MathML namespace.

Dictionary-based attributes

Name	values	default
form	prefix | infix | postfix	*set by position of operator in an* `mrow`
Specifies the role of the operator in the enclosing expression. This role and the operator content affect the lookup of the operator in the operator dictionary which affects the spacing and other default properties; see Section 3.2.5.7.		
fence	true | false	*set by dictionary* (false)
Specifies whether the operator represents a 'fence', such as a parenthesis. This attribute generally has no direct effect on the visual rendering, but may be useful in specific cases, such as non-visual renderers.		
separator	true | false	*set by dictionary* (false)
Specifies whether the operator represents a 'separator', or punctuation. This attribute generally has no direct effect on the visual rendering, but may be useful in specific cases, such as non-visual renderers.		
lspace	*length*	*set by dictionary* (thickmathspace)
Specifies the leading space appearing before the operator; see Section 3.2.5.7. (Note that before is on the right in a RTL context; see Section 3.1.5).		
rspace	*length*	*set by dictionary* (thickmathspace)
Specifies the trailing space appearing after the operator; see Section 3.2.5.7. (Note that after is on the left in a RTL context; see Section 3.1.5).		
stretchy	true | false	*set by dictionary* (false)
Specifies whether the operator should stretch to the size of adjacent material; see Section 3.2.5.8.		
symmetric	true | false	*set by dictionary* (true)
Specifies whether the operator should be kept symmetric around the baseline when stretchy. Note that the default is true, but this property only applies to vertically stretched symbols. See Section 3.2.5.8.		
maxsize	*length* | infinity	*set by dictionary* (infinity)
Specifies the maximum size of the operator when stretchy; see Section 3.2.5.8.		
minsize	*length*	*set by dictionary* (1em)
Specifies the minimum size of the operator when stretchy; see Section 3.2.5.8.		
largeop	true | false	*set by dictionary* (false)
Specifies whether the operator is considered a 'large' operator, that is, whether it should be drawn larger than normal when `displaystyle="true"` (similar to using TEX's `\displaystyle`). Examples of large operators include `&int;` and `&prod;`. See Section 3.1.6 for more discussion.		
movablelimits	true | false	*set by dictionary* (false)
Specifies whether under- and overscripts attached to this operator 'move' to the more compact sub- and superscript positions when `displaystyle` is false. Examples of operators that typically have `movablelimits="true"` are `&sum;`, `&prod;`, and `lim`. See Section 3.1.6 for more discussion.		
accent	true | false	*set by dictionary* (false)
Specifies whether this operator should be treated as an accent (diacritical mark) when used as an underscript or overscript; see `munder`, `mover` and `munderover`.		

Linebreaking attributes

The following attributes affect when a linebreak does or does not occur, and the appearance of the linebreak when it does occur.

Name	values	default
linebreak	auto | newline | nobreak | goodbreak | badbreak	auto
Specifies the desirability of a linebreak occurring at this operator: the default `"auto"` indicates the renderer should use its default linebreaking algorithm to determine whether to break; `"newline"` is used to force a linebreak; For automatic linebreaking, `"nobreak"` forbids a break; `"goodbreak"` suggests a good position; `"badbreak"` suggests a poor position.		
lineleading	*length*	*inherited* (100%)
Specifies the amount of vertical space to use after a linebreak. For tall lines, it is often clearer to use more leading at linebreaks. Rendering agents are free to choose an appropriate default.		
linebreakstyle	before | after | duplicate | infixlinebreakstyle	*set by dictionary* (after)
Specifies whether a linebreak occurs 'before' or 'after' the operator when a linebreaks occur on this operator; or whether the operator is duplicated. `"before"` causes the operator to appears at the beginning of the new line (but possibly indented); `"after"` causes it to appear at the end of the line before the break. `"duplicate"` places the operator at both positions. `"infixlinebreakstyle"` uses the value that has been specified for infix operators; This value (one of `"before"`, `"after"` or `"duplicate"`) can be specified by the application or bound by `mstyle` (`"before"` corresponds to the most common style of linebreaking).		
linebreakmultchar	*string*	*inhertied* (⁢)
Specifies the character used to make an ⁢ operator visible at a linebreak. For example, `linebreakmultchar="&#xB7;"` would make the multiplication visible as a center dot.		

Indentation attributes

The following attributes affect indentation of the lines making up a formula. Primarily these are to control the positioning of new lines following a linebreak, whether automatic or manual. However, `indentstylefirst` and `indentoffsetfirst` also control the positioning of single line formula without any linebreaks.

Formula indentation only applies to displayed equations (ie. `display="block"`). When these attributes appear on `mo` or `mspace` they apply if a linebreak occurs at that element. When the appear on `mstyle` or `math` elements, they determine defaults for the style to be used for any linebreaks occuring within. Note that except for cases where heavily marked-up manual linebreaking is desired, many of these attributes are most useful when bound on an `mstyle` or `math` element.

Note that since the rendering context, such as available the width and current font, is not always available to the author of the MathML, a render may ignore the values of these attributes if they result in a line in which the remaining width is too small to usefully display the expression or if they result in a line in which the remaining width exceeds the available linewrapping width.

Name	values	default
indentstyle	left | center | right | auto | id	*inherited* (auto)
Specifies the positioning of lines when linebreaking takes place within an `mrow`; see below for discussion of the attribute values.		
indentoffset	*length*	*inherited* (0)
Specifies an additional indentation offset relative to the position determined by `indentstyle`.		
indenttarget	*idref*	*inherited (none)*
Specifies the id of another element whose horizontal position determines the position of indented lines when `indentstyle="id"`. Note that the identified element may be outside of the current `math` element, allowing for inter-expression alignment, or may be within invisible content such as `mphantom`; it must appear *before* being referenced, however. This may lead to an id being unavailable to a given renderer; in such cases, the `indentstyle` should revert to `"auto"`.		
indentstylefirst	left | center | right | auto | id | indentstyle	*inherited* (indentstyle)
Specifies the indentation style to use for the first line of a formula; the value `"indentstyle"` (the default) means to indent the same way as used for the general line.		
indentoffsetfirst	*length* | indentoffset	*inherited* (indentoffset)
Specifies the offset to use for the first line of a formula; the value `"indentoffset"` (the default) means to use the same offset as used for the general line.		
indentstylelast	left | center | right | auto | id | indentstyle	*inherited* (indentstyle)
Specifies the indentation style to use for the last line when a linebreak occurs within a given `mrow`; the value `"indentstyle"` (the default) means to indent the same way as used for the general line. When there are exactly two lines, the value of this attribute should be used for the second line in preference to `indentstyle`.		
indentoffsetlast	*length* | indentoffset	*inherited* (indentoffset)
Specifies the offset to use for the last line when a linebreak occurs within a given `mrow`; the value `"indentoffset"` (the default) means to indent the same way as used for the general line. When there are exactly two lines, the value of this attribute should be used for the second line in preference to `indentoffset`.		

The legal values of indentstyle are:

Value	Meaning
left	Align the left side of the next line to the left side of the line wrapping width
center	Align the center of the next line to the center of the line wrapping width
right	Align the right side of the next line to the right side of the line wrapping width
auto	(default) indent using the renderer's default indenting style; this may be a fixed amount or one that varies with the depth of the element in the mrow nesting or some other similar method.
id	Align the left side of the next line to the left side of the element referenced by the *idref* (given by `indenttarget`); if no such element exists, use `"auto"` as the `indentstyle` value

3.2.5.3 Examples with ordinary operators

```
<mo> + </mo>
<mo> &lt; </mo>
<mo> &le; </mo>
<mo> &lt;= </mo>
<mo> ++ </mo>
<mo> &sum; </mo>
```

```
<mo> .NOT. </mo>
<mo> and </mo>
<mo> &InvisibleTimes; </mo>
<mo mathvariant='bold'> + </mo>
```

3.2.5.4 Examples with fences and separators

Note that the `mo` elements in these examples don't need explicit `fence` or `separator` attributes, since these can be found using the operator dictionary as described below. Some of these examples could also be encoded using the `mfenced` element described in Section 3.3.8.

$(a+b)$

```
<mrow>
  <mo> ( </mo>
  <mrow>
    <mi> a </mi>
    <mo> + </mo>
    <mi> b </mi>
  </mrow>
  <mo> ) </mo>
</mrow>
```

$[0,1)$

```
<mrow>
  <mo> [ </mo>
  <mrow>
    <mn> 0 </mn>
    <mo> , </mo>
    <mn> 1 </mn>
  </mrow>
  <mo> ) </mo>
</mrow>
```

$f(x,y)$

```
<mrow>
  <mi> f </mi>
  <mo> &ApplyFunction; </mo>
  <mrow>
    <mo> ( </mo>
    <mrow>
      <mi> x </mi>
      <mo> , </mo>
      <mi> y </mi>
    </mrow>
    <mo> ) </mo>
  </mrow>
</mrow>
```

3.2.5.5 Invisible operators

Certain operators that are 'invisible' in traditional mathematical notation should be represented using specific entity references within `mo` elements, rather than simply by nothing. The characters used for

these ‘invisible operators’ are:

Character	Entity name	Short name	Examples of use
U+2061	`&ApplyFunction;`	`&af;`	$f(x)\sin x$
U+2062	`&InvisibleTimes;`	`&it;`	xy
U+2063	`&InvisibleComma;`	`&ic;`	m_{12}
U+2064	`&InvisiblePlus;`	`&ip;`	$2\frac{3}{4}$

The MathML representations of the examples in the above table are:

```
<mrow>
  <mi> x </mi>
  <mo> &InvisibleTimes; </mo>
  <mi> y </mi>
</mrow>

<mrow>
  <mn> 2 </mn>
  <mo> &#x2064; </mo>
  <mfrac>
    <mn> 3 </mn>
    <mn> 4 </mn>
  </mfrac>
</mrow>

<mrow>
  <mi> f </mi>
  <mo> &ApplyFunction; </mo>
  <mrow>
    <mo> ( </mo>
    <mi> x </mi>
    <mo> ) </mo>
  </mrow>
</mrow>

<mrow>
  <mi> sin </mi>
  <mo> &ApplyFunction; </mo>
  <mi> x </mi>
</mrow>

<msub>
  <mi> m </mi>
  <mrow>
    <mn> 1 </mn>
    <mo> &InvisibleComma; </mo>
    <mn> 2 </mn>
  </mrow>
</msub>
```

The reasons for using specific `mo` elements for invisible operators include:

- such operators should often have specific effects on visual rendering (particularly spacing

and linebreaking rules) that are not the same as either the lack of any operator, or spacing represented by `mspace` or `mtext` elements;
- these operators should often have specific audio renderings different than that of the lack of any operator;
- automatic semantic interpretation of MathML presentation elements is made easier by the explicit specification of such operators.

For example, an audio renderer might render $f(x)$ (represented as in the above examples) by speaking 'f of x', but use the word 'times' in its rendering of xy. Although its rendering must still be different depending on the structure of neighboring elements (sometimes leaving out 'of' or 'times' entirely), its task is made much easier by the use of a different `mo` element for each invisible operator.

3.2.5.6 Names for other special operators

MathML also includes `&DifferentialD;` (U+2146) for use in an `mo` element representing the differential operator symbol usually denoted by 'd'. The reasons for explicitly using this special character are similar to those for using the special characters for invisible operators described in the preceding section.

3.2.5.7 Detailed rendering rules for <mo> elements

Typical visual rendering behaviors for `mo` elements are more complex than for the other MathML token elements, so the rules for rendering them are described in this separate subsection.

Note that, like all rendering rules in MathML, these rules are suggestions rather than requirements. Furthermore, no attempt is made to specify the rendering completely; rather, enough information is given to make the intended effect of the various rendering attributes as clear as possible.

The operator dictionary

Many mathematical symbols, such as an integral sign, a plus sign, or a parenthesis, have a well-established, predictable, traditional notational usage. Typically, this usage amounts to certain default attribute values for `mo` elements with specific contents and a specific `form` attribute. Since these defaults vary from symbol to symbol, MathML anticipates that renderers will have an 'operator dictionary' of default attributes for `mo` elements (see Appendix C) indexed by each `mo` element's content and `form` attribute. If an `mo` element is not listed in the dictionary, the default values shown in parentheses in the table of attributes for `mo` should be used, since these values are typically acceptable for a generic operator.

Some operators are 'overloaded', in the sense that they can occur in more than one form (prefix, infix, or postfix), with possibly different rendering properties for each form. For example, '+' can be either a prefix or an infix operator. Typically, a visual renderer would add space around both sides of an infix operator, while only in front of a prefix operator. The `form` attribute allows specification of which form to use, in case more than one form is possible according to the operator dictionary and the default value described below is not suitable.

Default value of the `form` attribute

The `form` attribute does not usually have to be specified explicitly, since there are effective heuristic rules for inferring the value of the `form` attribute from the context. If it is not specified, and there is more than one possible form in the dictionary for an `mo` element with given content, the renderer should choose which form to use as follows (but see the exception for embellished operators, described later):

- If the operator is the first argument in an `mrow` of length (i.e. number of arguments) greater than one (ignoring all space-like arguments (see Section 3.2.7) in the determination of both the length and the first argument), the prefix form is used;
- if it is the last argument in an `mrow` of length greater than one (ignoring all space-like arguments), the postfix form is used;
- in all other cases, including when the operator is not part of an `mrow`, the infix form is used.

Note that the `mrow` discussed above may be *inferred*; See Section 3.1.3.1.

Opening fences should have `form="prefix"`, and closing fences should have `form="postfix"`; separators are usually 'infix', but not always, depending on their surroundings. As with ordinary operators, these values do not usually need to be specified explicitly.

If the operator does not occur in the dictionary with the specified form, the renderer should use one of the forms that is available there, in the order of preference: infix, postfix, prefix; if no forms are available for the given `mo` element content, the renderer should use the defaults given in parentheses in the table of attributes for `mo`.

Exception for embellished operators

There is one exception to the above rules for choosing an `mo` element's default `form` attribute. An `mo` element that is 'embellished' by one or more nested subscripts, superscripts, surrounding text or whitespace, or style changes behaves differently. It is the embellished operator as a whole (this is defined precisely, below) whose position in an `mrow` is examined by the above rules and whose surrounding spacing is affected by its form, not the `mo` element at its core; however, the attributes influencing this surrounding spacing are taken from the `mo` element at the core (or from that element's dictionary entry).

For example, the '$+_4$' in $a+_4 b$ should be considered an infix operator as a whole, due to its position in the middle of an `mrow`, but its rendering attributes should be taken from the `mo` element representing the '+', or when those are not specified explicitly, from the operator dictionary entry for `<mo form="infix"> + </mo>`. The precise definition of an 'embellished operator' is:

- an `mo` element;
- or one of the elements `msub`, `msup`, `msubsup`, `munder`, `mover`, `munderover`, `mmultiscripts`, `mfrac`, or `semantics` (Section 5.1), whose first argument exists and is an embellished operator;
- or one of the elements `mstyle`, `mphantom`, or `mpadded`, such that an `mrow` containing the same arguments would be an embellished operator;
- or an `maction` element whose selected sub-expression exists and is an embellished operator;
- or an `mrow` whose arguments consist (in any order) of one embellished operator and zero or more space-like elements.

Note that this definition permits nested embellishment only when there are no intervening enclosing elements not in the above list.

The above rules for choosing operator forms and defining embellished operators are chosen so that in all ordinary cases it will not be necessary for the author to specify a `form` attribute.

Rationale for definition of embellished operators

The following notes are included as a rationale for certain aspects of the above definitions, but should not be important for most users of MathML.

An `mfrac` is included as an 'embellisher' because of the common notation for a differential operator:

```
<mfrac>
  <mo> &DifferentialD; </mo>
  <mrow>
    <mo> &DifferentialD; </mo>
    <mi> x </mi>
  </mrow>
</mfrac>
```

Since the definition of embellished operator affects the use of the attributes related to stretching, it is important that it includes embellished fences as well as ordinary operators; thus it applies to any `mo` element.

Note that an `mrow` containing a single argument is an embellished operator if and only if its argument is an embellished operator. This is because an `mrow` with a single argument must be equivalent in all respects to that argument alone (as discussed in Section 3.3.1). This means that an `mo` element that is the sole argument of an `mrow` will determine its default `form` attribute based on that `mrow`'s position in a surrounding, perhaps inferred, `mrow` (if there is one), rather than based on its own position in the `mrow` in which it is the sole argument.

Note that the above definition defines every `mo` element to be 'embellished' — that is, 'embellished operator' can be considered (and implemented in renderers) as a special class of MathML expressions, of which `mo` is a specific case.

Spacing around an operator

The amount of horizontal space added around an operator (or embellished operator), when it occurs in an `mrow`, can be directly specified by the `lspace` and `rspace` attributes. Note that `lspace` and `rspace` should be interpreted as leading and trailing space, in the case of RTL direction. By convention, operators that tend to bind tightly to their arguments have smaller values for spacing than operators that tend to bind less tightly. This convention should be followed in the operator dictionary included with a MathML renderer.

Some renderers may choose to use no space around most operators appearing within subscripts or superscripts, as is done in TEX.

Non-graphical renderers should treat spacing attributes, and other rendering attributes described here, in analogous ways for their rendering medium. For example, more space might translate into a longer pause in an audio rendering.

3.2.5.8 Stretching of operators, fences and accents

Four attributes govern whether and how an operator (perhaps embellished) stretches so that it matches the size of other elements: `stretchy`, `symmetric`, `maxsize`, and `minsize`. If an operator has the attribute `stretchy="true"`, then it (that is, each character in its content) obeys the stretching rules listed below, given the constraints imposed by the fonts and font rendering system. In practice, typical renderers will only be able to stretch a small set of characters, and quite possibly will only be able to generate a discrete set of character sizes.

There is no provision in MathML for specifying in which direction (horizontal or vertical) to stretch a specific character or operator; rather, when `stretchy="true"` it should be stretched in each direction for which stretching is possible. It is up to the renderer to know in which directions it is able to stretch each character. (Most characters can be stretched in at most one direction by typical renderers, but some renderers may be able to stretch certain characters, such as diagonal arrows, in both directions independently.)

The `minsize` and `maxsize` attributes limit the amount of stretching (in either direction). These two attributes are given as multipliers of the operator's normal size in the direction or directions of stretching, or as absolute sizes using units. For example, if a character has `maxsize="3"`, then it can grow to be no more than three times its normal (unstretched) size.

The `symmetric` attribute governs whether the height and depth above and below the axis of the character are forced to be equal (by forcing both height and depth to become the maximum of the two). An example of a situation where one might set `symmetric="false"` arises with parentheses around a matrix not aligned on the axis, which frequently occurs when multiplying non-square matrices. In this case, one wants the parentheses to stretch to cover the matrix, whereas stretching the parentheses symmetrically would cause them to protrude beyond one edge of the matrix. The `symmetric` attribute only applies to characters that stretch vertically (otherwise it is ignored).

If a stretchy `mo` element is embellished (as defined earlier in this section), the `mo` element at its core is stretched to a size based on the context of the embellished operator as a whole, i.e. to the same size as if the embellishments were not present. For example, the parentheses in the following example (which would typically be set to be stretchy by the operator dictionary) will be stretched to the same size as each other, and the same size they would have if they were not underlined and overlined, and furthermore will cover the same vertical interval:

```
<mrow>
  <munder>
    <mo> ( </mo>
    <mo> &UnderBar; </mo>
  </munder>
  <mfrac>
    <mi> a </mi>
    <mi> b </mi>
  </mfrac>
  <mover>
    <mo> ) </mo>
    <mo> &OverBar; </mo>
  </mover>
</mrow>
```

Note that this means that the stretching rules given below must refer to the context of the embellished operator as a whole, not just to the `mo` element itself.

Example of stretchy attributes

This shows one way to set the maximum size of a parenthesis so that it does not grow, even though its default value is `stretchy="true"`.

```
<mrow>
  <mo maxsize="1"> ( </mo>
  <mfrac>
    <mi> a </mi> <mi> b </mi>
  </mfrac>
  <mo maxsize="1"> ) </mo>
</mrow>
```

The above should render as $(\frac{a}{b})$ as opposed to the default rendering $\left(\frac{a}{b}\right)$.

Note that each parenthesis is sized independently; if only one of them had `maxsize="1"`, they would render with different sizes.

Vertical Stretching Rules

- If a stretchy operator is a direct sub-expression of an `mrow` element, or is the sole direct sub-expression of an `mtd` element in some row of a table, then it should stretch to cover the height and depth (above and below the `axis`) of the non-stretchy direct sub-expressions in the `mrow` element or table row, unless stretching is constrained by `minsize` or `maxsize` attributes.
- In the case of an embellished stretchy operator, the preceding rule applies to the stretchy operator at its core.
- If `symmetric="true"`, then the maximum of the height and depth is used to determine the size, before application of the `minsize` or `maxsize` attributes.
- The preceding rules also apply in situations where the `mrow` element is inferred.

Most common opening and closing fences are defined in the operator dictionary to stretch by default; and they stretch vertically. Also, operators such as `&sum;`, `&int;`, /, and vertical arrows stretch vertically by default.

In the case of a stretchy operator in a table cell (i.e. within an `mtd` element), the above rules assume each cell of the table row containing the stretchy operator covers exactly one row. (Equivalently, the value of the `rowspan` attribute is assumed to be 1 for all the table cells in the table row, including the cell containing the operator.) When this is not the case, the operator should only be stretched vertically to cover those table cells that are entirely within the set of table rows that the operator's cell covers. Table cells that extend into rows not covered by the stretchy operator's table cell should be ignored. See Section 3.5.4.2 for details about the `rowspan` attribute.

Horizontal Stretching Rules

- If a stretchy operator, or an embellished stretchy operator, is a direct sub-expression of an `munder`, `mover`, or `munderover` element, or if it is the sole direct sub-expression of an `mtd` element in some column of a table (see `mtable`), then it, or the `mo` element at its core, should stretch to cover the width of the other direct sub-expressions in the given element (or in the same table column), given the constraints mentioned above.
- If a stretchy operator is a direct sub-expression of an `munder`, `mover`, or `munderover` element, or if it is the sole direct sub-expression of an `mtd` element in some column of a table, then it should stretch to cover the width of the other direct sub-expressions in the given element (or in the same table column), given the constraints mentioned above.
- In the case of an embellished stretchy operator, the preceding rule applies to the stretchy operator at its core.

By default, most horizontal arrows and some accents stretch horizontally.

In the case of a stretchy operator in a table cell (i.e. within an `mtd` element), the above rules assume each cell of the table column containing the stretchy operator covers exactly one column. (Equivalently, the value of the `columnspan` attribute is assumed to be 1 for all the table cells in the table row, including the cell containing the operator.) When this is not the case, the operator should only be stretched horizontally to cover those table cells that are entirely within the set of table columns that the operator's cell covers. Table cells that extend into columns not covered by the stretchy operator's table cell should be ignored. See Section 3.5.4.2 for details about the `rowspan` attribute.

The rules for horizontal stretching include `mtd` elements to allow arrows to stretch for use in commutative diagrams laid out using `mtable`. The rules for the horizontal stretchiness include scripts to make examples such as the following work:

```
<mrow>
```

```
  <mi> x </mi>
  <munder>
    <mo> &RightArrow; </mo>
    <mtext> maps to </mtext>
  </munder>
  <mi> y </mi>
</mrow>
```

This displays as $x \underset{\text{maps to}}{\longrightarrow} y$.

Rules Common to both Vertical and Horizontal Stretching

If a stretchy operator is not required to stretch (i.e. if it is not in one of the locations mentioned above, or if there are no other expressions whose size it should stretch to match), then it has the standard (unstretched) size determined by the font and current `mathsize`.

If a stretchy operator is required to stretch, but all other expressions in the containing element (as described above) are also stretchy, all elements that can stretch should grow to the maximum of the normal unstretched sizes of all elements in the containing object, if they can grow that large. If the value of `minsize` or `maxsize` prevents this then that (min or max) size is used.

For example, in an `mrow` containing nothing but vertically stretchy operators, each of the operators should stretch to the maximum of all of their normal unstretched sizes, provided no other attributes are set that override this behavior. Of course, limitations in fonts or font rendering may result in the final, stretched sizes being only approximately the same.

3.2.5.9 Examples of Linebreaking

The following example demonstrates forced linebreaks and forced alignment:

```
<mrow>
  <mrow>  <mi>f</mi> <mo>&ApplyFunction;</mo> <mo>(</mo> <mi>x</mi> <mo>)</mo>  </mrow>

  <mo id='eq1-equals'>=</mo>
  <mrow>
   <msup>
    <mrow> <mo>(</mo> <mrow> <mi>x</mi> <mo>+</mo> <mn>1</mn> </mrow> <mo>)</mo> </mrow>
    <mn>4</mn>
   </msup>
   <mo linebreak='newline' linebreakstyle='before'
                    indentstyle='id' indenttarget='eq1-equals'>=</mo>
   <mrow>
    <msup> <mi>x</mi> <mn>4</mn> </msup>
    <mo id='eq1-plus'>+</mo>
    <mrow>  <mn>4</mn> <mo>&InvisibleTimes;</mo> <msup> <mi>x</mi> <mn>3</mn> </msup>  </mrow>
    <mo>+</mo>
    <mrow>  <mn>6</mn> <mo>&InvisibleTimes;</mo> <msup> <mi>x</mi> <mn>2</mn> </msup>  </mrow>

    <mo linebreak='newline' linebreakstyle='before'
                     indentstylelast='id' indenttarget='eq1-plus'>+</mo>
    <mrow>  <mn>4</mn> <mo>&InvisibleTimes;</mo> <mi>x</mi>  </mrow>
    <mo>+</mo>
    <mn>1</mn>
   </mrow>
  </mrow>
 </mrow>
```

This displays as

$$f(x) = (x+1)^4$$
$$= x^4 + 4x^3 + 6x^2$$
$$+ 4x + 1$$

Note that because `indentstylelast` defaults to `"indentstyle"`, in the above example `indentstyle` could have been used in place of `indentstylelast`. Also, the specifying `linebreakstyle='before'` is not needed because that is the default value.

3.2.6 Text `<mtext>`

3.2.6.1 Description

An `mtext` element is used to represent arbitrary text that should be rendered as itself. In general, the `mtext` element is intended to denote commentary text.

Note that some text with a clearly defined notational role might be more appropriately marked up using `mi` or `mo`; this is discussed further below.

An `mtext` element can be used to contain 'renderable whitespace', i.e. invisible characters that are intended to alter the positioning of surrounding elements. In non-graphical media, such characters are intended to have an analogous effect, such as introducing positive or negative time delays or affecting rhythm in an audio renderer. This is not related to any whitespace in the source MathML consisting of blanks, newlines, tabs, or carriage returns; whitespace present directly in the source is trimmed and collapsed, as described in Section 2.1.7. Whitespace that is intended to be rendered as part of an element's content must be represented by entity references or `mspace` elements (unless it consists only of single blanks between non-whitespace characters).

3.2.6.2 Attributes

`mtext` elements accept the attributes listed in Section 3.2.2.

See also the warnings about the legal grouping of 'space-like elements' in Section 3.2.7, and about the use of such elements for 'tweaking' in Section 3.1.8.

3.2.6.3 Examples

```
<mtext> Theorem 1: </mtext>
<mtext>   </mtext>
<mtext>    </mtext>
<mtext> /* a comment */ </mtext>
```

3.2.6.4 Mixing text and mathematics

In some cases, text embedded in mathematics could be more appropriately represented using `mo` or `mi` elements. For example, the expression 'there exists $\delta > 0$ such that $f(x) <1$' is equivalent to $\exists \delta > 0 \ni f(x) < 1$ and could be represented as:

```
<mrow>
  <mo> there exists </mo>
  <mrow>
```

```
    <mrow>
      <mi> &delta; </mi>
      <mo> &gt; </mo>
      <mn> 0 </mn>
    </mrow>
    <mo> such that </mo>
    <mrow>
      <mrow>
        <mi> f </mi>
        <mo> &ApplyFunction; </mo>
        <mrow>
          <mo> ( </mo>
          <mi> x </mi>
          <mo> ) </mo>
        </mrow>
      </mrow>
      <mo> &lt; </mo>
      <mn> 1 </mn>
    </mrow>
  </mrow>
</mrow>
```

An example involving an `mi` element is: $x+x^2+\cdots+x^n$. In this example, ellipsis should be represented using an `mi` element, since it takes the place of a term in the sum; (see Section 3.2.3).

On the other hand, expository text within MathML is best represented with an `mtext` element. An example of this is:

Theorem 1: if $x > 1$, then $x^2 > x$.

However, when MathML is embedded in HTML, or another document markup language, the example is probably best rendered with only the two inequalities represented as MathML at all, letting the text be part of the surrounding HTML.

Another factor to consider in deciding how to mark up text is the effect on rendering. Text enclosed in an `mo` element is unlikely to be found in a renderer's operator dictionary, so it will be rendered with the format and spacing appropriate for an 'unrecognized operator', which may or may not be better than the format and spacing for 'text' obtained by using an `mtext` element. An ellipsis entity in an `mi` element is apt to be spaced more appropriately for taking the place of a term within a series than if it appeared in an `mtext` element.

3.2.7 Space `<mspace/>`

3.2.7.1 Description

An `mspace` empty element represents a blank space of any desired size, as set by its attributes. It can also be used to make linebreaking suggestions to a visual renderer. Note that the default values for attributes have been chosen so that they typically will have no effect on rendering. Thus, the `mspace` element is generally used with one or more attribute values explicitly specified.

Note the warning about the legal grouping of 'space-like elements' given below, and the warning about the use of such elements for 'tweaking' in Section 3.1.8. See also the other elements that can render as whitespace, namely `mtext`, `mphantom`, and `maligngroup`.

3.2.7.2 Attributes

In addition to the attributes listed below, `mspace` elements accept the attributes described in Section 3.2.2, but note that `mathvariant` and `mathcolor` have no effect. `mathsize` only affects the interpretation of units in sizing attributes (see Section 2.1.5.2).

Name	values	default
width	*length*	0em
Specifies the desired width of the space.		
height	*length*	0ex
Specifies the desired height (above the baseline) of the space.		
depth	*length*	0ex
Specifies the desired depth (below the baseline) of the space.		
linebreak	auto | newline | nobreak | goodbreak | badbreak	auto
Specifies the desirability of a linebreak at this space.		

Note that if both `spacing` and `width` are used, the width of the `mspace` is the sum of these two contributions.

Linebreaking was originally specified on `mspace` in MathML2, but controlling linebreaking on `mo` is to be preferred. The value `"indentingnewline"` was defined in MathML2 for `mspace`; it is now deprecated. Its meaning is the same as `newline`, which is compatible with its earlier use when no other linebreaking attributes are specified. Note that `linebreak` values on adjacent `mo` and `mspace` elements do not interact; a `"nobreak"` on an `mspace` will not, in itself, inhibit a break on an adjacent `mo` element.

Issue (char): There are two ways that a character value might not be present. The first is that it wasn't part of the MathML. The second is that it was inserted, but the line from the character to the line break was so long that it wrapped and the character ended up on a line prior to the previous line. Is the "Indent" behavior appropriate?

Issue (count): Another possible value, which is similar to what Word uses, is specify a number and that number means 'indent to the ith operator on the previous line'. The operator is not specified.

Issue (align): Another option is to add a new element (or reuse malignmark) and allow the value "AlignMark" as an indent value. In this case, it would align to the mark in the previous line.

Issue (id): Yet another idea is to have indent=id and have an id specified on some element mean the point to be indented to.

3.2.7.3 Examples

```
<mspace spacing="00"/>
<mspace height="3ex" depth="2ex"/>

<mrow>
  <mi>a</mi>
  <mo id="firstop">+</mo>
  <mi>b</mi>
  <mspace linebreak="newline" indentto="firstop"/>
  <mo>+</mo>
  <mi>c</mi>
</mrow>
```

In the last example, `mspace` will cause the line to end after the "b" and the following line to be indented so that the "+" that follows will align with the "+" with `id`="firstop".

3.2.7.4 Definition of space-like elements

A number of MathML presentation elements are 'space-like' in the sense that they typically render as whitespace, and do not affect the mathematical meaning of the expressions in which they appear. As a consequence, these elements often function in somewhat exceptional ways in other MathML expressions. For example, space-like elements are handled specially in the suggested rendering rules for `mo` given in Section 3.2.5. The following MathML elements are defined to be 'space-like':

- an `mtext`, `mspace`, `maligngroup`, or `malignmark` element;
- an `mstyle`, `mphantom`, or `mpadded` element, all of whose direct sub-expressions are space-like;
- an `maction` element whose selected sub-expression exists and is space-like;
- an `mrow` all of whose direct sub-expressions are space-like.

Note that an `mphantom` is *not* automatically defined to be space-like, unless its content is space-like. This is because operator spacing is affected by whether adjacent elements are space-like. Since the `mphantom` element is primarily intended as an aid in aligning expressions, operators adjacent to an `mphantom` should behave as if they were adjacent to the *contents* of the `mphantom`, rather than to an equivalently sized area of whitespace.

3.2.7.5 Legal grouping of space-like elements

Authors who insert space-like elements or `mphantom` elements into an existing MathML expression should note that such elements *are* counted as arguments, in elements that require a specific number of arguments, or that interpret different argument positions differently.

Therefore, space-like elements inserted into such a MathML element should be grouped with a neighboring argument of that element by introducing an `mrow` for that purpose. For example, to allow for vertical alignment on the right edge of the base of a superscript, the expression

```
<msup>
  <mi> x </mi>
  <malignmark edge="right"/>
  <mn> 2 </mn>
</msup>
```

is illegal, because `msup` must have exactly 2 arguments; the correct expression would be:

```
<msup>
  <mrow>
    <mi> x </mi>
    <malignmark edge="right"/>
  </mrow>
  <mn> 2 </mn>
</msup>
```

See also the warning about 'tweaking' in Section 3.1.8.

3.2.8 String Literal `<ms>`

3.2.8.1 Description

The `ms` element is used to represent 'string literals' in expressions meant to be interpreted by computer algebra systems or other systems containing 'programming languages'. By default, string literals are displayed surrounded by double quotes, with no extra spacing added around the string. As explained in

Section 3.2.6, ordinary text embedded in a mathematical expression should be marked up with `mtext`, or in some cases `mo` or `mi`, but never with `ms`.

Note that the string literals encoded by `ms` are made up of characters, `mglyphs` and `malignmarks` rather than 'ASCII strings'. For example, `<ms>&amp;</ms>` represents a string literal containing a single character, `&`, and `<ms>&amp;amp;</ms>` represents a string literal containing 5 characters, the first one of which is `&`.

The content of `ms` elements should be rendered with visible 'escaping' of certain characters in the content, including at least the left and right quoting characters, and preferably whitespace other than individual space characters. The intent is for the viewer to see that the expression is a string literal, and to see exactly which characters form its content. For example, `<ms>double quote is "</ms>` might be rendered as "double quote is \"".

Like all token elements, `ms` *does* trim and collapse whitespace in its content according to the rules of Section 2.1.7, so whitespace intended to remain in the content should be encoded as described in that section.

3.2.8.2 Attributes

`ms` elements accept the attributes listed in Section 3.2.2, and additionally:

Name	values	default
lquote	*string*	`&quot;`
Specifies the opening quote to enclose the content.		
rquote	*string*	`&quot;`
Specifies the closing quote to enclose the content.		

3.2.9 Using images to represent symbols `<mglyph/>`

3.2.9.1 Description

The `mglyph` element provides a mechanism for displaying images to represent non-standard symbols. It is generally used as the content of `mi` or `mo` elements where existing Unicode characters are not adequate.

Unicode defines a large number of characters used in mathematics, and in most cases, glyphs representing these characters are widely available in a variety of fonts. Although these characters should meet almost all users needs, MathML recognizes that mathematics is not static and that new characters and symbols are added when convenient. Characters that become well accepted will likely be eventually incorporated by the Unicode Consortium or other standards bodies, but that is often a lengthy process.

Note that the glyph's `src` attribute uniquely identifies the `mglyph`; two `mglyphs` with the same values for `src` should be considered identical by applications that must determine whether two characters/glyphs are identical.

3.2.9.2 Attributes

`mglyph` elements accept the attributes listed in Section 3.2.2, but note that `mathvariant` and `mathcolor` have no effect. `mathsize` only affects the interpretation of units in sizing attributes (see Section 2.1.5.2). The background color, `mathbackground`, should show through if the specified image has transparency.

`mglyph` also accepts the additional attributes listed here.

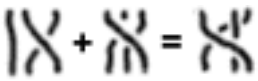
Name	values	default
src	URI	*required*
Specifies the location of the image resource; it may be a URI relative to the base-uri of the source of the MathML, if any. Examples of widely recognized image formats include GIF, JPEG and PNG; However, it may be advisable to omit the extension from the `src` uri, so that a user agent may use content-negotiation to choose the most appropriate format.		
width	*length*	*from image*
Specifies the desired width of the glyph; see `height`.		
height	*length*	*from image*
Specifies the desired height of the glyph. If only one of `width` and `height` are given, the image should be scaled to preserve the aspect ratio; if neither are given, the image should be displayed at its natural size.		
valign	*length*	0em
Specifies the alignment point of the image with respect to the current baseline. A positive value shifts the bottom of the image below the current baseline, while a negative value raises it above.		
alt	*string*	*required*
Provides an alternate name for the glyph. If the specified image can't be found or displayed, the renderer may use this name in a warning message or some unknown glyph notation. The name might also be used by an audio renderer or symbol processing system and should be chosen to be descriptive.		

3.2.9.3 Example

The following example illustrates how a researcher might use the `mglyph` construct with a set of images to work with braid group notation.

```
<mrow>
  <mi><mglyph src="my-braid-23" alt="23braid"/></mi>
  <mo>+</mo>
  <mi><mglyph src="my-braid-132" alt="132braid"/></mi>
  <mo>=</mo>
  <mi><mglyph src="my-braid-13" alt="13braid"/></mi>
</mrow>
```

This might render as:



3.2.9.4 Deprecated Attribute

Originally, `mglyph` was designed to provide access to non-standard fonts. Since this functionality was seldom implemented, nor were downloadable web fonts widely available, this use of `mglyph` has been deprecated. For reference, the following attribute was previously defined.

Name	values
index	*integer*
Specified a position of the desired glyph within the font named by the `fontfamily` attribute (see Section 3.2.2.1). In MathML 1 and 2, both were required attributes; they are now optional and should be ignored unless the `src` attribute is missing.	

3.3 General Layout Schemata

Besides tokens there are several families of MathML presentation elements. One family of elements deals with various 'scripting' notations, such as subscript and superscript. Another family is concerned with matrices and tables. The remainder of the elements, discussed in this section, describe other basic notations such as fractions and radicals, or deal with general functions such as setting style properties and error handling.

3.3.1 Horizontally Group Sub-Expressions <mrow>

3.3.1.1 Description

An mrow element is used to group together any number of sub-expressions, usually consisting of one or more mo elements acting as 'operators' on one or more other expressions that are their 'operands'.

Several elements automatically treat their arguments as if they were contained in an mrow element. See the discussion of inferred mrows in Section 3.1.3. See also mfenced (Section 3.3.8), which can effectively form an mrow containing its arguments separated by commas.

mrow elements are typically rendered visually as a horizontal row of their arguments, left to right in the order in which the arguments occur, in a context with LTR directionality, or right to left. The dir attribute can be used to specify the directionality for a specific mrow, otherwise it inherits the directionality from the context. For aural agents, the arguments would be rendered audibly as a sequence of renderings of the arguments. The description in Section 3.2.5 of suggested rendering rules for mo elements assumes that all horizontal spacing between operators and their operands is added by the rendering of mo elements (or, more generally, embellished operators), not by the rendering of the mrows they are contained in.

MathML provides support for both automatic and manual linebreaking of expressions (that is, to break excessively long expressions into several lines). All such linebreaks take place within mrows, whether they are explicitly marked up in the document, or inferred (See Section 3.1.3.1), although the control of linebreaking is effected through attributes on other elements (See Section 3.1.7).

3.3.1.2 Attributes

mrow elements accept the attributes listed in Section 2.1.6 and the dir attribute as described in Section 3.1.5.1.

3.3.1.3 Proper grouping of sub-expressions using <mrow>

Sub-expressions should be grouped by the document author in the same way as they are grouped in the mathematical interpretation of the expression; that is, according to the underlying 'syntax tree' of the expression. Specifically, operators and their mathematical arguments should occur in a single mrow; more than one operator should occur directly in one mrow only when they can be considered (in a syntactic sense) to act together on the interleaved arguments, e.g. for a single parenthesized term and its parentheses, for chains of relational operators, or for sequences of terms separated by + and −. A precise rule is given below.

Proper grouping has several purposes: it improves display by possibly affecting spacing; it allows for more intelligent linebreaking and indentation; and it simplifies possible semantic interpretation of presentation elements by computer algebra systems, and audio renderers.

Although improper grouping will sometimes result in suboptimal renderings, and will often make interpretation other than pure visual rendering difficult or impossible, any grouping of expressions using

mrow is allowed in MathML syntax; that is, renderers should not assume the rules for proper grouping will be followed.

<mrow> of one argument

MathML renderers are required to treat an mrow element containing exactly one argument as equivalent in all ways to the single argument occurring alone, provided there are no attributes on the mrow element's start tag. If there are attributes on the mrow element's start tag, no requirement of equivalence is imposed. This equivalence condition is intended to simplify the implementation of MathML-generating software such as template-based authoring tools. It directly affects the definitions of embellished operator and space-like element and the rules for determining the default value of the form attribute of an mo element; see Section 3.2.5 and Section 3.2.7. See also the discussion of equivalence of MathML expressions in Section 2.3.

Precise rule for proper grouping

A precise rule for when and how to nest sub-expressions using mrow is especially desirable when generating MathML automatically by conversion from other formats for displayed mathematics, such as TEX, which don't always specify how sub-expressions nest. When a precise rule for grouping is desired, the following rule should be used:

Two adjacent operators (i.e. mo elements, possibly embellished), possibly separated by operands (i.e. anything other than operators), should occur in the same mrow only when the leading operator has an infix or prefix form (perhaps inferred), the following operator has an infix or postfix form, and the operators have the same priority in the operator dictionary (Appendix C). In all other cases, nested mrows should be used.

When forming a nested mrow (during generation of MathML) that includes just one of two successive operators with the forms mentioned above (which mean that either operator could in principle act on the intervening operand or operands), it is necessary to decide which operator acts on those operands directly (or would do so, if they were present). Ideally, this should be determined from the original expression; for example, in conversion from an operator-precedence-based format, it would be the operator with the higher precedence.

Note that the above rule has no effect on whether any MathML expression is valid, only on the recommended way of generating MathML from other formats for displayed mathematics or directly from written notation.

(Some of the terminology used in stating the above rule in defined in Section 3.2.5.)

3.3.1.4 Examples

As an example, $2x+y-z$ should be written as:

```
<mrow>
  <mrow>
    <mn> 2 </mn>
    <mo> &InvisibleTimes; </mo>
    <mi> x </mi>
  </mrow>
  <mo> + </mo>
  <mi> y </mi>
  <mo> - </mo>
```

```
  <mi> z </mi>
</mrow>
```

The proper encoding of (x, y) furnishes a less obvious example of nesting `mrows`:

```
<mrow>
  <mo> ( </mo>
  <mrow>
    <mi> x </mi>
    <mo> , </mo>
    <mi> y </mi>
  </mrow>
  <mo> ) </mo>
</mrow>
```

In this case, a nested `mrow` is required inside the parentheses, since parentheses and commas, thought of as fence and separator 'operators', do not act together on their arguments.

3.3.2 Fractions `<mfrac>`

3.3.2.1 Description

The `mfrac` element is used for fractions. It can also be used to mark up fraction-like objects such as binomial coefficients and Legendre symbols. The syntax for `mfrac` is

`<mfrac>` *numerator* *denominator* `</mfrac>`

The `mfrac` element sets `displaystyle` to `"false"`, or if it was already false increments `scriptlevel` by 1, within *numerator* and *denominator*. (See Section 3.1.6.)

3.3.2.2 Attributes

`mfrac` elements accept the attributes listed below in addition to those listed in Section 2.1.6.

Name	values	default
linethickness	*length* | thin | medium | thick	medium
Specifies the thickness of the horizontal 'fraction bar', or 'rule' The default value is `"medium"`, `"thin"` is thinner, but visible, `"thick"` is thicker; the exact thickness of these is left up to the rendering agent.		
numalign	left | center | right	center
Specifies the alignment of the numerator over the fraction.		
denomalign	left | center | right	center
Specifies the alignment of the denominator under the fraction.		
bevelled	true | false	false
Specifies whether the fraction should be displayed in a beveled style (the numerator slightly raised, the denominator slightly lowered and both separated by a slash), rather than "build up" vertically. See below for an example.		

Thicker lines (eg. `linethickness="thick"`) might be used with nested fractions; a value of "0" renders without the bar such as for binomial coefficients. These cases are shown below:

$$\frac{\binom{a}{b} \quad \frac{a}{b}}{\frac{c}{d}}$$

An example illustrating the bevelled form is show below:

$$\frac{1}{x^3+\frac{x}{3}} = 1 \Big/ x^3+\frac{x}{3}$$

In a RTL directionality context, the numerator leads (on the right), the denominator follows (on the left) and the diagonal line slants upwards going from right to left. Although this format is an established convention, it is not universally followed; for situations where a forward slash is desired in a RTL context, alternative markup, such as an `mo` within an `mrow` should be used.

3.3.2.3 Examples

The examples shown above can be represented in MathML as:

```
<mrow>
   <mo> ( </mo>
   <mfrac linethickness="0">
      <mi> a </mi>
      <mi> b </mi>
   </mfrac>
   <mo> ) </mo>
</mrow>
<mfrac linethickness="2">
   <mfrac>
      <mi> a </mi>
      <mi> b </mi>
   </mfrac>
   <mfrac>
      <mi> c </mi>
      <mi> d </mi>
   </mfrac>
</mfrac>
<mfrac>
   <mn> 1 </mn>
   <mrow>
      <msup>
         <mi> x </mi>
         <mn> 3 </mn>
       </msup>
       <mo> + </mo>
       <mfrac>
         <mi> x </mi>
         <mn> 3 </mn>
       </mfrac>
   </mrow>
</mfrac>
<mo> = </mo>
<mfrac bevelled="true">
   <mn> 1 </mn>
   <mrow>
```

```
      <msup>
         <mi> x </mi>
         <mn> 3 </mn>
       </msup>
       <mo> + </mo>
       <mfrac>
         <mi> x </mi>
         <mn> 3 </mn>
       </mfrac>
   </mrow>
</mfrac>
```

A more generic example is:

```
<mfrac>
   <mrow>
      <mn> 1 </mn>
      <mo> + </mo>
      <msqrt>
         <mn> 5 </mn>
      </msqrt>
   </mrow>
   <mn> 2 </mn>
</mfrac>
```

3.3.3 Radicals <msqrt>, <mroot>

3.3.3.1 Description

These elements construct radicals. The msqrt element is used for square roots, while the mroot element is used to draw radicals with indices, e.g. a cube root. The syntax for these elements is:

```
<msqrt>  base  </msqrt>
<mroot>  base   index  </mroot>
```

The mroot element requires exactly 2 arguments. However, msqrt accepts a single argument, possibly being an inferred mrow of multiple children; see Section 3.1.3. The mroot element increments scriptlevel by 2, and sets displaystyle to "false", within *index*, but leaves both attributes unchanged within *base*. The msqrt element leaves both attributes unchanged within its argument. (See Section 3.1.6.)

Note that in a RTL directionality, the surd begins on the right, rather than the left, along with the index in the case of mroot.

3.3.3.2 Attributes

msqrt and mroot elements accept the attributes listed in Section 2.1.6.

3.3.4 Style Change <mstyle>

3.3.4.1 Description

The mstyle element is used to make style changes that affect the rendering of its contents. mstyle can be given any attribute accepted by any MathML presentation element provided that the attribute value

is inherited, computed or has a default value; presentation element attributes whose values are required are not accepted by the mstyle element. In addition mstyle can also be given certain special attributes listed below.

The mstyle element accepts a single argument, possibly being an inferred mrow of multiple children; see Section 3.1.3.

Loosely speaking, the effect of the mstyle element is to change the default value of an attribute for the elements it contains. Style changes work in one of several ways, depending on the way in which default values are specified for an attribute. The cases are:

- Some attributes, such as displaystyle or scriptlevel (explained below), are inherited from the surrounding context when they are not explicitly set. Specifying such an attribute on an mstyle element sets the value that will be inherited by its child elements. Unless a child element overrides this inherited value, it will pass it on to its children, and they will pass it to their children, and so on. But if a child element does override it, either by an explicit attribute setting or automatically (as is common for scriptlevel), the new (overriding) value will be passed on to that element's children, and then to their children, etc, until it is again overridden.
- Other attributes, such as linethickness on mfrac, have default values that are not normally inherited. That is, if the linethickness attribute is not set on the start tag of an mfrac element, it will normally use the default value of "1", even if it was contained in a larger mfrac element that set this attribute to a different value. For attributes like this, specifying a value with an mstyle element has the effect of changing the default value for all elements within its scope. The net effect is that setting the attribute value with mstyle propagates the change to all the elements it contains directly or indirectly, except for the individual elements on which the value is overridden. Unlike in the case of inherited attributes, elements that explicitly override this attribute have no effect on this attribute's value in their children.
- Another group of attributes, such as stretchy and form, are computed from operator dictionary information, position in the enclosing mrow, and other similar data. For these attributes, a value specified by an enclosing mstyle overrides the value that would normally be computed.

Note that attribute values inherited from an mstyle in any manner affect a given element in the mstyle's content only if that attribute is not given a value in that element's start tag. On any element for which the attribute is set explicitly, the value specified on the start tag overrides the inherited value. The only exception to this rule is when the value given on the start tag is documented as specifying an incremental change to the value inherited from that element's context or rendering environment.

Note also that the difference between inherited and non-inherited attributes set by mstyle, explained above, only matters when the attribute is set on some element within the mstyle's contents that has children also setting it. Thus it never matters for attributes, such as mathcolor, which can only be set on token elements (or on mstyle itself).

There are several exceptional elements, mpadded, mtable, mtr, mlabeledtr and mtd that have attributes which cannot be set with mstyle. The mpadded and mtable elements share attribute names with the mspace element. The mtable, mtr, mlabeledtr and mtd all share attribute names. Similarly, mpadded and mo elements also share an attribute name. Since the syntax for the values these shared attributes accept differs between elements, MathML specifies that when the attributes height, width or depth are specified on an mstyle element, they apply only to mspace elements, and not the corresponding attributes of mpadded or mtable. Similarly, when rowalign, columnalign or groupalign are specified on an mstyle element, the apply only to the mtable element, and not the row and cell elements. Finally, when lspace is set with mstyle, it applies only to the mo element and not mpadded.

3.3.4.2 Attributes

As stated above, `mstyle` accepts all attributes of all MathML presentation elements which do not have required values. That is, all attributes which have an explicit default value or a default value which is inherited or computed are accepted by the `mstyle` element.

`mstyle` elements accept the attributes listed in Section 2.1.6.

Additionally, `mstyle` can be given the following special attributes that are implicitly inherited by every MathML element as part of its rendering environment:

Name	values	default
scriptlevel	[+ | -] *unsigned-integer*	*inherited*
Changes the `scriptlevel` in effect for the children. When the value is given without a sign, it sets `scriptlevel` to the specified value; when a sign is given, it increments ("+") or decrements ("-") the current value. (Note that large decrements can result in negative values of `scriptlevel`, but these values are considered legal.) See Section 3.1.6.		
displaystyle	true | false	*inherited*
Changes the `displaystyle` in effect for the children. See Section 3.1.6.		
scriptsizemultiplier	*number*	0.71
Specifies the multiplier to be used to adjust font size due to changes in `scriptlevel`. See Section 3.1.6.		
scriptminsize	*length*	8pt
Specifies the minimum font size allowed due to changes in `scriptlevel`. Note that this does not limit the font size due to changes to `mathsize`. See Section 3.1.6.		
mathbackground	*color* | transparent	transparent
Specifies the default background color to be used for displaying the content.		
infixlinebreakstyle	before | after | duplicate	before
Specifies the default linebreakstyle to use for infix operators; see Section 3.2.5.2		
decimalseparator	*character*	.
specifies the default separator used to horizontally align the rows of an `mstack`.		

If `scriptlevel` is changed incrementally by an `mstyle` element that also sets certain other attributes, the overall effect of the changes may depend on the order in which they are processed. In such cases, the attributes in the following list should be processed in the following order, regardless of the order in which they occur in the XML-format attribute list of the `mstyle` start tag: `scriptsizemultiplier`, `scriptminsize`, `scriptlevel`, `mathsize`.

Precise background region not specified

The suggested MathML visual rendering rules do not define the precise extent of the region whose background is affected by using the `background` attribute on `mstyle`, except that, when `mstyle`'s content does not have negative dimensions and its drawing region is not overlapped by other drawing due to surrounding negative spacing, this region should lie behind all the drawing done to render the content of the `mstyle`, but should not lie behind any of the drawing done to render surrounding expressions. The effect of overlap of drawing regions caused by negative spacing on the extent of the region affected by the `background` attribute is not defined by these rules.

Deprecated Attributes

MathML2 allowed the binding of *namedspaces* to new values. It appears that this capability was never implemented, and is now deprecated; *namedspaces* are now considered constants. For backwards

compatibility, the following attributes are accepted on the `mstyle` element, but are expected to have no effect.

Name	values	default
veryverythinmathspace	*length*	0.0555556em
verythinmathspace	*length*	0.111111em
thinmathspace	*length*	0.166667em
mediummathspace	*length*	0.222222em
thickmathspace	*length*	0.277778em
verythickmathspace	*length*	0.333333em
veryverythickmathspace	*length*	0.388889em

3.3.4.3 Examples

The example of limiting the stretchiness of a parenthesis shown in the section on <mo>,

```
<mrow>
   <mo maxsize="1"> ( </mo>
   <mfrac> <mi> a </mi> <mi> b </mi> </mfrac>
   <mo maxsize="1"> ) </mo>
</mrow>
```

can be rewritten using `mstyle` as:

```
<mstyle maxsize="1">
   <mrow>
      <mo> ( </mo>
      <mfrac> <mi> a </mi> <mi> b </mi> </mfrac>
      <mo> ) </mo>
   </mrow>
</mstyle>
```

3.3.5 Error Message `<merror>`

3.3.5.1 Description

The `merror` element displays its contents as an 'error message'. This might be done, for example, by displaying the contents in red, flashing the contents, or changing the background color. The contents can be any expression or expression sequence.

`merror` accepts a single argument possibly being an inferred `mrow` of multiple children; see Section 3.1.3.

The intent of this element is to provide a standard way for programs that *generate* MathML from other input to report syntax errors in their input. Since it is anticipated that preprocessors that parse input syntaxes designed for easy hand entry will be developed to generate MathML, it is important that they have the ability to indicate that a syntax error occurred at a certain point. See Section 2.3.2.

The suggested use of `merror` for reporting syntax errors is for a preprocessor to replace the erroneous part of its input with an `merror` element containing a description of the error, while processing the surrounding expressions normally as far as possible. By this means, the error message will be rendered where the erroneous input would have appeared, had it been correct; this makes it easier for an author to determine from the rendered output what portion of the input was in error.

No specific error message format is suggested here, but as with error messages from any program, the format should be designed to make as clear as possible (to a human viewer of the rendered error

message) what was wrong with the input and how it can be fixed. If the erroneous input contains correctly formatted subsections, it may be useful for these to be preprocessed normally and included in the error message (within the contents of the `merror` element), taking advantage of the ability of `merror` to contain arbitrary MathML expressions rather than only text.

3.3.5.2 Attributes

`merror` elements accept the attributes listed in Section 2.1.6.

3.3.5.3 Example

If a MathML syntax-checking preprocessor received the input

```
<mfraction>
   <mrow> <mn> 1 </mn> <mo> + </mo> <msqrt> <mn> 5 </mn> </msqrt> </mrow>
   <mn> 2 </mn>
</mfraction>
```

which contains the non-MathML element `mfraction` (presumably in place of the MathML element `mfrac`), it might generate the error message

```
<merror>
   <mtext> Unrecognized element: mfraction;
           arguments were:  </mtext>
   <mrow> <mn> 1 </mn> <mo> + </mo> <msqrt> <mn> 5 </mn> </msqrt> </mrow>
   <mtext>  and  </mtext>
   <mn> 2 </mn>
</merror>
```

Note that the preprocessor's input is not, in this case, valid MathML, but the error message it outputs is valid MathML.

3.3.6 Adjust Space Around Content `<mpadded>`

3.3.6.1 Description

An `mpadded` element renders the same as its content, but with its 'bounding box' and position modified according to its attributes. It does not rescale (stretch or shrink) its content, but affects the relative position of the content with respect to surrounding elements. While the name of the element reflects the use of `mpadded` to add 'padding', or extra space, around its content, negative 'padding' can cause the content of `mpadded` to be rendered outside the `mpadded` element's bounding box; See Section 3.1.8 for warnings about several potential pitfalls of this effect.

The `mpadded` element accepts a single argument possibly being an inferred `mrow` of multiple children; see Section 3.1.3.

It is suggested that audio renderers add (or shorten) time delays based on the attributes representing horizontal space (`width` and `lspace`).

3.3.6.2 Attributes

`mpadded` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
width	[+ | -] *unsigned-number* (% [*pseudo-unit*] | *pseudo-unit* | *unit* | *namedspace*)	*same as content*
Sets or increments the width of the `mpadded` element. See below for discussion.		
lspace	[+ | -] *unsigned-number* (% [*pseudo-unit*] | *pseudo-unit* | *unit* | *namedspace*)	*same as content*
Sets or increments the leading space of the `mpadded` element being the space between the preceding content and the child content. See below for discussion.		
height	[+ | -] *unsigned-number* (% [*pseudo-unit*] | *pseudo-unit* | *unit*)	*same as content*
Sets or increments the height of the `mpadded` element. See below for discussion.		
depth	[+ | -] *unsigned-number* (% [*pseudo-unit*] | *pseudo-unit* | *unit*)	*same as content*
Sets or increments the depth of the `padded` element. See below for discussion.		

(The *pseudo-unit* syntax symbol is described below.)

These attributes modify the size and position of the 'bounding box' of the `mpadded` element. The typographical layout parameters defined by these attributes are described in the next subsection. Depending on the format of the attribute value, a dimension may be set to a new value, or to an incremented or decremented version of the content's corresponding dimension. Values may be specified as multiples or percentages of any of the dimensions of the normal rendering of the element's content (using so-called 'pseudo-units'), or they can be set directly using standard units Section 2.1.5.2.

If value begins with a + or − sign, it specifies an *increment* or *decrement* of the corresponding dimension by the following length value (extended as explained below). Otherwise, the corresponding dimension is set directly to the following length value. Note that signs are thus not allowed in the following length, and these attributes cannot be set directly to negative values.

Length values (excluding any sign) can be specified in several formats. Each format begins with an *unsigned-number*, which may be followed by a % sign (effectively scaling the number) and an optional *pseudo-unit*, by a *pseudo-unit* alone, or by a *units* (excepting %). The possible *pseudo-units* are the keywords `width`, `lspace`, `height`, and `depth`; they each represent the length of the same-named dimension of the `mpadded` element's content (not of the `mpadded` element itself).

For any of these length formats, the resulting length is the product of the number (possibly including the %) and the following *pseudo-unit*, *units*, *namedspace* or the default value for the attribute if no such unit or space is given.

Some examples of attribute formats using pseudo-units (explicit or default) are as follows:
`depth="100% height"` and `depth="1.0 height"` both set the depth of the `mpadded` element to the height of its content. `depth="105%"` sets the depth to 1.05 times the content's depth, and either `depth="+100%"` or `depth="200%"` sets the depth to twice the content's depth.

The rules given above imply that all of the following attribute settings have the same effect, which is to leave the content's dimensions unchanged:

```
<mpadded width="+0em"> ... </mpadded>
<mpadded width="+0%"> ... </mpadded>
<mpadded width="-0em"> ... </mpadded>
<mpadded width="- 0 height"> ... </mpadded>
<mpadded width="100%"> ... </mpadded>
<mpadded width="100% width"> ... </mpadded>
<mpadded width="1 width"> ... </mpadded>
<mpadded width="1.0 width"> ... </mpadded>
<mpadded> ... </mpadded>
```

3.3.6.3 Meanings of size and position attributes

See Appendix D for further information about some of the typesetting terms used here.

The content of an `mpadded` element defines some mathematical notation (e.g. a character, a fraction, an expression, etc.) that can be regarded as single typographical element with a positioning point at a fixed relative location to its natural visual bounding box.

The size of the bounding box and the relative location of the positioning point for the `mpadded` element are defined by its size and positioning attributes. The argument of the `mpadded` element is always rendered with its natural positioning point coinciding with the positioning point of the `mpadded` elements. Thus, by using the size and position attributes of `mpadded` to expand or shrink its bounding box, the visual effect is to pad the child content or the move the content so that it overlaps neighboring elements.

Issue (clipping): Should the bounding box act as a clipping rectangle? Nope.

The `width` attribute refers to the horizontal width of the natural visual bounding box of the `mpadded` element's content. Decreasing the width causes following content to be rendered closer to the positioning point than would normally have occurred; setting the width to 0 causes it to completely overlap the argument. Decreasing the width should generally be avoided.

The `lspace` attribute refers to the amount of space between the left edge of the bounding box and the positioning poin of the `mpadded` element. This is sometimes called the left side bearing in typesetting. Increasing the `lspace` increases the space between the preceding content and the child content, introducing padding at the left edge of the child content rendering. Decreasing the lspace may cause overprinting of the preceding content, and should generally be avoided.

The `height` attribute refers to the amount of vertical space between the baseline of the `mpadded` element's child content, and the top of the `mpadded` element's bounding box. This is also known as the ascent in typography. Increasing the height increases the space between the child content and any content above it, thus introducing padding at the top of the child content rendering. Decreasing the height causes any content above it to be rendered lower than normal, possibly overlapping the rendering of child content, and should generally be avoided.

The `depth` attribute refers to the amount of vertical space between the bottom of the `mpadded`'s bounding box and the baseline of the child content. It is also know as the descent in typography. It functions analogously to the `height` attribute above.

MathML renderers should ensure that, except for the effects of the attributes, relative spacing between the contents of `mpadded` and surrounding MathML elements is not modified by replacing an `mpadded` element with an `mrow` element with the same content. This holds even if linebreaking occurs within the `mpadded` element. However, if an `mpadded` element with non-default attribute values is subjected to linebreaking, MathML does not define how its attributes or rendering interact with the linebreaking algorithm.

Issue (examples): One or more illustrated examples should be included.

3.3.7 Making Sub-Expressions Invisible `<mphantom>`

3.3.7.1 Description

The `mphantom` element renders invisibly, but with the same size and other dimensions, including baseline position, that its contents would have if they were rendered normally. `mphantom` can be used to align parts of an expression by invisibly duplicating sub-expressions.

The `mphantom` element accepts a single argument possibly being an inferred `mrow` of multiple children; see Section 3.1.3.

Note that it is possible to wrap both an mphantom and an mpadded element around one MathML expression, as in <mphantom><mpadded attribute-settings> ... </mpadded></mphantom>, to change its size and make it invisible at the same time.

MathML renderers should ensure that the relative spacing between the contents of an mphantom element and the surrounding MathML elements is the same as it would be if the mphantom element were replaced by an mrow element with the same content. This holds even if linebreaking occurs within the mphantom element.

For the above reason, mphantom is *not* considered space-like (Section 3.2.7) unless its content is space-like, since the suggested rendering rules for operators are affected by whether nearby elements are space-like. Even so, the warning about the legal grouping of space-like elements may apply to uses of mphantom.

3.3.7.2 Attributes

mphantom elements accept the attributes listed in Section 2.1.6.

3.3.7.3 Examples

There is one situation where the preceding rules for rendering an mphantom may not give the desired effect. When an mphantom is wrapped around a subsequence of the arguments of an mrow, the default determination of the form attribute for an mo element within the subsequence can change. (See the default value of the form attribute described in Section 3.2.5.) It may be necessary to add an explicit form attribute to such an mo in these cases. This is illustrated in the following example.

In this example, mphantom is used to ensure alignment of corresponding parts of the numerator and denominator of a fraction:

```
<mfrac>
  <mrow>
    <mi> x </mi>
    <mo> + </mo>
    <mi> y </mi>
    <mo> + </mo>
    <mi> z </mi>
  </mrow>
  <mrow>
    <mi> x </mi>
    <mphantom>
      <mo form="infix"> + </mo>
      <mi> y </mi>
    </mphantom>
    <mo> + </mo>
    <mi> z </mi>
  </mrow>
</mfrac>
```

This would render as something like

$$\frac{x+y+z}{x\quad\ +z}$$

rather than as

$$\frac{x+y+z}{x+z}$$

The explicit attribute setting `form="infix"` on the `mo` element inside the `mphantom` sets the `form` attribute to what it would have been in the absence of the surrounding `mphantom`. This is necessary since otherwise, the + sign would be interpreted as a prefix operator, which might have slightly different spacing.

Alternatively, this problem could be avoided without any explicit attribute settings, by wrapping each of the arguments `<mo>+</mo>` and `<mi>y</mi>` in its own `mphantom` element, i.e.

```
<mfrac>
   <mrow>
      <mi> x </mi>
      <mo> + </mo>
      <mi> y </mi>
      <mo> + </mo>
      <mi> z </mi>
   </mrow>
   <mrow>
      <mi> x </mi>
      <mphantom>
         <mo> + </mo>
      </mphantom>
      <mphantom>
         <mi> y </mi>
      </mphantom>
      <mo> + </mo>
      <mi> z </mi>
   </mrow>
</mfrac>
```

3.3.8 Expression Inside Pair of Fences `<mfenced>`

3.3.8.1 Description

The `mfenced` element provides a convenient form in which to express common constructs involving fences (i.e. braces, brackets, and parentheses), possibly including separators (such as comma) between the arguments.

For example, `<mfenced> <mi>x</mi> </mfenced>` renders as '(x)' and is equivalent to

```
<mrow> <mo> ( </mo> <mi>x</mi> <mo> ) </mo> </mrow>
```

and `<mfenced> <mi>x</mi> <mi>y</mi> </mfenced>` renders as '(x, y)' and is equivalent to

```
<mrow>
  <mo> ( </mo>
  <mrow> <mi>x</mi> <mo>,</mo> <mi>y</mi> </mrow>
  <mo> ) </mo>
</mrow>
```

Individual fences or separators are represented using `mo` elements, as described in Section 3.2.5. Thus, any `mfenced` element is completely equivalent to an expanded form described below; either form can

be used in MathML, at the convenience of an author or of a MathML-generating program. A MathML renderer is required to render either of these forms in exactly the same way.

In general, an `mfenced` element can contain zero or more arguments, and will enclose them between fences in an `mrow`; if there is more than one argument, it will insert separators between adjacent arguments, using an additional nested `mrow` around the arguments and separators for proper grouping (Section 3.3.1). The general expanded form is shown below. The fences and separators will be parentheses and comma by default, but can be changed using attributes, as shown in the following table.

3.3.8.2 Attributes

`mfenced` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
open	*string*	(
Specifies the opening delimiter. Since it is used as the content of an `mo` element, any whitespace will be trimmed and collapsed as described in Section 2.1.7.		
close	*string*	)
Specifies the closing delimiter. Since it is used as the content of an `mo` element, any whitespace will be trimmed and collapsed as described in Section 2.1.7.		
separators	*character* *	,
Specifies a sequence of zero or more separator characters. Each pair of arguments is displayed separated by the corresponding separator (none appears after the last argument). If there are too many separators, the excess are ignored; if there are too few, the last separator is repeated. Any whitespace within `separators` is ignored.		

A generic `mfenced` element, with all attributes explicit, looks as follows:

```
<mfenced open="opening-fence"
         close="closing-fence"
         separators="sep#1 sep#2 ... sep#(n-1)" >
   arg#1
   ...
   arg#n
</mfenced>
```

In a RTL directionality context, since the initial text direction is RTL, characters in the `open` and `close` attributes that have a mirroring counterpart will be rendered in that mirrored form. In particular, the default values will render correctly as a parenthesized sequence in both LTR and RTL contexts.

The general `mfenced` element shown above is equivalent to the following expanded form:

```
<mrow>
   <mo fence="true"> opening-fence </mo>
   <mrow>
      arg#1
      <mo separator="true"> sep#1 </mo>
      ...
      <mo separator="true"> sep#(n-1) </mo>
      arg#n
   </mrow>
   <mo fence="true"> closing-fence </mo>
</mrow>
```

Each argument except the last is followed by a separator. The inner mrow is added for proper grouping, as described in Section 3.3.1.

When there is only one argument, the above form has no separators; since <mrow> arg#1 </mrow> is equivalent to arg#1 (as described in Section 3.3.1), this case is also equivalent to:

```
<mrow>
   <mo fence="true"> opening-fence </mo>
   arg#1
   <mo fence="true"> closing-fence </mo>
</mrow>
```

If there are too many separator characters, the extra ones are ignored. If separator characters are given, but there are too few, the last one is repeated as necessary. Thus, the default value of separators="," is equivalent to separators=",,", separators=",,,", etc. If there are no separator characters provided but some are needed, for example if separators=" " or "" and there is more than one argument, then no separator elements are inserted at all — that is, the elements <mo separator="true"> sep#i </mo> are left out entirely. Note that this is different from inserting separators consisting of mo elements with empty content.

Finally, for the case with no arguments, i.e.

```
<mfenced open="opening-fence"
         close="closing-fence"
         separators="anything" >
</mfenced>
```

the equivalent expanded form is defined to include just the fences within an mrow:

```
<mrow>
   <mo fence="true"> opening-fence </mo>
   <mo fence="true"> closing-fence </mo>
</mrow>
```

Note that not all 'fenced expressions' can be encoded by an mfenced element. Such exceptional expressions include those with an 'embellished' separator or fence or one enclosed in an mstyle element, a missing or extra separator or fence, or a separator with multiple content characters. In these cases, it is necessary to encode the expression using an appropriately modified version of an expanded form. As discussed above, it is always permissible to use the expanded form directly, even when it is not necessary. In particular, authors cannot be guaranteed that MathML preprocessors won't replace occurrences of mfenced with equivalent expanded forms.

Note that the equivalent expanded forms shown above include attributes on the mo elements that identify them as fences or separators. Since the most common choices of fences and separators already occur in the operator dictionary with those attributes, authors would not normally need to specify those attributes explicitly when using the expanded form directly. Also, the rules for the default form attribute (Section 3.2.5) cause the opening and closing fences to be effectively given the values form="prefix" and form="postfix" respectively, and the separators to be given the value form="infix".

Note that it would be incorrect to use mfenced with a separator of, for instance, '+', as an abbreviation for an expression using '+' as an ordinary operator, e.g.

```
<mrow>
  <mi>x</mi> <mo>+</mo> <mi>y</mi> <mo>+</mo> <mi>z</mi>
</mrow>
```

This is because the + signs would be treated as separators, not infix operators. That is, it would render as if they were marked up as <mo separator="true">+</mo>, which might therefore render inappropriately.

3.3.8.3 Examples

(a+b)

```
<mfenced>
  <mrow>
    <mi> a </mi>
    <mo> + </mo>
    <mi> b </mi>
  </mrow>
</mfenced>
```

Note that the above `mrow` is necessary so that the `mfenced` has just one argument. Without it, this would render incorrectly as '(a, +, b)'.

[0,1)

```
<mfenced open="[">
  <mn> 0 </mn>
  <mn> 1 </mn>
</mfenced>
```

$f(x,y)$

```
<mrow>
  <mi> f </mi>
  <mo> &ApplyFunction; </mo>
  <mfenced>
    <mi> x </mi>
    <mi> y </mi>
  </mfenced>
</mrow>
```

3.3.9 Enclose Expression Inside Notation `<menclose>`

3.3.9.1 Description

The `menclose` element renders its content inside the enclosing notation specified by its `notation` attribute. `menclose` accepts a single argument possibly being an inferred `mrow` of multiple children; see Section 3.1.3.

3.3.9.2 Attributes

`menclose` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

The values allowed for `notation` are open-ended. Conforming renderers may ignore any value they do not handle, although renderers are encouraged to render as many of the values listed below as possible.

Name	values	default
notation	(longdiv | actuarial | radical | box | roundedbox | circle | left | right | top | bottom | updiagonalstrike | downdiagonalstrike | verticalstrike | horizontalstrike) + | madruwb	longdiv
Specifies a space separated list of notations to be used to enclose the children. See below for a description of each type of notation.		

Any number of values can be given for `notation` separated by whitespace; all of those given and understood by a MathML renderer should be rendered. Each should be rendered as if the others were

not present; they should not nest one inside of the other. For example, `notation="circle box"` should result in circle and a box around the contents of `menclose`; the circle and box may overlap. This is shown in the first example below.

When `notation` has the value `"longdiv"`, the contents are drawn enclosed by a long division symbol. A complete example of long division is accomplished by also using `mtable` and `malign`. When `notation` is specified as `"actuarial"`, the contents are drawn enclosed by an actuarial symbol. A similar result can be achieved with the value `"top right"`. The case of `notation="radical"` is equivalent to the `msqrt` schema.

The values `"box"`, `"roundedbox"`, and `"circle"` should enclose the contents as indicated by the values. The amount of distance between the box, roundedbox, or circle, and the contents are not specified by MathML, and is left to the renderer. In practice, paddings on each side of 0.4em in the horizontal direction and .5ex in the vertical direction seem to work well.

The values `"left"`, `"right"`, `"top"` and `"bottom"` should result in lines drawn on those sides of the contents. The values `"updiagonalstrike"`, `"downdiagonalstrike"`, `"verticalstrike"` and `"horizontalstrike"` should result in the indicated strikeout lines being superimposed over the content of the `menclose`, e.g. a strikeout that extends from the lower left corner to the upper right corner of the `menclose` element for `"updiagonalstrike"`, etc.

The value `"madruwb"` should generate an enclosure representing an Arabic factorial (‘madruwb’ is the transliteration of the Arabic [ARABIC LETTER MEEM][ARABIC LETTER DAD][ARABIC LETTER REH][ARABIC LETTER WAW][ARABIC LETTER BEH] for factorial). This is shown in the third example below.

The baseline of an `menclose` element is the baseline of its child (which might be an implied `mrow`).

3.3.9.3 Examples

An example of using multiple attributes is

```
<menclose notation='circle box'>
  <mi> x </mi><mo> + </mo><mi> y </mi>
</menclose>
```

which renders with the box and circle overlapping roughly as  .

An example of using `menclose` for actuarial notation is

```
<msub>
  <mi>a</mi>
  <mrow>
    <menclose notation='actuarial'>
      <mi>n</mi>
    </menclose>
    <mo>&it;</mo>
    <mi>i</mi>
  </mrow>
</msub>
```

which renders roughly as

$$\overline{a}_{n|i}$$

An example of `"madruwb""` is:

```
<menclose notation="madruwb">
    <mn>12</mn>
  </menclose>
```

which renders roughly as 

.

3.4 Script and Limit Schemata

The elements described in this section position one or more scripts around a base. Attaching various kinds of scripts and embellishments to symbols is a very common notational device in mathematics. For purely visual layout, a single general-purpose element could suffice for positioning scripts and embellishments in any of the traditional script locations around a given base. However, in order to capture the abstract structure of common notation better, MathML provides several more specialized scripting elements.

In addition to sub/superscript elements, MathML has overscript and underscript elements that place scripts above and below the base. These elements can be used to place limits on large operators, or for placing accents and lines above or below the base. The rules for rendering accents differ from those for overscripts and underscripts, and this difference can be controlled with the `accent` and `accentunder` attributes, as described in the appropriate sections below.

Rendering of scripts is affected by the `scriptlevel` and `displaystyle` attributes, which are part of the environment inherited by the rendering process of every MathML expression, and are described in Section 3.1.6. These attributes cannot be given explicitly on a scripting element, but can be specified on the start tag of a surrounding `mstyle` element if desired.

MathML also provides an element for attachment of tensor indices. Tensor indices are distinct from ordinary subscripts and superscripts in that they must align in vertical columns. Tensor indices can also occur in prescript positions. Note that ordinary scripts follow the base (on the right in LTR context, but on the left in RTL context); prescripts precede the base (on the left (right) in LTR (RTL) context).

Because presentation elements should be used to describe the abstract notational structure of expressions, it is important that the base expression in all ‘scripting’ elements (i.e. the first argument expression) should be the entire expression that is being scripted, not just the trailing character. For example, $(x+y)^2$ should be written as:

```
<msup>
  <mrow>
    <mo> ( </mo>
    <mrow>
      <mi> x </mi>
      <mo> + </mo>
      <mi> y </mi>
    </mrow>
    <mo> ) </mo>
  </mrow>
  <mn> 2 </mn>
</msup>
```

3.4.1 Subscript <msub>

3.4.1.1 Description

The msub element attaches a subscript to a base using the syntax

<msub> *base* *subscript* </msub>

It increments scriptlevel by 1, and sets displaystyle to "false", within *subscript*, but leaves both attributes unchanged within *base*. (see Section 3.1.6.)

3.4.1.2 Attributes

msub elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
subscriptshift	*length*	*automatic*
Specifies the minimum amount to shift the baseline of *subscript* down; the default is for the rendering agent to use its own positioning rules.		

3.4.2 Superscript <msup>

3.4.2.1 Description

The msup element attaches a superscript to a base using the syntax

<msup> *base* *superscript* </msup>

It increments scriptlevel by 1, and sets displaystyle to "false", within *superscript*, but leaves both attributes unchanged within *base*. (see Section 3.1.6.)

3.4.2.2 Attributes

msup elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
superscriptshift	*length*	*automatic*
Specifies the minimum amount to shift the baseline of *superscript* up; the default is for the rendering agent to use its own positioning rules.		

3.4.3 Subscript-superscript Pair <msubsup>

3.4.3.1 Description

The msubsup element is used to attach both a subscript and superscript to a base expression.

<msubsup> *base* *subscript* *superscript* </msubsup>

It increments scriptlevel by 1, and sets displaystyle to "false", within *subscript* and *superscript*, but leaves both attributes unchanged within *base*. (see Section 3.1.6.)

Note that both scripts are positioned tight against the base as shown here x_1^2 versus the staggered positioning of nested scripts as shown here ${x_1}^2$; the latter can be achieved by nesting an msub inside an msup.

3.4.3.2 Attributes

`msubsup` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
subscriptshift	*length*	*automatic*
Specifies the minimum amount to shift the baseline of *subscript* down; the default is for the rendering agent to use its own positioning rules.		
superscriptshift	*length*	*automatic*
Specifies the minimum amount to shift the baseline of *superscript* up; the default is for the rendering agent to use its own positioning rules.		

3.4.3.3 Examples

The `msubsup` is most commonly used for adding sub/superscript pairs to identifiers as illustrated above. However, another important use is placing limits on certain large operators whose limits are traditionally displayed in the script positions even when rendered in display style. The most common of these is the integral. For example,

$$\int_0^1 \mathrm{e}^x \,\mathrm{d}x$$

would be represented as

```
<mrow>
  <msubsup>
    <mo> &int; </mo>
    <mn> 0 </mn>
    <mn> 1 </mn>
  </msubsup>
  <mrow>
    <msup>
      <mi> &ExponentialE; </mi>
      <mi> x </mi>
    </msup>
    <mo> &InvisibleTimes; </mo>
    <mrow>
      <mo> &DifferentialD; </mo>
      <mi> x </mi>
    </mrow>
  </mrow>
</mrow>
```

3.4.4 Underscript `<munder>`

3.4.4.1 Description

The `munder` element attaches an accent or limit placed under a base using the syntax

`<munder>` *base* *underscript* `</munder>`

It always sets `displaystyle` to `"false"` within the underscript, but increments `scriptlevel` by 1 only when `accentunder` is `"false"`. Within *base*, it always leaves both attributes unchanged. (see Section 3.1.6.)

If *base* is an operator with `movablelimits="true"` (or an embellished operator whose `mo` element core has `movablelimits="true"`), and `displaystyle="false"`, then *underscript* is drawn in a subscript position. In this case, the `accentunder` attribute is ignored. This is often used for limits on symbols such as `&sum;`.

3.4.4.2 Attributes

`munder` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
accentunder	true | false	*automatic*
Specfies whether *underscript* is drawn as an 'accent' or as a limit. An accent is drawn the same size as the base (without incrementing `scriptlevel`) and is drawn closer to the base.		
align	left | right | center	center
Specifies whether the script is aligned left, center, or right under/over the base.		

The default value of `accentunder` is false, unless *underscript* is an `mo` element or an embellished operator (see Section 3.2.5). If *underscript* is an `mo` element, the value of its `accent` attribute is used as the default value of `accentunder`. If *underscript* is an embellished operator, the `accent` attribute of the `mo` element at its core is used as the default value. As with all attributes, an explicitly given value overrides the default.

Here is an example (accent versus underscript): $\underbrace{x+y+z}$ versus $\underbrace{x+y+z}$. The MathML representation for this example is shown below.

3.4.4.3 Examples

The MathML representation for the example shown above is:

```
<mrow>
  <munder accentunder="true">
    <mrow>
      <mi> x </mi>
      <mo> + </mo>
      <mi> y </mi>
      <mo> + </mo>
      <mi> z </mi>
    </mrow>
    <mo> &UnderBrace; </mo>
  </munder>
  <mtext> versus </mtext>
  <munder accentunder="false">
    <mrow>
      <mi> x </mi>
      <mo> + </mo>
      <mi> y </mi>
      <mo> + </mo>
```

```
      <mi> z </mi>
    </mrow>
    <mo> &UnderBrace; </mo>
  </munder>
</mrow>
```

3.4.5 Overscript <mover>

3.4.5.1 Description

The mover element attaches an accent or limit placed over a base using the syntax

<mover> *base* *overscript* </mover>

It always sets displaystyle to "false" within overscript, but increments scriptlevel by 1 only when accent is "false". Within *base*, it always leaves both attributes unchanged. (see Section 3.1.6.)

If *base* is an operator with movablelimits="true" (or an embellished operator whose mo element core has movablelimits="true"), and displaystyle="false", then *overscript* is drawn in a superscript position. In this case, the accent attribute is ignored. This is often used for limits on symbols such as ∑.

3.4.5.2 Attributes

mover elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
accent	true | false	*automatic*
Specfies whether *overscript* is drawn as an 'accent' or as a limit. An accent is drawn the same size as the base (without incrementing scriptlevel) and is drawn closer to the base.		
align	left | right | center	center
Specifies whether the script is aligned left, center, or right under/over the base.		

The difference between an accent versus limit is shown here: $\hat{x}$ versus $\hat{x}$. These differences also apply to 'mathematical accents' such as bars or braces over expressions: $\overbrace{x+y+z}$ versus $\overbrace{x+y+z}$. The MathML representation for each of these examples is shown below.

The default value of *accent* is false, unless *overscript* is an mo element or an embellished operator (see Section 3.2.5). If *overscript* is an mo element, the value of its accent attribute is used as the default value of accent for mover. If *overscript* is an embellished operator, the accent attribute of the mo element at its core is used as the default value.

3.4.5.3 Examples

The MathML representation for the examples shown above is:

```
<mrow>
  <mover accent="true">
    <mi> x </mi>
    <mo> &Hat; </mo>
  </mover>
  <mtext> versus </mtext>
  <mover accent="false">
```

```
    <mi> x </mi>
    <mo> &Hat; </mo>
  </mover>
</mrow>
<mrow>
  <mover accent="true">
    <mrow>
      <mi> x </mi>
      <mo> + </mo>
      <mi> y </mi>
      <mo> + </mo>
      <mi> z </mi>
    </mrow>
    <mo> &OverBrace; </mo>
  </mover>
  <mtext> versus </mtext>
  <mover accent="false">
    <mrow>
      <mi> x </mi>
      <mo> + </mo>
      <mi> y </mi>
      <mo> + </mo>
      <mi> z </mi>
    </mrow>
    <mo> &OverBrace; </mo>
  </mover>
</mrow>
```

3.4.6 Underscript-overscript Pair `<munderover>`

3.4.6.1 Description

The `munderover` element attaches accents or limits placed both over and under a base using the syntax

`<munderover>` *base* *underscript* *overscript* `</munderover>`

It always sets `displaystyle` to `"false"` within *underscript* and *overscript*, but increments `scriptlevel` by 1 only when `accentunder` or `accent`, respectively, are `"false"`. Within *base*, it always leaves both attributes unchanged. (see Section 3.1.6).

If *base* is an operator with `movablelimits="true"` (or an embellished operator whose `mo` element core has `movablelimits="true"`), and `displaystyle="false"`, then *underscript* and *overscript* are drawn in a subscript and superscript position, respectively. In this case, the `accentunder` and `accent` attributes are ignored. This is often used for limits on symbols such as `&sum;`.

3.4.6.2 Attributes

`munderover` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
accent	true | false	*automatic*
Specfies whether *overscript* is drawn as an 'accent' or as a limit. An accent is drawn the same size as the base (without incrementing `scriptlevel`) and is drawn closer to the base.		
accentunder	true | false	*automatic*
Specfies whether *underscript* is drawn as an 'accent' or as a limit. An accent is drawn the same size as the base (without incrementing `scriptlevel`) and is drawn closer to the base.		
align	left | right | center	center
Specifies whether the scripts are aligned left, center, or right under/over the base.		

The `munderover` element is used so that the underscript and overscript are vertically spaced equally in relation to the base and so that they follow the slant of the base as in the second expression shown below:

$\int_0^\infty$ versus $\int_0^\infty$ The MathML representation for this example is shown below.

The difference in the vertical spacing is too small to be noticed on a low resolution display at a normal font size, but is noticeable on a higher resolution device such as a printer and when using large font sizes. In addition to the visual differences, attaching both the underscript and overscript to the same base more accurately reflects the semantics of the expression.

The defaults for `accent` and `accentunder` are computed in the same way as for `munder` and `mover`, respectively.

3.4.6.3 Examples

The MathML representation for the example shown above with the first expression made using separate `munder` and `mover` elements, and the second one using an `munderover` element, is:

```
<mrow>
  <mover>
    <munder>
      <mo> &int; </mo>
      <mn> 0 </mn>
    </munder>
    <mi> &infin; </mi>
  </mover>
  <mtext> versus </mtext>
  <munderover>
    <mo> &int; </mo>
    <mn> 0 </mn>
    <mi> &infin; </mi>
  </munderover>
</mrow>
```

3.4.7 Prescripts and Tensor Indices `<mmultiscripts>`

3.4.7.1 Description

Presubscripts and tensor notations are represented by a single element, `mmultiscripts`, using the syntax:

```
<mmultiscripts>
     base
     ( subscript superscript )*
     [ <mprescripts/> ( presubscript presuperscript )* ]
</mmultiscripts>
```

This element allows the representation of any number of vertically-aligned pairs of subscripts and superscripts, attached to one base expression. It supports both postscripts (to the right of the base in visual notation) and prescripts (to the left of the base in visual notation). Missing scripts can be represented by the empty element `none`.

The prescripts are optional, and when present are given *after* the postscripts, because prescripts are relatively rare compared to tensor notation.

The argument sequence consists of the base followed by zero or more pairs of vertically-aligned subscripts and superscripts (in that order) that represent all of the postscripts. This list is optionally followed by an empty element `mprescripts` and a list of zero or more pairs of vertically-aligned presubscripts and presuperscripts that represent all of the prescripts. The pair lists for postscripts and prescripts are given in the same order as the directional context (ie. left-to-right order in LTR context). If no subscript or superscript should be rendered in a given position, then the empty element `none` should be used in that position.

The base, subscripts, superscripts, the optional separator element `mprescripts`, the presubscripts, and the presuperscripts, are all direct sub-expressions of the `mmultiscripts` element, i.e. they are all at the same level of the expression tree. Whether a script argument is a subscript or a superscript, or whether it is a presubscript or a presuperscript is determined by whether it occurs in an even-numbered or odd-numbered argument position, respectively, ignoring the empty element `mprescripts` itself when determining the position. The first argument, the base, is considered to be in position 1. The total number of arguments must be odd, if `mprescripts` is not given, or even, if it is.

The empty elements `mprescripts` and `none` are only allowed as direct sub-expressions of `mmultiscripts`.

3.4.7.2 Attributes

Same as the attributes of `msubsup`. See Section 3.4.3.2.

The `mmultiscripts` element increments `scriptlevel` by 1, and sets `displaystyle` to `"false"`, within each of its arguments except *base*, but leaves both attributes unchanged within *base*. (see Section 3.1.6.)

3.4.7.3 Examples

Two examples of the use of `mmultiscripts` are:

${}_0F_1(;a;z)$.

```
<mrow>
  <mmultiscripts>
    <mi> F </mi>
    <mn> 1 </mn>
    <none/>
    <mprescripts/>
    <mn> 0 </mn>
```

```
    <none/>
  </mmultiscripts>
  <mo> &ApplyFunction; </mo>
  <mrow>
    <mo> ( </mo>
    <mrow>
      <mo> ; </mo>
      <mi> a </mi>
      <mo> ; </mo>
      <mi> z </mi>
    </mrow>
    <mo> ) </mo>
  </mrow>
</mrow>
```

$R_{i\,kl}^{\,j}$ (where k and l are different indices)

```
<mmultiscripts>
  <mi> R </mi>
  <mi> i </mi>
  <none/>
  <none/>
  <mi> j </mi>
  <mi> k </mi>
  <none/>
  <mi> l </mi>
  <none/>
</mmultiscripts>
```

An additional example of `mmultiscripts` shows how the binomial coefficient $\binom{5}{12}$ can be displayed in Arabic style ${}_{12}ل^{5}$

```
<mmultiscripts><mo>&#x0644;</mo>
    <mn>12</mn><none/>
    <mprescripts/>
    <none/><mn>5</mn>
  </mmultiscripts>
```

3.5 Tabular Math

Matrices, arrays and other table-like mathematical notation are marked up using `mtable`, `mtr`, `mlabeledtr` and `mtd` elements. These elements are similar to the `table`, `tr` and `td` elements of HTML, except that they provide specialized attributes for the fine layout control necessary for commutative diagrams, block matrices and so on.

While the two-dimensional layouts used for elementary math such as addition and multiplication are somewhat similar to tables, they differ in important ways. For layout and for accessibility reasons, the `mstack` and `mlongdiv` elements discussed in Section 3.6 should be used for elementary math notations.

In addition to the table elements mentioned above, the `mlabeledtr` element is used for labeling rows of a table. This is useful for numbered equations. The first child of `mlabeledtr` is the label. A label is somewhat special in that it is not considered an expression in the matrix and is not counted when determining the number of columns in that row.

3.5.1 Table or Matrix `<mtable>`

3.5.1.1 Description

A matrix or table is specified using the `mtable` element. Inside of the `mtable` element, only `mtr` or `mlabeledtr` elements may appear. (In MathML 1.x, the `mtable` was allowed to 'infer' `mtr` elements around its arguments, and the `mtr` element could infer `mtd` elements. This behaviour is deprecated.)

Table rows that have fewer columns than other rows of the same table (whether the other rows precede or follow them) are effectively padded on the right (or left in RTL context) with empty `mtd` elements so that the number of columns in each row equals the maximum number of columns in any row of the table. Note that the use of `mtd` elements with non-default values of the `rowspan` or `columnspan` attributes may affect the number of `mtd` elements that should be given in subsequent `mtr` elements to cover a given number of columns. Note also that the label in an `mlabeledtr` element is not considered a column in the table.

MathML does not specify a table layout algorithm. In particular, it is the responsibility of a MathML renderer to resolve conflicts between the `width` attribute and other constraints on the width of a table, such as explicit values for `columnwidth` attributes, and minimum sizes for table cell contents. For a discussion of table layout algorithms, see Cascading Style Sheets, level 2.

3.5.1.2 Attributes

`mtable` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
align	(top | bottom | center | baseline | axis) [*rownumber*]	axis
specifies the vertical alignment of the table with respect to its environment. `"axis"` means to align the vertical center of the table on the environment's axis. (The axis of an equation is an alignment line used by typesetters. It is the line on which a minus sign typically lies.) `"center"` and `"baseline"` both mean to align the center of the table on the environment's baseline. `"top"` or `"bottom"` aligns the top or bottom of the table on the environment's baseline. If the `align` attribute value ends with a *rownumber*, the specified row (counting from 1 for the top row) is aligned in the way described above, rather than the table as a whole; if *rownumber* is negative, it counts rows from the bottom. Other values of *rownumber* are illegal, but ignored		
rowalign	(top | bottom | center | baseline | axis) +	baseline
specifies the vertical alignment of the cells with respect to other cells within the same row: `"top"` aligns the tops of each entry across the row; `"bottom"` aligns the bottoms of the cells, `"center"` centers the cells; `"baseline"` aligns the baselines of the cells; `"axis"` aligns the axis of each cells. (See the note below about multiple values).		
columnalign	(left | center | right) +	center
specifies the horizontal alignment of the cells with respect to other cells within the same column: `"left"` aligns the left side of the cells; `"center"` centers each cells; `"right"` aligns the right side of the cells. (See the note below about multiple values).		
groupalign	*group-alignment-list-list*	left
[this attribute is described with the alignment elements, `maligngroup` and `malignmark`, in Section 3.5.5.]		
alignmentscope	(true | false) +	true
[this attribute is described with the alignment elements, `maligngroup` and `malignmark`, in Section 3.5.5.]		
columnwidth	(auto | *length* | fit) +	auto
specifies how wide a column should be: `"auto"` means that the column should be as wide as needed; an explicit length means that the column is exactly that wide and the contents of that column are made to fit by linewrapping or clipping at the discretion of the renderer; `"fit"` means that the page width remaining after subtracting the `"auto"` or fixed width columns is divided equally among the `"fit"` columns. If insufficient room remains to hold the contents of the `"fit"` columns, renderers may linewrap or clip the contents of the `"fit"` columns. Note that when the `columnwidth` is specified as a percentage, the value is relative to the width of the table, not as a percentage of the default (which is `"auto"`). That is, a renderer should try to adjust the width of the column so that it covers the specified percentage of the entire table width. (See the note below about multiple values).		
width	auto | *length*	auto
specifies the desired width of the entire table and is intended for visual user agents. When the value is a percentage value, the value is relative to the horizontal space a MathML renderer has available for the `math` element. When the value is `"auto"`, the MathML renderer should calculate the table width from its contents using whatever layout algorithm it chooses.		
rowspacing	(*length*) +	1.0ex
specifies how much space to add between rows. (See the note below about multiple values).		
columnspacing	(*length*) +	0.8em
specifies how much space to add between rows. (See the note below about multiple values).		
rowlines	(none | solid | dashed) +	none
specifies whether and what kind of lines should be added between each row: `"none"` means no lines; `"solid"` means solid lines; `"dashed"` means dashed lines		

In the above specifications for attributes affecting rows (respectively, columns, or the gaps between rows or columns), the notation `(...)+` means that multiple values can be given for the attribute as a space separated list (see Section 2.1.5). In this context, a single value specifies the value to be used for all rows (resp., columns or gaps). A list of values are taken to apply to corresponding rows (resp., columns or gaps) starting from the top (resp., left or gap after the first row or column). If there are more rows (resp., columns or gaps) than supplied values, the last value is repeated as needed. If there are too many values supplied, the excess are ignored.

Note that none of the spaces occupied by lines `frame`, `rowlines` and `columnlines`, nor the spacing `framespacing`, `rowspacing` or `columnspacing`, nor the label in `mlabeledtr` are counted as rows or columns.

3.5.1.3 Examples

A 3 by 3 identity matrix could be represented as follows:

```
<mrow>
  <mo> ( </mo>
  <mtable>
    <mtr>
      <mtd> <mn>1</mn> </mtd>
      <mtd> <mn>0</mn> </mtd>
      <mtd> <mn>0</mn> </mtd>
    </mtr>
    <mtr>
      <mtd> <mn>0</mn> </mtd>
      <mtd> <mn>1</mn> </mtd>
      <mtd> <mn>0</mn> </mtd>
    </mtr>
    <mtr>
      <mtd> <mn>0</mn> </mtd>
      <mtd> <mn>0</mn> </mtd>
      <mtd> <mn>1</mn> </mtd>
    </mtr>
  </mtable>
  <mo> ) </mo>
</mrow>
```

This might be rendered as:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note that the parentheses must be represented explicitly; they are not part of the `mtable` element's rendering. This allows use of other surrounding fences, such as brackets, or none at all.

3.5.2 Row in Table or Matrix `<mtr>`

3.5.2.1 Description

An `mtr` element represents one row in a table or matrix. An `mtr` element is only allowed as a direct sub-expression of an `mtable` element, and specifies that its contents should form one row of the table.

Each argument of mtr is placed in a different column of the table, starting at the leftmost column in a LTR context or rightmost column in a RTL context.

As described in Section 3.5.1, mtr elements are effectively padded on the right with mtd elements when they are shorter than other rows in a table.

3.5.2.2 Attributes

mtr elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
rowalign	top | bottom | center | baseline | axis	*inherited*
overrides, for this row, the vertical alignment of cells specified by the rowalign attribute on the mtable.		
columnalign	(left | center | right) +	*inherited*
overrides, for this row, the horizontal alignment of cells specified by the columnalign attribute on the mtable.		
groupalign	*group-alignment-list-list*	*inherited*
[this attribute is described with the alignment elements, maligngroup and malignmark, in Section 3.5.5.]		

3.5.3 Labeled Row in Table or Matrix <mlabeledtr>

3.5.3.1 Description

An mlabeledtr element represents one row in a table that has a label on either the left or right side, as determined by the side attribute. The label is the first child of mlabeledtr. The rest of the children represent the contents of the row and are identical to those used for mtr; all of the children except the first must be mtd elements.

An mlabeledtr element is only allowed as a direct sub-expression of an mtable element. Each argument of mlabeledtr except for the first argument (the label) is placed in a different column of the table, starting at the leftmost column.

Note that the label element is not considered to be a cell in the table row. In particular, the label element is not taken into consideration in the table layout for purposes of width and alignment calculations. For example, in the case of an mlabeledtr with a label and a single centered mtd child, the child is first centered in the enclosing mtable, and then the label is placed. Specifically, the child is *not* centered in the space that remains in the table after placing the label.

While MathML does not specify an algorithm for placing labels, implementors of visual renderers may find the following formatting model useful. To place a label, an implementor might think in terms of creating a larger table, with an extra column on both ends. The columnwidth attributes of both these border columns would be set to "fit" so that they expand to fill whatever space remains after the inner columns have been laid out. Finally, depending on the values of side and minlabelspacing, the label is placed in whatever border column is appropriate, possibly shifted down if necessary, and aligned according to columnalignment.

3.5.3.2 Attributes

The attributes for mlabeledtr are the same as for mtr. Unlike the attributes for the mtable element, attributes of mlabeledtr that apply to column elements also apply to the label. For example, in a one column table,

`<mlabeledtr rowalign='top'>`

means that the label and other entries in the row are vertically aligned along their top. To force a particular alignment on the label, the appropriate attribute would normally be set on the `mtd` start tag that surrounds the label content.

3.5.3.3 Equation Numbering

One of the important uses of `mlabeledtr` is for numbered equations. In a `mlabeledtr`, the label represents the equation number and the elements in the row are the equation being numbered. The `side` and `minlabelspacing` attributes of `mtable` determine the placement of the equation number.

In larger documents with many numbered equations, automatic numbering becomes important. While automatic equation numbering and automatically resolving references to equation numbers is outside the scope of MathML, these problems can be addressed by the use of style sheets or other means. The mlabeledtr construction provides support for both of these functions in a way that is intended to facilitate XSLT processing. The `mlabeledtr` element can be used to indicate the presence of a numbered equation, and the first child can be changed to the current equation number, along with incrementing the global equation number. For cross references, an `id` on either the mlabeledtr element or on the first element itself could be used as a target of any link.

```
<mtable>
  <mlabeledtr id='e-is-m-c-square'>
    <mtd>
      <mtext> (2.1) </mtext>
    </mtd>
    <mtd>
     <mrow>
       <mi>E</mi>
       <mo>=</mo>
       <mrow>
        <mi>m</mi>
        <mo>&it;</mo>
        <msup>
         <mi>c</mi>
         <mn>2</mn>
        </msup>
       </mrow>
     </mrow>
    </mtd>
  </mlabeledtr>
</mtable>
```

This should be rendered as:

$$E = mc^2 \tag{2.1}$$

3.5.4 Entry in Table or Matrix `<mtd>`

3.5.4.1 Description

An `mtd` element represents one entry, or cell, in a table or matrix. An `mtd` element is only allowed as a direct sub-expression of an `mtr` or an `mlabeledtr` element.

The mtd element accepts a single argument possibly being an inferred mrow of multiple children; see Section 3.1.3.

3.5.4.2 Attributes

mtd elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
rowspan	*positive-integer*	1
causes the cell to be treated as if it occupied the number of rows specified. The corresponding td in the following "rowspan"-1 rows must be omitted. The interpretation corresponds with the similar attributes for HTML 4.01 tables.		
columnspan	*positive-integer*	1
causes the cell to be treated as if it occupied the number of columns specified. The following "rowspan"-1 tds must be omitted. The interpretation corresponds with the similar attributes for HTML 4.01 tables.		
rowalign	top | bottom | center | baseline | axis	*inherited*
specifies the vertical alignment of this cell, overriding any value specified on the containing mrow and mtable. See the rowalign attribute of mtable.		
columnalign	left | center | right	*inherited*
specifies the horizontal alignment of this cell, overriding any value specified on the containing mrow and mtable. See the columnalign attribute of mtable.		
groupalign	*group-alignment-list*	*inherited*
[this attribute is described with the alignment elements, maligngroup and malignmark, in Section 3.5.5.]		

The rowspan and columnspan attributes can be used around an mtd element that represents the label in a mlabeledtr element. Also, the label of a mlabeledtr element is not considered to be part of a previous rowspan and columnspan.

3.5.5 Alignment Markers <maligngroup>, <malignmark>

3.5.5.1 Description

Alignment markers are space-like elements (see Section 3.2.7) that can be used to vertically align specified points within a column of MathML expressions by the automatic insertion of the necessary amount of horizontal space between specified sub-expressions.

The discussion that follows will use the example of a set of simultaneous equations that should be rendered with vertical alignment of the coefficients and variables of each term, by inserting spacing somewhat like that shown here:

```
8.44x + 55  y =  0
    3.1 x -  0.7y = -1.1
```

If the example expressions shown above were arranged in a column but not aligned, they would appear as:

```
8.44x + 55y = 0
    3.1x - 0.7y = -1.1
```

For audio renderers, it is suggested that the alignment elements produce the analogous behavior of altering the rhythm of pronunciation so that it is the same for several sub-expressions in a column, by the insertion of the appropriate time delays in place of the extra horizontal spacing described here.

The expressions whose parts are to be aligned (each equation, in the example above) must be given as the table elements (i.e. as the `mtd` elements) of one column of an `mtable`. To avoid confusion, the term 'table cell' rather than 'table element' will be used in the remainder of this section.

All interactions between alignment elements are limited to the `mtable` column they arise in. That is, every column of a table specified by an `mtable` element acts as an 'alignment scope' that contains within it all alignment effects arising from its contents. It also excludes any interaction between its own alignment elements and the alignment elements inside any nested alignment scopes it might contain.

The reason `mtable` columns are used as alignment scopes is that they are the only general way in MathML to arrange expressions into vertical columns. Future versions of MathML may provide an `malignscope` element that allows an alignment scope to be created around any MathML element, but even then, table columns would still sometimes need to act as alignment scopes, and since they are not elements themselves, but rather are made from corresponding parts of the content of several `mtr` elements, they could not individually be the content of an alignment scope element.

An `mtable` element can be given the attribute `alignmentscope="false"` to cause its columns not to act as alignment scopes. This is discussed further at the end of this section. Otherwise, the discussion in this section assumes that this attribute has its default value of `"true"`.

3.5.5.2 *Specifying alignment groups*

To cause alignment, it is necessary to specify, within each expression to be aligned, the points to be aligned with corresponding points in other expressions, and the beginning of each *alignment group* of sub-expressions that can be horizontally shifted as a unit to effect the alignment. Each alignment group must contain one alignment point. It is also necessary to specify which expressions in the column have no alignment groups at all, but are affected only by the ordinary column alignment for that column of the table, i.e. by the `columnalign` attribute, described elsewhere.

The alignment groups start at the locations of invisible `maligngroup` elements, which are rendered with zero width when they occur outside of an alignment scope, but within an alignment scope are rendered with just enough horizontal space to cause the desired alignment of the alignment group that follows them. A simple algorithm by which a MathML application can achieve this is given later. In the example above, each equation would have one `maligngroup` element before each coefficient, variable, and operator on the left-hand side, one before the = sign, and one before the constant on the right-hand side.

In general, a table cell containing n `maligngroup` elements contains n alignment groups, with the ith group consisting of the elements entirely after the ith `maligngroup` element and before the $(i+1)$-th; no element within the table cell's content should occur entirely before its first `maligngroup` element.

Note that the division into alignment groups does *not* necessarily fit the nested expression structure of the MathML expression containing the groups — that is, it is permissible for one alignment group to consist of the end of one `mrow`, all of another one, and the beginning of a third one, for example. This can be seen in the MathML markup for the present example, given at the end of this section.

The nested expression structure formed by `mrows` and other layout schemata should reflect the mathematical structure of the expression, not the alignment-group structure, to make possible optimal renderings and better automatic interpretations; see the discussion of proper grouping in section Section 3.3.1. Insertion of alignment elements (or other space-like elements) should not alter the correspondence between the structure of a MathML expression and the structure of the mathematical expression it represents.

Although alignment groups need not coincide with the nested expression structure of layout schemata, there are nonetheless restrictions on where an `maligngroup` element is allowed within a table cell. The

`maligngroup` element may only be contained within elements (directly or indirectly) of the following types (which are themselves contained in the table cell):

- an `mrow` element, including an inferred `mrow` such as the one formed by a multi-child `mtd` element;
- an `mstyle` element;
- an `mphantom` element;
- an `mfenced` element;
- an `maction` element, though only its selected sub-expression is checked;
- a `semantics` element.

These restrictions are intended to ensure that alignment can be unambiguously specified, while avoiding complexities involving things like overscripts, radical signs and fraction bars. They also ensure that a simple algorithm suffices to accomplish the desired alignment.

Note that some positions for an `maligngroup` element, although legal, are not useful, such as for an `maligngroup` element to be an argument of an `mfenced` element. When inserting an `maligngroup` element before a given element in pre-existing MathML, it will often be necessary, and always acceptable, to form a new `mrow` element to contain just the `maligngroup` element and the element it is inserted before. In general, this will be necessary except when the `maligngroup` element is inserted directly into an `mrow` or into an element that can form an inferred `mrow` from its contents. See the warning about the legal grouping of 'space-like elements' in Section 3.2.7.

For the table cells that are divided into alignment groups, every element in their content must be part of exactly one alignment group, except the elements from the above list that contain `maligngroup` elements inside them, and the `maligngroup` elements themselves. This means that, within any table cell containing alignment groups, the first complete element must be an `maligngroup` element, though this may be preceded by the start tags of other elements.

This requirement removes a potential confusion about how to align elements before the first `maligngroup` element, and makes it easy to identify table cells that are left out of their column's alignment process entirely.

Note that it is not required that the table cells in a column that are divided into alignment groups each contain the same number of groups. If they don't, zero-width alignment groups are effectively added on the right side of each table cell that has fewer groups than other table cells in the same column.

3.5.5.3 Table cells that are not divided into alignment groups

Expressions in a column that are to have no alignment groups should contain no `maligngroup` elements. Expressions with no alignment groups are aligned using only the `columnalign` attribute that applies to the table column as a whole, and are not affected by the `groupalign` attribute described below. If such an expression is wider than the column width needed for the table cells containing alignment groups, all the table cells containing alignment groups will be shifted as a unit within the column as described by the `columnalign` attribute for that column. For example, a column heading with no internal alignment could be added to the column of two equations given above by preceding them with another table row containing an `mtext` element for the heading, and using the default `columnalign="center"` for the table, to produce:

```
equations with aligned variables
      8.44x + 55  y =  0
      3.1 x -  0.7y = -1.1
```

or, with a shorter heading,

```
some equations
8.44x + 55  y =  0
3.1 x -  0.7y = -1.1
```

3.5.5.4 Specifying alignment points using <malignmark>

Each alignment group's alignment point can either be specified by an malignmark element anywhere within the alignment group (except within another alignment scope wholly contained inside it), or it is determined automatically from the groupalign attribute. The groupalign attribute can be specified on the group's preceding maligngroup element or on its surrounding mtd, mtr, or mtable elements. In typical cases, using the groupalign attribute is sufficient to describe the desired alignment points, so no malignmark elements need to be provided.

The malignmark element indicates that the alignment point should occur on the right edge of the preceding element, or the left edge of the following element or character, depending on the edge attribute of malignmark. Note that it may be necessary to introduce an mrow to group an malignmark element with a neighboring element, in order not to alter the argument count of the containing element. (See the warning about the legal grouping of 'space-like elements' in Section 3.2.7).

When an malignmark element is provided within an alignment group, it can occur in an arbitrarily deeply nested element within the group, as long as it is not within a nested alignment scope. It is not subject to the same restrictions on location as maligngroup elements. However, its immediate surroundings need to be such that the element to its immediate right or left (depending on its edge attribute) can be unambiguously identified. If no such element is present, renderers should behave as if a zero-width element had been inserted there.

For the purposes of alignment, an element X is considered to be to the immediate left of an element Y, and Y to the immediate right of X, whenever X and Y are successive arguments of one (possibly inferred) mrow element, with X coming before Y. In the case of mfenced elements, MathML applications should evaluate this relation as if the mfenced element had been replaced by the equivalent expanded form involving mrow. Similarly, an maction element should be treated as if it were replaced by its currently selected sub-expression. In all other cases, no relation of 'to the immediate left or right' is defined for two elements X and Y. However, in the case of content elements interspersed in presentation markup, MathML applications should attempt to evaluate this relation in a sensible way. For example, if a renderer maintains an internal presentation structure for rendering content elements, the relation could be evaluated with respect to that. (See Chapter 4 and Chapter 5 for further details about mixing presentation and content markup.)

malignmark elements are allowed to occur within the content of token elements, such as mn, mi, or mtext. When this occurs, the character immediately before or after the malignmark element will carry the alignment point; in all other cases, the element to its immediate left or right will carry the alignment point. The rationale for this is that it is sometimes desirable to align on the edges of specific characters within multi-character token elements.

If there is more than one malignmark element in an alignment group, all but the first one will be ignored. MathML applications may wish to provide a mode in which they will warn about this situation, but it is not an error, and should trigger no warnings by default. The rationale for this is that it would be inconvenient to have to remove all unnecessary malignmark elements from automatically generated data, in certain cases, such as when they are used to specify alignment on 'decimal points' other than the '.' character.

3.5.5.5 <malignmark> Attributes

malignmark elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
edge	left | right	left
see the discussion below.		

malignmark has one attribute, edge, which specifies whether the alignment point will be found on the left or right edge of some element or character. The precise location meant by 'left edge' or 'right edge' is discussed below. If edge="right", the alignment point is the right edge of the element or character to the immediate left of the malignmark element. If edge="left", the alignment point is the left edge of the element or character to the immediate right of the malignmark element. Note that the attribute refers to the choice of edge rather than to the direction in which to look for the element whose edge will be used.

For malignmark elements that occur within the content of MathML token elements, the preceding or following character in the token element's content is used; if there is no such character, a zero-width character is effectively inserted for the purpose of carrying the alignment point on its edge. For all other malignmark elements, the preceding or following element is used; if there is no such element, a zero-width element is effectively inserted to carry the alignment point.

The precise definition of the 'left edge' or 'right edge' of a character or glyph (e.g. whether it should coincide with an edge of the character's bounding box) is not specified by MathML, but is at the discretion of the renderer; the renderer is allowed to let the edge position depend on the character's context as well as on the character itself.

For proper alignment of columns of numbers (using groupalign values of "left", "right", or "decimalpoint"), it is likely to be desirable for the effective width (i.e. the distance between the left and right edges) of decimal digits to be constant, even if their bounding box widths are not constant (e.g. if '1' is narrower than other digits). For other characters, such as letters and operators, it may be desirable for the aligned edges to coincide with the bounding box.

The 'left edge' of a MathML element or alignment group refers to the left edge of the leftmost glyph drawn to render the element or group, except that explicit space represented by mspace or mtext elements should also count as 'glyphs' in this context, as should glyphs that would be drawn if not for mphantom elements around them. The 'right edge' of an element or alignment group is defined similarly.

3.5.5.6 <maligngroup> Attributes

maligngroup elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
groupalign	left | center | right | decimalpoint	*inherited*
see the discussion below.		

maligngroup has one attribute, groupalign, which is used to determine the position of its group's alignment point when no malignmark element is present. The following discussion assumes that no malignmark element is found within a group.

In the example given at the beginning of this section, there is one column of 2 table cells, with 7 alignment groups in each table cell; thus there are 7 columns of alignment groups, with 2 groups, one above the other, in each column. These columns of alignment groups should be given the 7 groupalign values 'decimalpoint left left decimalpoint left left decimalpoint', in that order. How to specify this list

of values for a table cell or table column as a whole, using attributes on elements surrounding the `maligngroup` element is described later.

If `groupalign` is 'left', 'right', or 'center', the alignment point is defined to be at the group's left edge, at its right edge, or halfway between these edges, respectively. The meanings of 'left edge' and 'right edge' are as discussed above in relation to `malignmark`.

If `groupalign` is 'decimalpoint', the alignment point is the right edge of the last character before the decimal point. The decimal point is the first '.' character (ASCII 0x2e) in the first `mn` element found along the alignment group's baseline. More precisely, the alignment group is scanned recursively, depth-first, for the first `mn` element, descending into all arguments of each element of the types `mrow` (including inferred `mrows`), `mstyle`, `mpadded`, `mphantom`, `menclose`, `mfenced`, or `msqrt`, descending into only the first argument of each 'scripting' element (`msub`, `msup`, `msubsup`, `munder`, `mover`, `munderover`, `mmultiscripts`) or of each `mroot` or `semantics` element, descending into only the selected sub-expression of each `maction` element, and skipping the content of all other elements. The first `mn` so found always contains the alignment point, which is the right edge of the last character before the first decimal point in the content of the `mn` element. If there is no decimal point in the `mn` element, the alignment point is the right edge of the last character in the content. If the decimal point is the first character of the `mn` element's content, the right edge of a zero-width character inserted before the decimal point is used. If no `mn` element is found, the right edge of the entire alignment group is used (as for `groupalign="right"`).

In order to permit alignment on decimal points in `cn` elements, a MathML application can convert a content expression into a presentation expression that renders the same way before searching for decimal points as described above.

If characters other than '.' should be used as 'decimal points' for alignment, they should be preceded by `malignmark` elements within the `mn` token's content itself.

For any of the `groupalign` values, if an explicit `malignmark` element is present anywhere within the group, the position it specifies (described earlier) overrides the automatic determination of alignment point from the `groupalign` value.

3.5.5.7 *Inheritance of `groupalign` values*

It is not usually necessary to put a `groupalign` attribute on every `maligngroup` element. Since this attribute is usually the same for every group in a column of alignment groups to be aligned, it can be inherited from an attribute on the `mtable` that was used to set up the alignment scope as a whole, or from the `mtr` or `mtd` elements surrounding the alignment group. It is inherited via an 'inheritance path' that proceeds from `mtable` through successively contained `mtr`, `mtd`, and `maligngroup` elements. There is exactly one element of each of these kinds in this path from an `mtable` to any alignment group inside it. In general, the value of `groupalign` will be inherited by any given alignment group from the innermost element that surrounds the alignment group and provides an explicit setting for this attribute. For example, if an `mtable` element specifies values for `groupalign` and a `maligngroup` element within the table also specifies an explicit `groupalign` value, then then the value from the `maligngroup` takes priority.

Note, however, that each `mtd` element needs, in general, a list of `groupalign` values, one for each `maligngroup` element inside it, rather than just a single value. Furthermore, an `mtr` or `mtable` element needs, in general, a list of lists of `groupalign` values, since it spans multiple `mtable` columns, each potentially acting as an alignment scope. Such lists of *group-alignment* values are specified using the following syntax rules:

```
group-alignment             := left | right | center | decimalpoint
group-alignment-list        :=  group-alignment  +
group-alignment-list-list := ( '{'  group-alignment-list  '}' ) +
```

As described in Section 2.1.5, | separates alternatives; + represents optional repetition (i.e. 1 or more copies of what precedes it), with extra values ignored and the last value repeated if necessary to cover additional table columns or alignment group columns; '' and '' represent literal braces; and (and) are used for grouping, but do not literally appear in the attribute value.

The permissible values of the groupalign attribute of the elements that have this attribute are specified using the above syntax definitions as follows:

Element type	groupalign attribute syntax	default value
mtable	*group-alignment-list-list*	left
mtr	*group-alignment-list-list*	*inherited from* mtable *attribute*
mlabeledtr	*group-alignment-list-list*	*inherited from* mtable *attribute*
mtd	*group-alignment-list*	*inherited from within* mtr *attribute*
maligngroup	*group-alignment*	*inherited from within* mtd *attribute*

In the example near the beginning of this section, the group alignment values could be specified on every mtd element using groupalign = 'decimalpoint left left decimalpoint left left decimalpoint', or on every mtr element using groupalign = 'decimalpoint left left decimalpoint left left decimalpoint', or (most conveniently) on the mtable as a whole using groupalign = 'decimalpoint left left decimalpoint left left decimalpoint', which provides a single braced list of *group-alignment* values for the single column of expressions to be aligned.

3.5.5.8 MathML representation of an alignment example

The above rules are sufficient to explain the MathML representation of the example given near the start of this section. To repeat the example, the desired rendering is:

```
8.44x + 55  y =  0
    3.1 x -  0.7y = -1.1
```

One way to represent that in MathML is:

```
<mtable groupalign="{decimalpoint left left decimalpoint left left decimalpoint}">
  <mtr>
    <mtd>
      <mrow>
        <mrow>
          <mrow>
            <maligngroup/>
            <mn> 8.44 </mn>
            <mo> &InvisibleTimes; </mo>
            <maligngroup/>
            <mi> x </mi>
          </mrow>
          <maligngroup/>
          <mo> + </mo>
          <mrow>
            <maligngroup/>
            <mn> 55 </mn>
            <mo> &InvisibleTimes; </mo>
            <maligngroup/>
```

```
            <mi> y </mi>
          </mrow>
        </mrow>
      <malignmark/>
      <mo> = </mo>
      <malignmark/>
      <mn> 0 </mn>
    </mrow>
    </mtd>
  </mtr>
  <mtr>
    <mtd>
      <mrow>
        <mrow>
          <mrow>
            <malignmark/>
            <mn> 3.1 </mn>
            <mo> &InvisibleTimes; </mo>
            <malignmark/>
            <mi> x </mi>
          </mrow>
          <malignmark/>
          <mo> - </mo>
          <mrow>
            <malignmark/>
            <mn> 0.7 </mn>
            <mo> &InvisibleTimes; </mo>
            <malignmark/>
            <mi> y </mi>
          </mrow>
        </mrow>
        <malignmark/>
        <mo> = </mo>
        <malignmark/>
        <mrow>
          <mo> - </mo>
          <mn> 1.1 </mn>
        </mrow>
      </mrow>
    </mtd>
  </mtr>
</mtable>
```

3.5.5.9 Further details of alignment elements

The alignment elements `malignmark` and `malignmark` can occur outside of alignment scopes, where they are ignored. The rationale behind this is that in situations in which MathML is generated, or copied from another document, without knowing whether it will be placed inside an alignment scope, it would be inconvenient for this to be an error.

An `mtable` element can be given the attribute `alignmentscope="false"` to cause its columns not

to act as alignment scopes. In general, this attribute has the syntax `(true | false) +`; if its value is a list of boolean values, each boolean value applies to one column, with the last value repeated if necessary to cover additional columns, or with extra values ignored. Columns that are not alignment scopes are part of the alignment scope surrounding the `mtable` element, if there is one. Use of `alignmentscope="false"` allows nested tables to contain `malignmark` elements for aligning the inner table in the surrounding alignment scope.

As discussed above, processing of alignment for content elements is not well-defined, since MathML does not specify how content elements should be rendered. However, many MathML applications are likely to find it convenient to internally convert content elements to presentation elements that render the same way. Thus, as a general rule, even if a renderer does not perform such conversions internally, it is recommended that the alignment elements should be processed as if it did perform them.

A particularly important case for renderers to handle gracefully is the interaction of alignment elements with the `matrix` content element, since this element may or may not be internally converted to an expression containing an `mtable` element for rendering. To partially resolve this ambiguity, it is suggested, but not required, that if the `matrix` element is converted to an expression involving an `mtable` element, that the `mtable` element be given the attribute `alignmentscope="false"`, which will make the interaction of the `matrix` element with the alignment elements no different than that of a generic presentation element (in particular, it will allow it to contain `malignmark` elements that operate within the alignment scopes created by the columns of an `mtable` that contains the `matrix` element in one of its table cells).

The effect of alignment elements within table cells that have non-default values of the `columnspan` or `rowspan` attributes is not specified, except that such use of alignment elements is not an error. Future versions of MathML may specify the behavior of alignment elements in such table cells.

The effect of possible linebreaking of an `mtable` element on the alignment elements is not specified.

3.5.5.10 A simple alignment algorithm

A simple algorithm by which a MathML application can perform the alignment specified in this section is given here. Since the alignment specification is deterministic (except for the definition of the left and right edges of a character), any correct MathML alignment algorithm will have the same behavior as this one. Each `mtable` column (alignment scope) can be treated independently; the algorithm given here applies to one `mtable` column, and takes into account the alignment elements, the `groupalign` attribute described in this section, and the `columnalign` attribute described under `mtable` (Section 3.5.1).

First, a rendering is computed for the contents of each table cell in the column, using zero width for all `maligngroup` and `malignmark` elements. The final rendering will be identical except for horizontal shifts applied to each alignment group and/or table cell. The positions of alignment points specified by any `malignmark` elements are noted, and the remaining alignment points are determined using `groupalign` values.

For each alignment group, the horizontal positions of the left edge, alignment point, and right edge are noted, allowing the width of the group on each side of the alignment point (left and right) to be determined. The sum of these two 'side-widths', i.e. the sum of the widths to the left and right of the alignment point, will equal the width of the alignment group.

Second, each column of alignment groups, from left to right, is scanned. The ith scan covers the ith alignment group in each table cell containing any alignment groups. Table cells with no alignment groups, or with fewer than i alignment groups, are ignored. Each scan computes two maximums over

the alignment groups scanned: the maximum width to the left of the alignment point, and the maximum width to the right of the alignment point, of any alignment group scanned.

The sum of all the maximum widths computed (two for each column of alignment groups) gives one total width, which will be the width of each table cell containing alignment groups. Call the maximum number of alignment groups in one cell n; each such cell is divided into $2n$ horizontally adjacent sections, called L(i) and R(i) for i from 1 to n, using the $2n$ maximum side-widths computed above; for each i, the width of all sections called L(i) is the maximum width of any cell's ith alignment group to the left of its alignment point, and the width of all sections called R(i) is the maximum width of any cell's ith alignment group to the right of its alignment point.

Each alignment group is then shifted horizontally as a block to unique position that places: in the section called L(i) that part of the ith group to the left of its alignment point; in the section called R(i) that part of the ith group to the right of its alignment point. This results in the alignment point of each ith group being on the boundary between adjacent sections L(i) and R(i), so that all alignment points of ith groups have the same horizontal position.

The widths of the table cells that contain no alignment groups were computed as part of the initial rendering, and may be different for each cell, and different from the single width used for cells containing alignment groups. The maximum of all the cell widths (for both kinds of cells) gives the width of the table column as a whole.

The position of each cell in the column is determined by the applicable part of the value of the `columnalign` attribute of the innermost surrounding `mtable`, `mtr`, or `mtd` element that has an explicit value for it, as described in the sections on those elements. This may mean that the cells containing alignment groups will be shifted within their column, in addition to their alignment groups having been shifted within the cells as described above, but since each such cell has the same width, it will be shifted the same amount within the column, thus maintaining the vertical alignment of the alignment points of the corresponding alignment groups in each cell.

3.6 Elementary Math

Mathematics used in the lower grades such as two-dimensional addition, multiplication, and long division tends to be tabular in nature. However, the specific notations used varies among countries much more than for higher level math. Furthermore, elementary math often presents examples in some intermediate state and MathML must be able to capture these intermediate or intentionally missing partial forms. Indeed, these constructs represent memory aids or procedural guides, as much as they represent 'mathematics'.

The elements used for basic alignments in elementary math are: `mstack`, for aligning rows of digits and operators; `msgroup`, for grouping rows with similar alignment; `msrow`, for grouping digits and operators into a row; and `msline`, for drawing lines between the rows of the stack. Carries are supported by `mscarry`, with `mscarries` used for associating a set of carries with a row. Long division, `mlongdiv`, composes an `mstack` with a divisor and quotient. `mstack` and `mlongdiv` are the parent elements for all elementary math layout.

Since the primary use of these stacking constructs is to stack rows of numbers aligned on their digits, and since numbers are always formatted left-to-right, the columns of an mstack are always processed left-to-right; the overall directionality in effect (ie. the `dir` attribute) does not affect to the ordering of display of columns or carries in rows and, in particular, does not affect the ordering of any operators within a row (See Section 3.1.5).

These elements are described in this section followed by examples of their use. In addition to two-dimensional addition, subtraction, multiplication, and long division, these elements can be used to represent several notations used for repeating decimals.

A very simple example of two-dimensional addition is shown below:

$$\begin{array}{r} 424 \\ +33 \\ \hline \end{array}$$

The MathML for this is:

```
<mstack>
  <mn>424</mn>
  <msrow> <mo>+</mo> <mn>33</mn> </msrow>
  <msline/>
</mstack>
```

Many more examples are given in Section 3.6.8.

3.6.1 Stacks of Characters `<mstack>`

3.6.1.1 Description

`mstack` is used to lay out rows of numbers that are aligned on each digit. This is common in many elementary math notations such as 2D addition, subtraction, and multiplication.

The children of an `mstack` represent rows, or groups of them, to be stacked each below the previous row; there can be any number of rows. An `msrow` represents a row; an `msgroup` groups a set of rows together so that their horizontal alignment can be adjusted together; an `mscarries` represents a set of carries to be applied to the following row; an `msline` represents a line separating rows. Any other element is treated as if implicitly surrounded by `msrow`.

Each row contains 'digits' that are placed into columns. (see Section 3.6.4 for further details). The `stackalign` attribute together with the `position` and `shift` attributes of `msgroup`, `mscarries`, and `msrow` determine to which column a character belongs.

The width of a column is the maximum of the widths of each 'digit' in that column — carries do *not* participate in the width calculation; they are treated as having zero width. If an element is too wide to fit into a column, it overflows into the adjacent column(s) as determined by the `charalign` attribute. If there is no character in a column, its width is taken to be the width of a 0 in the current language (in many fonts, all digits have the same width).

The method for laying out an mstack is:

1. The 'digits' in a row are determined.
2. All of the digits in a row are initially aligned according to the `stackalign` value.
3. Each row is positioned relative to that alignment based on the `position` attribute (if any) that controls that controls that row.
4. The maximumn width of the digits in a column are determined and shorter and wider entries in that column are aligned according to the `charalign` attribute.
5. The width and height of the mstack element are computed based on the rows and columns. Any overflow from a column is *not* used as part of that computation.
6. The baseline of the mstack element is determined by the `align` attribute.

Issue (overflows-mcolumn): Should an entry too large or too small for a column be centered?

Issue (mcolumn): Should an `mphantom` also act as a wrapper for computing digits? If so, people might be encouraged to use it to play alignment games that make the result not very accessible.

3.6.1.2 Attributes

`mstack` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
align	(top | bottom | center | baseline | axis) [*rownumber*]	baseline
specifies the vertical alignment of the `mstack` with respect to its environment. The legal values and their meanings are the same as that for `mtable`'s `align` attribute.		
stackalign	left | center | right | decimalseparator | . | ,	decimalseparator
specifies which column is used to horizontally align the rows. For `"left"`, rows are aligned flush on the left; similarly for `"right"`, rows are flush on the right; for `"center"`, the middle column (or to the right of the middle, for an even number of columns) is used for aligment. Rows with non-zero `position` or `shift` are treated as if the requisite number of empty columns were added on the appropriate side; see Section 3.6.3 and Section 3.6.4. For `"decimalseparator"`, `"."`, or `","`, the column used is whichever column in each row that contains a decimal separator. If there is no decimal separator, an implied decimal is assumed on the right of the first number in the row; See `"decimalseparator"` for a discussion of `"decimalseparator"`.		
charalign	left | center | right	right
specifies the horizontal alignment of digits within a column. If the content is larger than the column width, then it overflows the opposite side from the alignment. For example, for `"right"`, the content will overflow on the left side; for center, it overflows on both sides. This excess does not participate in the column width calculation, nor does it participate in the overall width of the `mstack`. In these cases, authors should take care to avoid collisions between column overflows.		
charspacing	*length*	0.1em
specifies the amount of space to put between each column. Larger spacing might be useful if carries are not placed above or are particularly wide.		

Issue (): Need to add a discussion of decimalseparator to mstyle.

Issue (multidigit-alignment): If there is more than one number in a row, which number should be used to determine the alignment if decimal point alignment is specfied?

3.6.2 Long Division `<mlongdiv>`

3.6.2.1 Description

Long division notation varies quite a bit around the world, although the heart of the notation is often similar. `mlongdiv` is similar to `mstack` and used to layout long division. The first two children of `mlongdiv` are the result of the division and the divisor. The remaining children are treated as if they were children of `mstack`. The placement of these and the lines and separators used to display long division are controlled by the `longdivstyle` attribute.

In the remainder of this section on elementary math, anything that is said about `mstack` applies to `mlongdiv` unless stated otherwise.

3.6.2.2 Attributes

`mlongdiv` elements accept all of the attributes that `mstack` elements accept (including those specified in Section 2.1.6), along with the attribute listed below.

The values allowed for `longdivstyle` are open-ended. Conforming renderers may ignore any value they do not handle, although renderers are encouraged to render as many of the values listed below as possible.

Name	values	default
longdivstyle	lefttop | stackedrightright | mediumstackedrightright | short-stackedrightright | righttop | left/\right | left)(right | :right=right | stackedleftleft | stackedleftlinetop	lefttop
Controls the style of the long division layout. The names are meant as a rough mnemonic that describes the position of the divisor and result in relation to the dividend.		

See Section 3.6.8.3 for examples of how these notations are drawn. The values listed above are used for long division notations in different countries around the world:

`"lefttop"` a notation that is commonly used in the United States, Great Britian, and elsewhere
`"stackedrightright"` a notation that is commonly used in used in France and elsewhere
`"mediumrightright"` a notation that is commonly used in used in Russia and elsewhere
`"shortstackedrightright"` a notation that is commonly used in used in Brazil and elsewhere
`"righttop"` a notation that is commonly used in used in China, Sweden, and elsewhere
`"left/\right"` a notation that is commonly used in used in Netherlands
`"left)(right"` a notation that is commonly used in used in India
`":right=right "` a notation that is commonly used in used in Germany
`"stackedleftleft "` a notation that is commonly used in Arabic countries
`"stackedleftlinetop"` a notation that is commonly used in used in Arabic countries

3.6.3 Group Rows with Similiar Positions `<msgroup>`

3.6.3.1 Description

`msgroup` is used to group rows inside of the `mstack` element that have a similar position relative to the alignment of stack. Any children besides `msrow`, `msgroup`, `mscarries` and `msline` are treated as if implicitly surrounded by an `msrow` (See Section 3.6.4 for more details about rows).

3.6.3.2 Attributes

`msgroup` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
position	[+ | -] *unsigned-integer*	0
specifies the position of the rows in this group relative to the column specified by `stackalign`: positive values move each row towards the tens digit, like multiplying by a power of 10, effectively padding with empty columns on the right; negative values move towards the ones digit, effectively padding on the left. The decimal point is counted as a column and should be taken into account for negative values.		
shift	[+ | -] *unsigned-integer*	0
specifies an incremental shift of position for successive rows in the group. The value is interpreted as with position, but specifies the position of each row (except the first) with respect to the previous row in the group.		

If both position and shift are set to "0", then msgroup has no effect.

3.6.4 Rows in Elementary Math <msrow>

3.6.4.1 Description

An msrow represents a row in an mstack. In most cases it is implied by the context, but is useful explicitly for putting multiple elements in a single row, such as when placing an operator "+" or "-" along side a number within an addition or subtraction.

If an mn element is a child of msrow (whether implicit or not), then the number is split into its digits and the digits are placed into successive columns. Any other element, with the exception of mstyle is treated effectively as a single digit occupying the next column. An mstyle is treated as if its children were the directly the children of the msrow, but with their style affected by the attributes of the mstyle. The empty element none may be used to create an empty column.

Note that a row is considered primarily as if it were a number, which are always displayed left-to-right, and so the directionality used to display the columns is always left-to-right; textual bidirectionality within token elements (other than mn) still applies, as does the overall directionality *within* any children of the msrow (which end up treated as single digits); see Section 3.1.5.

3.6.4.2 Attributes

msrow elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
position	[+ | -] *unsigned-integer*	0
specifies the position of the rows in this group relative to the column specified by stackalign: positive values move each row towards the tens digit, like multiplying by a power of 10, effectively padding with empty columns on the right; negative values move towards the ones digit, effectively padding on the left. The decimal point is counted as a column and should be taken into account for negative values.		
shift	[+ | -] *unsigned-integer*	0
specifies an incremental shift of position for this row with respect to the previous row of the mstack. The value is interpreted as with position. Note that the meaning of shift here is slightly different than from that used by msgroup, and that shift is ignored if position is also given on the same msrow.		

3.6.5 Carries, Borrows, and Crossouts <mscarries>

3.6.5.1 Description

mscarries is used for the various annotations such as carries, borrows, and crossouts that occur in elementary math. The children are associated with the element in the same column in the *following* row of the mstack, although this correspondence can be adjusted by position. Additionally, since these annotations are used to adorn what are treated as numbers, the attachment of carries to columns proceeds from left-to-right; The overall directionality does not apply to the ordering of the carries, although it may apply to the contents of each carry; see Section 3.1.5.

Each child of mscarries other than mscarry or none is treated as if implicitly surrounded by mscarry; the element none is used when no carry for a particular column is needed. mscarries increments scriptlevel, so the children are typically displayed in a smaller font. It also changes

`scriptsizemultiplier` from the inherited value; `scriptsizemultiplier` can be set on the `mscarries` element.

3.6.5.2 Attributes

`mscarries` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
position	[+ | -] *unsigned-integer*	0
Specifies the position of the group of carries relative to the column specified by `stackalign`. The interpretation of the value is the same as `position` for `msgroup` or `msrow`, but it alters the association of each carry with the column below. For example, `position=1` would cause the rightmost carry to be associated with the second digit column from the right.		
location	w | nw | n | ne | e | se | s | sw	n
specifies the location of the carry or borrow relative to the character below it in the associated column. Compass directions are used for the values; the default is to place the carry above the character.		
crossout	(none | updiagonalstrike | downdiagonalstrike | verticalstrike | horizontalstrike)*	none
specifies how the column content below each carry is "crossed out"; one or more values may be given and all values are drawn. If `"none"` is given with other values, it is ignored. See Section 3.6.8 for examples of the different values. The crossout is only applied for columns which have a corresponding `mscarry`.		
scriptsizemultiplier	*number*	0.6
specifies the factor to change the font size by. See Section 3.1.6 for a description of how this works with the `scriptsize` attribute.		

3.6.6 A Single Carry `<mscarry>`

3.6.6.1 Description

`mscarry` is used inside of `mscarries` to represent the carry for an individual column. A carry is treated as if its width were zero; it does not participate in the calculation of the width of its corresponding column; as such, it may extend beyond the column boundaries. Although it is usually implied, the element may be used explicitly to override the `location` and/or `crossout` attributes of the containing `mscarries`. It may also be useful with `none` as its content in order to display no actual carry, but still enable a `crossout` due to the enclosing `mscarries` to be drawn for the given column.

3.6.6.2 Attributes

`mscarries` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
location	w | nw | n | ne | e | se | s | sw	n
specifies the location of the carry or borrow relative to the character in the corresponding column in the row below it. Compass directions are used for the values.		
crossout	(none | updiagonalstrike | downdiagonalstrike | verticalstrike | horizontalstrike)*	none
specifies how the column content associated with the carry is "crossed out"; one or more values may be given and all values are drawn. If `"none"` is given with other values, it is essentially ignored.		

3.6.7 Horizontal Line `<msline/>`

3.6.7.1 Description

`msline` draws a horizontal line inside of a `mstack` element. The position, length, and thickness of the line are specified as attributes.

3.6.7.2 Attributes

`msline` elements accept the attributes listed below in addition to those specified in Section 2.1.6.

Name	values	default
position	*integer*	0
specifies the position of the line relative to the column specified by `stackalign`: positive values moves towards the tens digit (like multiplying by a power of 10); negative values moves towards the ones digit. The decimal point is counted as a column and should be taken into account for negative values.		
mslinethickness	*length* | thin | medium | thick	medium
Specifies how thick the line should be drawn. The line should have height=0, and depth=mslinethickness so that the top of the `msline` is on the baseline of the surrounding context (if any). (See Section 3.3.2 for discussion of the thickness keywords `"medium"`, `"thin"` and `"thick"`.)		
length	*unsigned-integer*	0
Specifies the the number of columns that should be spanned. '0' means all remaining columns in the row.		
leftoverhang	*length*	0
Specifies an extra amount that the line should overhang on the left of the leftmost column spanned by the line.		
rightoverhang	*length*	0
Specifies an extra amount that the line should overhang on the right of the rightmost column spanned by the line.		
mathcolor	*color*	*inherited*
Specifies the color to use to draw the line.		

3.6.8 Elementary Math Examples

3.6.8.1 Addition and Subtraction

Two-dimensional addition, subtraction, and multiplication typically involve numbers, carrries/borrows, lines, and the sign of the operation.

Notice that the `msline` spans all of the columns and that `none` is used to make the "+" appear to the left of all of the operands.

$$\begin{array}{r} 424 \\ +\ 33 \\ \hline \end{array}$$

The MathML for this is:

```
<mstack>
  <mn>424</mn>
  <msrow> <mo>+</mo> <none/> <mn>33</mn> </msrow>
```

```
  <msline/>
</mstack>
```

Here is an example with the operator on the right. Placing the operator on the right is standard in the Netherlands and some other countries. Notice that although there are a total of four columns in the example, because the default alignment in on the implied decimal point to the right of the numbers, it is not necessary to pad any row.

$$\begin{array}{r} 123 \\ 456+ \\ \hline 579 \end{array}$$

```
<mstack>
  <mn>123</mn>
  <msrow> <mn>456</mn> <mo>+</mo> </msrow>
  <msline/>
  <mn>579</mn>
</mstack>
```

Because the default alignment is placed to the right of number, the numbers align properly and none of the rows need to be shifted.

The following two examples illustrate the use of `mscarries`, `mscarry` and using `none` to fill in a column. The examples illustrate two different ways of displaying a borrow.

$$\begin{array}{r} {}^{2}2\,{}^{12}\cancel{3}\,\cancel{2}7 \\ -1{,}156 \\ \hline 1{,}171 \end{array} \qquad \begin{array}{r} {}^{2}2\,{}^{1}\cancel{3}\,27 \\ -1{,}156 \\ \hline 1{,}171 \end{array}$$

The MathML for the first example is:

```
<mstack>
  <mscarries crossout='updiagonalstrike'>  <mn>2</mn> <mn>12</mn> <none/> </mscarries>
  <mn>2,327</mn>
  <msrow> <mo>-</mo> <mn> 1,156</mn> </msrow>
  <msline/>
  <mn>1,171</mn>
</mstack>
```

The MathML for the second example uses `mscarry` because a crossout should only happen on a single column:

```
<mstack>
  <mscarries location='nw'>
    <none/>
    <mscarry crossout='updiagonalstrike'> <none/> </mscarry>
    <mn>1</mn>
    <none/>
  </mscarries>
  <mn>2,327</mn>
  <msrow> <mo>-</mo> <mn> 1,156</mn> </msrow>
  <msline/>
  <mn>1,171</mn>
</mstack>
```

Here is an example of subtraction where there is a borrow with multiple digits in a single column and a cross out. The borrowed amount is underlined (the example is from a Swedish source):

$$\begin{array}{r} \overset{10}{\underline{}} \\ \not{5}2 \\ -\,7 \\ \hline 45 \end{array}$$

There are two things to notice. The first is that `menclose` is used in the carry and that `none` is used for the empty element so that `mscarry` can be used to create a crossout.

```
<mstack>
  <mscarries>
    <mscarry crossout='updiagonalstrike'><none/></mscarry>
    <menclose notation='bottom'> <mn>10</mn> </menclose>
  </mscarries>
  <mn>52</mn>
  <msrow> <mo>-</mo> <mn> 7</mn> </msrow>
  <msline/>
  <mn>45</mn>
</mstack>
```

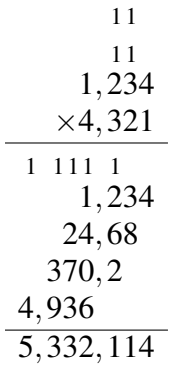
3.6.8.2 Multiplication

Below is a simple multiplication example that illustrates the use of `msgroup` and the `shift` attribute. The first `msgroup` does nothing. The second `msgroup` could also be removed, but `msrow` would be needed for its second and third children. They would set the `position` or `shift` attributes, or would add `none` elements.

$$\begin{array}{r} 123 \\ \times 321 \\ \hline 123 \\ 246 \\ 369 \\ \hline \end{array}$$

```
<mstack>
  <msgroup>
    <mn>123</mn>
    <msrow><mo>&#xD7;</mo><mn>321</mn></msrow>
  </msgroup>
  <msline/>
  <msgroup shift="1">
    <mn>123</mn>
    <mn>246</mn>
    <mn>369</mn>
  </msgroup>
  <msline/>
</mstack>
```

This example has multiple rows of carries. It also (somewhat artificially) includes commas (",") as digit separators. The encoding includes these separators in the spacing attribute value, along non-ASCII values.



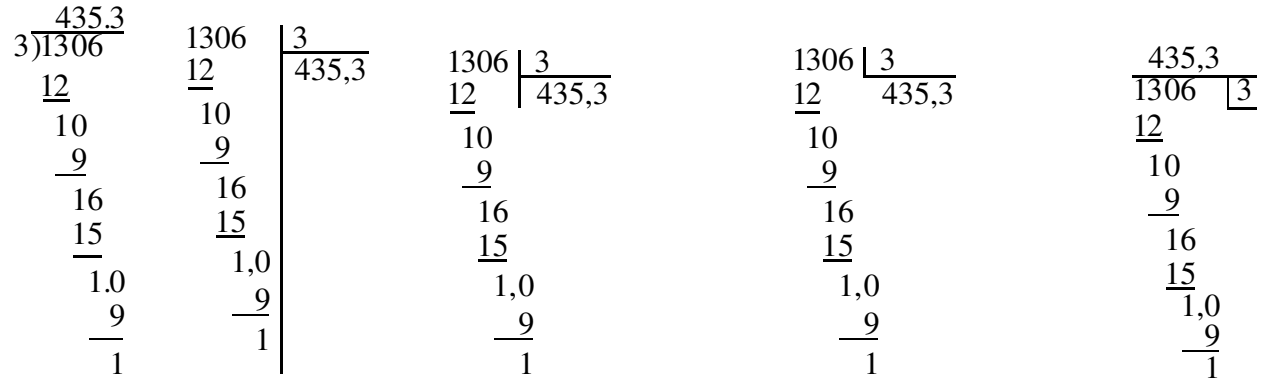
```
<mstack>
  <mscarries><mn>1</mn><mn>1</mn><none/></mscarries>
  <mscarries><mn>1</mn><mn>1</mn><none/></mscarries>
  <mn>1,234</mn>
  <msrow><mo>&#xD7;</mo><mn>4,321</mn></msrow>
  <msline/>

  <mscarries position='2'>
    <mn>1</mn>
    <none/>
    <mn>1</mn>
    <mn>1</mn>
    <mn>1</mn>
    <none/>
    <mn>1</mn>
  </mscarries>
  <msgroup shift="1">
    <mn>1,234</mn>
    <mn>24,68</mn>
    <mn>370,2</mn>
    <msrow shift="2"> <mn>4,936</mn> </msrow>
  </msgroup>
  <msline/>

  <mn>5,332,114</mn>
</mstack>
```

3.6.8.3 Long Division

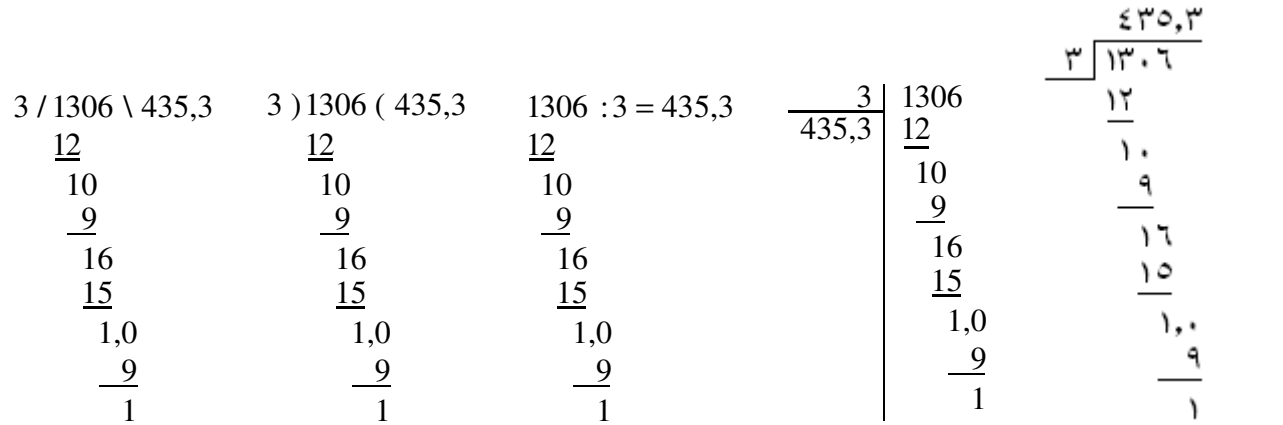
The notation used for long division varies considerably among countries. Most notations share the common characteristics of aligning intermediate results and drawing lines for the operands to be subtracted. Minus signs are sometimes shown for the intermediate calculations, and sometimes they are not. The line that is drawn varies in length depending upon the notation. The most apparently difference among the notations is that the position of the divisor varies, as does the location of the quotient, remainder, and intermediate terms.

The layout used is controlled by the `longdivstyle` attribute. Below are examples for the values listed in Section 3.6.2.2

"lefttop" "stackedrightright" "mediumstackedrightright" "shortstackedrightright" "righttop"



"left/\right" "left)(right" ":right=right" "stackedleftleft" "stackedleftlinetop"



The MathML for the first example is:

```
<mlongdiv longdivstyle="lefttop">
  <mn> 435.3</mn>
  <mn> 3 </mn>

  <msgroup>
    <mn> 1306</mn>
    <msrow position="2"><mn> 12</mn> </msrow>
  </msgroup>
  <msline length="2" position="2"/>

  <msgroup position="1">
    <mn> 10</mn>
    <mn> 9</mn>
    <msline length="2"/>
  </msgroup>

  <msgroup position="0">
    <mn> 16</mn>
    <mn> 15</mn>
    <msline length="2"/>
  </msgroup>

  <msgroup position="-2">      <!- negative to move to the right of the "." ->
```

```
    <mn> 1.0</mn>
    <mn> 9</mn>
    <msline length="3"/>
  </msgroup>

  <mn> 1</mn>
</mlongdiv>
```

With the exception of the last example, the encodings for the other examples are the same except that the values for `longdivstyle` differ and that a "," is used instead of a "." for the decimal point. For the last example, the only difference from the other examples besides a different value for `longdivstyle` is that Arabic numerals have been used in place of Latin numerals.

3.6.8.4 Repeating decimal

Decimal numbers that have digits that repeat infinitely such as 1/3 (.3333...) are represented using several notations. One common notation is to put a horizontal line over the digits that repeat (in Portugal an underline is used). Another notation involves putting dots over the digits that repeat. These notations are shown below:

$$0.333333\overline{3}$$

$$0.\overline{142857}$$

$$0.\underline{142857}$$

$$0.\dot{1}42857\dot{7}$$

The MathML for these involves using `mstack`, `msrow`, and `msline` in a straightforward manner. The MathML for the preceeding examples above is given below.

```
<mstack>
  <msline length="1"/>
  <mn> 0.3333 </mn>
</mstack>
<mstack>
  <msline length="6"/>
  <mn> 0.142857 </mn>
</mstack>
<mstack>
  <mn> 0.142857 </mn>
  <msline length="6"/>
</mstack>
<mstack>
  <msrow> <mo>.</mo> <none/><none/><none/><none/> <mo>.</mo> </msrow>
  <mn> 0.142857 </mn>
</mstack>
```

3.7 Enlivening Expressions

3.7.1 Bind Action to Sub-Expression <maction>

Issue (): There is concensus that maction should be deprecated or restricted in some way. There is also consensus that in any event, all attribute values and their behavior should be fully specified (in contrast to the present text.) Note that maction is currently used for linking, so the fate of maction is tied to producing a satisfactory substitute. There is also a dependency on the decision on how to handle foreign markup within MathML. MathQTI has a requirement for form elements that appear in typeset equations, e.g. an input field for an exponent, which could be satisfied by either maction or XForms.

To provide a mechanism for binding actions to expressions, MathML provides the maction element. This element accepts any number of sub-expressions as arguments and the type of action that should happen is controlled by the actiontype attribute. Only three actions are predefined by MathML, but the list of possible actions is open. Additional predefined actions may be added in future versions of MathML.

Linking to other elements, either locally within the math element or to some URL, is not handled by maction. Instead, it is handled by adding a link directly on a MathML element as specified in Section 6.4.1.

3.7.1.1 Attributes

maction elements accept the attributes listed below in addition to those specified in Section 2.1.6.

By default, MathML applications that do not recognize the specified actiontype should render the selected sub-expression as defined below. If no selected sub-expression exists, it is a MathML error; the appropriate rendering in that case is as described in Section 2.3.2.

Name	values	default
actiontype	toggle | statusline | tooltip | input	*required*
Specifies what should happen for this element. The values allowed are open-ended. Conforming renderers may ignore any value they do not handle, although renderers are encouraged to render the listed values.		
selection	*positive-integer*	1
Specifies which child should be used for viewing. Its value should be between 1 and the number of children of the element. The specified child is referred to as the 'selected sub-expression' of the maction element. If the value specified is out of range, it is an error. When the selection attribute is not specified (including for actiontypes for which it makes no sense), its default value is 1, so the selected sub-expression will be the first sub-expression.		

If a MathML application responds to a user command to copy a MathML sub-expression to the environment's 'clipboard' (see Section 6.3), any maction elements present in what is copied should be given selection values that correspond to their selection state in the MathML rendering at the time of the copy command.

The meanings of the various actiontype values is given below. Note that not all renderers support all of the actiontype values, and that the allowed values are open-ended.

<maction actiontype="toggle" selection="positive-integer" > (first expression) (second expression)... </maction>

The renderer alternately display the selected subexpression, cycling through them when there is a click on the selected subexpression. Each click increments the selection value, wrapping back to 1 when it reaches the last child. Typical uses would be for exercises in education,

ellipses in long computer algebra output, or to illustrate alternate notations. Note that the expressions may be of significantly different size, so that size negotiation with the browser may be desirable. If size negotiation is not available, scrolling, elision, panning, or some other method may be necessary to allow full viewing.

<maction actiontype="statusline"> (expression) (message) </maction>
The renderer displays the first child. When a reader clicks on the expression or moves the pointer over it, the renderer sends a rendering of the message to the browser statusline. Because most browsers in the foreseeable future are likely to be limited to displaying text on their statusline, the second child should be an `mtext` element in most circumstances. For non-`mtext` messages, renderers might provide a natural language translation of the markup, but this is not required.

<maction actiontype="tooltip"> (expression) (message) </maction>
The renderer displays the first child. When the pointer pauses over the expression for a long enough delay time, the renderer displays a rendering of the message in a pop-up 'tooltip' box near the expression. Many systems may limit the popup to be text, so the second child should be an `mtext` element in most circumstances. For non-`mtext` messages, renderers may provide a natural language translation of the markup if full MathML rendering is not practical, but this is not required.

<maction actiontype="input"> (expression) </maction>
The renderer displays the expression. For renderers that allow editing, when focus is passed to this element, the `maction` is replaced by what is entered, pasted, etc. MathML does not restrict what is allowed as input, nor does it require an editor to allow arbitrary input. Some renderers/editors may restrict the input to simple (linear) text.

The `actiontype` values are open-ended. If another value is given and it requires additional attributes, the attributes must be in a different namespace. This is shown below:

<maction actiontype="highlight" my:color="red" my:background="yellow"> expression </maction>
In the example, non-standard attributes from another namespace are being used to pass additional information to renderers that support them, without violating the MathML DTD (see Section 2.3.3). The `my:color` attributes might change the color of the characters in the presentation, while the `my:background` attribute might change the color of the background behind the characters.

3.8 Semantics and Presentation

MathML uses the `semantics` element to allow specifying semantic annotations to presentation MathML elements; these can be content MathML or other notations. As such, `semantics` should be considered part of both presentation MathML and content MathML. All MathML processors should process the `semantics` element, even if they only process one of those subsets.

In semantic annotations a presentation MathML expression is typically the first child of the `semantics` element. However, it can also be given inside of an `annotation-xml` element inside the `semantics` element. If it is part of an `annotation-xml` element, then `encoding="MathML-presentation"` must be used and presentation MathML processors should use this value for the presentation.

See Section 5.1 for more details about the `semantics` and `annotation-xml` elements.

Chapter 4

Content Markup

4.1 Introduction

4.1.1 The Intent of Content Markup

The intent of Content Markup is to provide an explicit encoding of the *underlying mathematical meaning* of an expression, rather than any particular rendering for the expression. Mathematics is distinguished both by its use of rigorous formal logic to define and analyze mathematical concepts, and by the use of a (relatively) formal notational system to represent and communicate those concepts. However, mathematics and its presentation should not be viewed as one and the same thing. Mathematical notation, though more rigorous than natural language, is nonetheless at times ambiguous, context-dependent, and varies from community to community. In some cases, heuristics may adequately infer mathematical semantics from mathematical notation. But in many others cases, it is preferable to work directly with the underlying, formal, mathematical objects. Content Markup provides a rigorous, extensible semantic framework and a markup language for this purpose.

The difficulties in inferring semantics from a presentation stem from the fact that there are many to one mappings from presentation to semantics and vice versa. For example the mathematical construct 'H multiplied by e' is often encoded using an explicit operator as in $H \times e$. In different presentational contexts, the multiplication operator might be invisible '$H\ e$', or rendered as the spoken word 'times'. Generally, many different presentations are possible depending on the context and style preferences of the author or reader. Thus, given '$H\ e$' out of context it may be impossible to decide if this is the name of a chemical or a mathematical product of two variables H and e. Mathematical presentation also changes with culture and time: some expressions in combinatorial mathematics today have one meaning to a Russian mathematician, and quite another to a French mathematician. Notations may lose currency, for example the use of musical sharp and flat symbols to denote maxima and minima [Chaundy1954]. A notation in use in 1644 for the multiplication mentioned above was ■He [Cajori1928].

By encoding the underlying mathematical structure explicitly, without regard to how it is presented aurally or visually, it is possible to interchange information more precisely between systems that semantically process mathematical objects. In the trivial example above, such a system could substitute values for the variables H and e and evaluate the result. Important application areas include computer algebra systems, automatic reasoning system, industrial and scientific applications, multi-lingual translation systems, mathematical search, and interactive textbooks.

The organization of this chapter is as follows. In Section 4.2, a core collection of elements comprising Strict Content Markup are described. Strict Content Markup is sufficient to encode general expression trees in a semantically rigorous way. It is in one-to-one correspondence with OpenMath element set. OpenMath is a standard for representing formal mathematical objects and semantics through the use of extensible Content Dictionaries. Strict Content Markup defines a mechanism for associating precise mathematical semantics with expression trees by referencing OpenMath Content Dictionaries. In Sec-

tion 4.3, markup is introduced for representing a small number of mathematical idioms, such as limits on integrals, sums and product. These constructs may all be rewritten as Strict Content Markup expressions, and rules for doing so are given. In Section 4.4, elements are introduced for many common function, operators and constants. This section contains many examples, including equivalent Strict Content expressions. Section 4.5 is a minor section. Finally, Section 4.6 summarizes the alrogrithm for translating arbitrary Content Markup into Strict Content Markup. It collects together in sequence all the rewrite rules introduced throughout the rest of the chapter.

4.1.2 The Structure and Scope of Content MathML Expressions

Content MathML represents mathematical objects as *expression trees*. The notion of constructing a general expression tree is e.g. that of applying an operator to sub-objects. For example, the sum 'x+y' can be thought of as an application of the addition operator to two arguments x and y. And the expression 'cos(π)' as the application of the cosine function to the number π.

As a general rule, the terminal nodes in the tree represent basic mathematical objects such as numbers, variables, arithmetic operations and so on. The internal nodes in the tree represent function application or other mathematical constructions that build up a compound objects. Function application provides the most important example; an internal node might represent the application of a function to several arguments, which are themselves represented by the nodes underneath the internal node.

The semantics of general mathematical expressions is not a matter of consensus. It would be an enormous job to systematically codify most of mathematics – a task that can never be complete. Instead, MathML makes explicit a relatively small number of commonplace mathematical constructs, chosen carefully to be sufficient in a large number of applications. In addition, it provides a mechanism for associating semantics with new notational constructs. In this way, mathematical concepts that are not in the base collection of elements can still be encoded.

The base set of content elements is chosen to be adequate for simple coding of most of the formulas used from kindergarten to the end of high school in the United States, and probably beyond through the first two years of college, that is up to A-Level or Baccalaureate level in Europe.

While the primary role of the MathML content element set is to directly encode the mathematical structure of expressions independent of the notation used to present the objects, rendering issues cannot be ignored. There are different approaches for rendering Content MathML formulae, ranging from from native implementations of the MathML elements to declarative notation definitions, to XSLT style sheets. The MathML 3 Recommendation will not make one of these normative, but only specify sample notations by way of examples.

4.1.3 Strict Content MathML

In MathML 3, a subset, or profile, of Content MathML is defined: *Strict Content MathML*. This uses a minimal set of elements to represent the meaning of a mathematical expression in a uniform structure, while the full Content MathML grammar is backward compatible with MathML 2.0, and generally tries to strike a more pragmatic balance between verbosity and formality.

Content MathML provides a large number of predefined functions encoded as empty elements (e.g. `sin`, `log`, etc.) and a variety of constructs for forming compound objects (e.g. `set`, `interval`, etc.). By contrast, Strict Content MathML uses a single element (`csymbol`) with an attribute pointing to an external definition in extensible content dictionaries to represent all functions, and uses only `apply` and `bind` for building up compound objects. The token elements such as `ci` and `cn` are also considered part of Strict Content MathML, but with a more restricted set of attributes and with content restricted to text.

In particular, Strict Content MathML is designed to be compatible with OpenMath (in fact it is an XML encoding of OpenMath Objects in the sense of [OpenMath2004]). OpenMath is a standard for representing formal mathematical objects and semantics through the use of extensible Content Dictionaries. The table below gives an element-by-element correspondence between the OpenMath XML encoding of OpenMath objects and Strict Content MathML.

Strict Content MathML	OpenMath
`cn`	`OMI`, `OMF`
`csymbol`	`OMS`
`ci`	`OMV`
`cs`	`OMSTR`
`apply`	`OMA`
`bind`	`OMBIND`
`bvar`	`OMBVAR`
`share`	`OMR`
`semantics`	`OMATTR`
`annotation`, `annotation-xml`	`OMATP`, `OMFOREIGN`
`error`	`OME`
`cbytes`	`OMB`

In MathML 3, formal semantics for general Content MathML expressions are given by specifying equivalent Strict Content MathML expressions, so that they inherit their semantics. To make the correspondence exact, a transformation algorithm is given in terms of transformation rules that are applied in order to rewrite particular MathML constructs into a strict equivalents. The individual rules are introduce in context throughout the chapter. In Section 4.6, the algorithm as a whole is described.

As most transformation rules relate to classes of MathML elements that have similar argument structure, they are introduced in Section 4.3.4 where these classes are defined. Some special case rules for specific elements are given in Section Section 4.4. Transformations in Section 4.2 concern extended usages of the core Content MathML elements, those in Section 4.3 concern the rewriting of some additional structures not supported in Strict Content MathML.

The transformation algorithm from Section 4.6 is complete: it gives every Content MathML expression a specific meaning in terms of a Strict Content MathML expression. This means it has to give specific strict interpretations to some expressions whose meaning was insufficiently specified in MathML2. The intention of this algorithm is to be faithful to mathematical intuitions. However edge cases may remain where the normative interpretation of the algorithm may break earlier intuitions.

A conformant MathML processor need not implement this transformation. The existence of these transformation rules does not imply that a system must treat equivalent expressions identically. In particular systems may give different presentation renderings for expresssions that the transformation rules imply are mathematically equivalent.

4.1.4 Content Dictionaries

Due to the nature of mathematics, any method for formalizing the meaning of the mathematical expressions must be extensible. The key to extensibility is the ability to define new functions and other symbols to expand the terrain of mathematical discourse. To do this, two things are required: a mechanism for representing symbols not already defined by Content MathML, and a means of associating a specific mathematical meaning with them in an unambiguous way. In MathML 3, the `csymbol` element provides the means to represent new symbols, while *Content Dictionaries* are the way in which mathematical semantics are described. The association is accomplished via attributes of the `csymbol`

element that point at a definition in a CD. The syntax and usage of these attributes are described in detail in Section 4.2.3.

Content Dictionaries are structured documents for the definition of mathematical concepts; see the OpenMath standard, [OpenMath2004]. To maximize modularity and reuse, a Content Dictionary typically contains a relatively small collection of definitions for closely related concepts. The OpenMath Society maintains a large set of public Content Dictionaries including the MathML CD group that including contains definitions for all pre-defined symbols in MathML. There is a process for contributing privately developed CDs to the OpenMath Society repository to facilitate discovery and reuse. MathML 3 does not require CDs be publicly available, though in most situations the goals of semantic markup will be best served by referencing public CDs available to all user agents.

In the text below, descriptions of semantics for predefined MathML symbols refer to the Content Dictionaries developed by the OpenMath Society in conjunction with the W3C Math Working Group. It is important to note, however, that this information is informative, and not normative. In general, the precise mathematical semantics of predefined symbols are not not fully specified by the MathML 3 Recommendation, and the only normative statements about symbol semantics are those present in the text of this chapter. The semantic definitions provided by the OpenMath Content CDs are intended to be sufficient for most applications, and are generally compatible with the semantics specified for analogous constructs in the MathML 2.0 Recommendation. However, in contexts where highly precise semantics are required (e.g. communication between computer algebra systems, within formal systems such as theorem provers, etc.) it is the responsibility of the relevant community of practice to verify, extend or replace definitions provided by OpenMath CDs as appropriate.

4.2 Content MathML Elements Encoding Expression Structure

In this section we will present the elements for encoding the structure of content MathML expressions. These elements are the only ones used for the Strict Content MathML encoding. Concretely, we have

- basic expressions, i.e. Numbers, string literals, encoded bytes, Symbols, and Identifiers.
- derived expressions, i.e. function applications and binding expressions, and
- semantic annotations
- error markup

Full Content MathML allows further elements presented in Section 4.3 and Section 4.4, and allows a richer content model presented in this section. We will contrast the strict and full content models in syntax tables at the beginning of the element specifications.

In these tables, the `Content`, `Attributes`, and `Attribute Values` rows specify the XML encoding. Where applicable, the `Class` row specifies the operator class, which indicate how many arguments the operator represented by this element takes, and also in many cases determines the mapping to Strict Content MathML, as described in Section 4.3.4. Finally, the `Qualifiers` row clarifies whether the operator takes qualifiers and if so, which. Both specify how many siblings may follow the operator element in an `apply`; see Section 4.2.5 and Section 4.3.3 for details).

4.2.1 Numbers <cn>

	Schema Fragment (Strict)	Schema Fragment (Full)	
Class	Cn	Cn	
Attributes	CommonAtt, type	CommonAtt, type?, base?	
type Attribute Values	"integer" | "real" | "double" | "hexdouble" |	"integer" | "real" | "double" | "hexdouble" | "e-notation" | "rational" | "complex-cartesian" | "complex-polar" | "constant"	*real*
base Attribute Values		integer	*10*
Content	text	(text |mglyph |sep | PresentationExp)*	

The cn element is the Content MathML element used to represent numbers. Strict Content MathML supports integers, real numbers, and double precision floating point numbers. In these types of numbers, the content of cn is text. Additionally, cn supports rational numbers and complex numbers in which the different parts are separated by use of the sep element. Constructs using sep may be rewritten in Strict Content MathML as constructs using apply as described below.

The type attribute specifies which kind of number is represented in the cn element. The default value is "real". Each type implies that the content be of a certain form, as detailed below.

4.2.1.1 Rendering <cn>-Represented Numbers

The default rendering of the text content of cn is the same as that of the Presentation element mn, with suggested variants in the case of attributes or sep being used, as listed below.

4.2.1.2 Strict Content MathML

In Strict Content MathML, the type attribute is mandatory, and may only take the values "integer", "real", "hexdouble" or "double":

integer An integer is represented by an optional sign followed by a string of one or more decimal 'digits'.

real A real number is presented in radix notation. Radix notation consists of an optional sign ('+' or '-') followed by a string of digits possibly separated into an integer and a fractional part by a 'decimal point'. Some examples are 0.3, 1, and -31.56.

double This type is used to mark up those double-precision floating point numbers that can be represented in the IEEE 754 standard format [IEEE754]. This includes a subset of the (mathematical) real numbers, negative zero, positive and negative real infinity and a set of "not a number" values. The lexical rules for interpreting the text content of a cn as an IEEE double are specified by Section 3.1.2.5 of XML Schema Part 2: Datatypes Second Edition [XMLSchemaDatatypes]. For example, -1E4, 1267.43233E12, 12.78e-2, 12 , -0, 0 and INF are all valid doubles in this format.

hexdouble This type is used to directly represent the the 64 bits of an IEEE 754 double-precision floating point number as a 16 digit hexadecimal number. Thus the number represents mantissa, exponent, and sign from lowest to highest bits using a least significant byte ordering. This consists of a string of 16 digits 0-9, A-F. The following example represents a NaN value. Note that certain IEEE doubles, such as the preceding NaN, cannot be represented in the lexical format for the "double" type.

```
<cn type="hexdouble">7F800000</cn>
```

Sample Presentation

```
<mn>0x7F800000</mn>
```

0x7F800000

4.2.1.3 Extended uses of <cn>

The base attribute is used to specify how the content is to be parsed. The attribute value is a base 10 positive integer giving the value of base in which the text content of the cn is to be interpreted. The base attribute should only be used on elements with type "integer" or "real". Its use on cn elements of other type is deprecated. The default value for base is "10".

Additional values for the type attribute element for supporting e-notations for real numbers, rational numbers, complex numbers and selected important constants. As with the "integer", "real", "double" and "hexdouble" types, each of these types implies that the content be of a certain form. If the type attribute is omitted, it defaults to "real".

integer Integers can be represented with respect to a base different from 10: If base is present, it specifies (in base 10) the base for the digit encoding. Thus base='16' specifies a hexadecimal encoding. When base > 10, Latin letters (A-Z, a-z) are used in alphabetical order as digits. The case of letters used as digits is not significant. The following example encodes the number written as 32736 in base ten.

```
<cn base="16">7FE0</cn>
```

Sample Presentation

```
<msub><mn>7FE0</mn><mn>16</mn></msub>
```

$7FE0_{16}$

When base > 36, some integers cannot be represented using numbers and letters alone. For example, while

```
<cn base="1000">10F</cn>
```

arguably represents the number written in base 10 as 1,000,015, the number written in base 10 as 1,000,037 cannot be represented using letters and numbers alone when base is 1000. Consequently, it is up to applications to specify what additional characters (if any) may be used for digits when base > 36.

real Real numbers can be represented with respect to a base different than 10. If a base attribute is present, then the digits are interpreted as being digits computed relative to that base (in the same way as described for type "integer").

e-notation A real number may be presented in scientific notation using this type. Such numbers have two parts (a significand and an exponent) separated by a <sep/> element. The first part is a real number, while the second part is an integer exponent indicating a power of the base. For example, <cn type="e-notation">12.3<sep/>5</cn> represents 12.3 times 10^5. The default presentation of this example is 12.3e5. Note that this type is primarily useful for backwards compatibility with MathML 2, and in most cases, it is preferable to use the "double" type, if the number to be represented is in the range of IEEE doubles:

rational A rational number is given as two integers to be used as the numerator and denominator of a quotient. The numerator and denominator are separated by <sep/>.

```
<cn type="rational">22<sep/>7</cn>
```

Sample Presentation

```
<mrow><mn>22</mn><mo>/</mo><mn>7</mn></mrow>
```

$$22/7$$

complex-cartesian A complex cartesian number is given as two numbers specifying the real and imaginary parts. The real and imaginary parts are separated by the `<sep/>` element, and each part has the format of a real number as described above.

```
<cn type="complex-cartesian"> 12.3 <sep/> 5 </cn>
```

Sample Presentation

```
<mrow>
 <mn>12.3</mn><mo>+</mo><mn>5</mn><mo>&#x2062;</mo><mi>i</mi>
</mrow>
```

$$12.3+5i$$

complex-polar A complex polar number is given as two numbers specifying the magnitude and angle. The magnitude and angle are separated by the `<sep/>` element, and each part has the format of a real number as described above.

```
<cn type="complex-polar"> 2 <sep/> 3.1415 </cn>
```

Sample Presentation

```
<mrow>
 <mn>2</mn>
 <mo>&#x2062;</mo>
 <msup>
  <mi>e</mi>
  <mrow><mi>i</mi><mo>&#x2062;</mo><mn>3.1415</mn></mrow>
 </msup>
</mrow>
```

$$2e^{i3.1415}$$

```
<mrow>
 <mi>Polar</mi>
 <mo>&#x2061;</mo>
 <mfenced><mn>2</mn><mn>3.1415</mn></mfenced>
</mrow>
```

$$\text{Polar}\,(2,3.1415)$$

constant If the value `type` is `"constant"`, then the content should be Unicode representations of a well-known constant. Some important constants and their common Unicode representations are listed below.This `cn` type is primarily for backward compatibility with MathML 1.0. MathML 2.0 introduced many empty elements, such as `<pi/>` to represent constants, and the empty element representations are preferred.

Mapping to Strict Content MathML

If a `base` attribute is present, it specifies the base used for the digit encoding of both integers. The use of `base` with `"rational"` numbers is deprecated.

Rewrite: cn sep

If there are sep children of the cn, then intervening text may be rewritten as cn elements. If the cn element containing sep also has a base attribute, this is copied to each of the cn arguments of the resulting symbol, as shown below.

```
<cn type="rational" base="b">n<sep/>d</cn>
```

is rewritten to

```
<apply><csymbol cd="num1">rational</csymbol>
  <cn type="integer" base="b">n</cn>
  <cn type="integer" base="b">d</cn>
</apply>
```

The symbol used in the result depends on the type attribute according to the following table:

type attribute	OpenMath Symbol
e-notation	bigfloat
rational	rational
complex-cartesian	complex_cartesian
complex-polar	complex_polar

Note: In the case of bigfloat the symbol takes three arguments, `<cn type="integer">10</cn>` should be inserted as the second argument, denoting the base of the exponent used.

If the type attribute has a different value, or if there is more than one `<sep/>` element, then the intervening expressions are converted as above, but a system-dependent choice of symbol for the head of the application must be used.

If a base attribute has been used then the resulting expression is not Strict Content MathML, and each of the arguments needs to be recursively processed.

Rewrite: cn based_integer

A cn element with a base attribute other than 10 is rewritten as follows. (A base attribute with value 10 is simply removed) .

```
<cn type="integer" base="16">FF60</cn>
```

```
<apply><csymbol cd="nums1">based_integer</csymbol>
  <cn type="integer">16</cn>
  <cs>FF60</cs>
</apply>
```

If the original element specified type "integer" or if there is no type attribute, but the the content of the element just consists of the characters [a-zA-Z0-9] and white space then the symbol used as the head in the resulting application should be based_integer as shown. Otherwise it should be should be based_float.

Rewrite: cn constant

In Strict Content MathML, constants should be represented using `csymbol` elements. A number of important constants are defined in the nums1 content dictionary. An expression of the form

```
<cn type="constant">c</cn>
```

has the Strict Content MathML equivalent

```
<csymbol cd="nums1">c2</csymbol>
```

where `c2` corresponds to `c` as specified in the following table.

Content	Description	OpenMath Symbol
U+03C0 (`&pi;`)	The usual π of trigonometry: approximately 3.141592653...	pi
U+2147 (`&ExponentialE;` or `&ee;`)	The base for natural logarithms: approximately 2.718281828...	e
U+2148 (`&ImaginaryI;` or `&ii;`)	Square root of -1	i
U+03B3 (`&gamma;`)	Euler's constant: approximately 0.5772156649...	gamma
U+221E (`&infin;` or `&infty;`)	Infinity. Proper interpretation varies with context	infinity

4.2.2 Content Identifiers `<ci>`

	Schema Fragment (Strict)	Schema Fragment (Full)
Class	Ci	Ci
Attributes	CommonAtt, type?	CommonAtt, type?
`type` Attribute Values	`"integer"`, `"rational"`, `"real"`, `"complex"`, `"complex-polar"` `"complex-cartesian"`, `"constant"`, `"function"`, `"vector"`, `"list"`, `"set"`, `"matrix"`	string
Qualifiers		BvarQ, DomainQ, degree, momentabout, logbase
Content	text	StringMglyph | PresentationExp

Content identifiers represent 'mathematical variables' which have properties, but no fixed value, e.g. *x* and *y* in the sum expression '*x*+*y*' above. Mathematically, we distinguish 'bound variables' which are in the scope of a binding construct from 'free variables' i.e. ones that are not; see Section 4.2.6.1 for details.

4.2.2.1 Strict Content MathML

Content MathML uses the `ci` element (mnemonic for 'content identifier') to construct a variable, i.e. an identifier that is not a symbol. In the sum expression '*x*+*y*' above, the variable *x* would be represented as

```
<ci>x</ci>
```

After white space normalization the content of a `ci` element is interpreted as a name that identifies it. Two variables are considered equal, if and only if their names are identical and in the same scope (see Section 4.2.6 for a discussion).

The `ci` element uses the `type` attribute to specify the basic type of object that it represents. In Strict Content MathML, the set of permissible values is `"integer"`, `"rational"`, `"real"`, `"complex"`, `"complex-polar"`, `"complex-cartesian"`, `"constant"`, `"function"`, `vector`, `list`, `set`, and `matrix`. These values correspond to the symbols integer_type, rational_type, real_type, complex_polar_type, complex_cartesian_type, constant_type, fn_type, vector_type, list_type, set_type, and matrix_type in the mathmltypes Content Dictionary: In this sense the following two expressions are considered equivalent:

```
<ci type="integer">n</ci>
```

```
<semantics>
  <ci>n</ci>
  <annotation-xml cd="mathmltypes" name="type" encoding="MathML Content">
    <csymbol cd="mathmltypes">integer_type</csymbol>
  </annotation-xml>
</semantics>
```

4.2.2.2 Extended uses of `<ci>`

The `ci` element allows any string value for the `type` attribute, in particular any of the names of the MathML container elements or their type values.

Mapping to Strict Content MathML

Rewrite: ci type annotation

In Strict Content, type attributes are represented via semantic attribution. An expression of the form

```
<ci type="T">n</ci>
```

is rewritten to

```
<semantics>
  <ci>n</ci>
  <annotation-xml cd="mathmltypes" name="type" encoding="MathML Content">
    <ci>T</ci>
  </annotation-xml>
</semantics>
```

For a more advanced treatment of types, the `type` attribute is inappropriate. Advanced types require significant structure of their own (for example, $vector(complex)$) and are probably best constructed as mathematical objects and then associated with a MathML expression through use of the `semantics` element. See Section 4.2.8.1 for an example and [MathMLTypes] for more examples.

In addition to the forms described above, the `ci` and element can contain `mglyph` elements to refer to characters not currently available in Unicode, or a general presentation construct (see Section 3.1.9), which is used for rendering (see Section 4.1.2).

Rewrite: ci presentation mathml

An `ci` expression with non-text content of the form

`<ci> P </ci>`

is transformed to Strict Content MathML by rewriting it to

```
<semantics>
  <ci>p</ci>
  <annotation-xml encoding="MathML Presentation">
     P
  </annotation-xml>
</semantics>
```

Where the identifier name (which has to be a text string) should be determined from the presentation MathML content, in a system defined way, perhaps as in the above example by taking the character data of the element, ignoring any element markup. Systems doing such rewriting should ensure that constructs using the same Presentation MathML content are rewritten to `semantics` elements using the same `ci`, and that conversely constructs that use different MathML should be rewritten to different identifier names (even if the Presentation MathML has the same character data).

The following example encodes an atomic symbol that displays visually as C^2 and that, for purposes of content, is treated as a single symbol

```
<ci>
  <msup><mi>C</mi><mn>2</mn></msup>
</ci>
```

The Strict Content MathML equivalent is

```
<semantics>
  <ci>C2</ci>
  <annotation-xml encoding="MathML Presentation">
    <msup><mi>C</mi><mn>2</mn></msup>
  </annotation-xml>
</semantics>
```

Sample Presentation

```
<msup><mi>C</mi><mn>2</mn></msup>
```

C^2

4.2.2.3 Rendering Content Identifiers

If the content of a `ci` element consists of Presentation MathML, that presentation is used. If no such tagging is supplied then the text content is rendered as if it were the content of an `mi` element. If an application supports bidirectional text rendering, then the rendering follows the Unicode bidirectional rendering.

The `type` attribute can be interpreted to provide rendering information. For example in

```
<ci type="vector">V</ci>
```

a renderer could display a bold V for the vector.

4.2.3 Content Symbols <csymbol>

	Schema Fragment (Strict)	Schema Fragment (Full)
Class	Csymbol	Csymbol
Attributes	CommonAtt, cd	CommonAtt, TypeAtt?, cd?,
Content	Name	StringMglyph | PresentationExp
Qualifiers		BvarQ, DomainQ, degree, momentabout, logbase

Content MathML makes a crucial semantic distinction between a function itself and the expression resulting from applying that function to zero or more arguments. This is addressed by making functions self-contained objects with their own properties and providing an explicit `apply` construct corresponding to function application. We will consider the `apply` construct in the next section.

In the sum expression '*x*+*y*' above, *x* and *y* are typically taken to be '`variables`', since they have properties, but no fixed value, whereas the addition function is a '`constant`' or '`symbol`' as it denotes a specific function, which is defined somewhere externally. Note that the term 'symbol' is used here in the abstract sense and has no connection with any presentation of the construct on screen or paper. These are handled by the infrastructure in Chapter 3.

4.2.3.1 Strict Content MathML

A `csymbol` is used to refer to a specific, mathematically-defined concept with an external definition referenced via attributes. Conceptually, a reference to an external definition is merely a URI, i.e. a label uniquely identifying the definition. However, to be useful for communication between user agents, external definitions must be shared. For this reason, over the years several efforts have been organized to develop systematic, public repositories of mathematical definitions. Of these, the ongoing development of OpenMath Content Dictionaries (CDs) is the most open and extensive, and in MathML 3, OpenMath CDs are the preferred source of external definitions. In particular, the definitions of pre-defined MathML 3 operators and functions are given in terms of OpenMath CDs.

MathML 3 provides two mechanisms for referencing external definitions or content dictionaries. The first, using the `cd` attribute, follows conventions established by OpenMath specifically for referencing CDs. The second, using the `definitionURL` attribute, is backward compatible with MathML 2, and can be used to reference CDs or any other source of definitions that can be identified by a URI.

When referencing OpenMath CDs, the preferred method is to use the `cd` attribute as follows. Abstractly, OpenMath symbol definitions are identified by a triple of values: a *symbol name*, a *CD name*, and a *CD base*, which is a URI that disambiguates CDs of the same name. To associate such a triple with a `csymbol`, the content of the `csymbol` specifies the symbol name, and the name of the Content Dictionary is given using the `cd` attribute. The CD base is determined either from the document embedding the `math` element which contains the `csymbol` by a mechanism given by the embedding document format, or by system defaults, or by the `cdgroup` attribute , which is optionally specified on the enclosing `math` element; see Section 2.2.1. In the absence of specific information `http://www.openmath.org/cd` is assumed as the CD base for all `csymbol` elements `annotation`, and `annotation-xml`. This is the CD base for the collection of standard CDs maintained by the OpenMath Society.

The `cdgroup` specifies a URL to an OpenMath CD Group file. For a detailed description of the format of a CD Group file, see Section 4.4.2 (CDGroups) in [OpenMath2004]. Conceptually, a CD group file is a list of pairs consisting of a CD name, and a corresponding CD base. When a `csymbol` references a CD name using the `cd` attribute, the name is looked up in the CD Group file, and the associated CD base value is used for that `csymbol`. When a CD Group file is specified, but a referenced CD name does not appear in the group file, or there is an error in retrieving the group file, the referencing `csymbol` is

not defined. However, the handling of the resulting error is not defined, and is the responsibility of the user agent.

While references to external definitions are URIs, it is strongly recommended that CD files be retrievable at the location obtained by interpreting the URI as a URL. In particular, other properties of the symbol being defined may be available by inspecting the Content Dictionary specified. These include not only the symbol definition, but also examples and other formal properties. Note, however, that there are multiple encodings for OpenMath Content Dictionaries, and it is up to the user agent to correctly determine the encoding when retrieving a CD.

4.2.3.2 Extended uses of `<csymbol>`

In addition to the forms described above, the `csymbol` and element can contain `mglyph` elements to refer to characters not currently available in Unicode, or a general presentation construct (see Section 3.1.9), which is used for rendering (see Section 4.1.2).

External definitions (in OpenMath CDs or elsewhere) may also be specified directly for a `csymbol` using the `definitionURL` attribute. When used to reference OpenMath symbol definitions, the abstract triple of (symbol name, CD name, CD base) is mapped to a fully-qualified URI as follows:

`{URI = }`*cdbase*`{ + '/' + }`*cd* − *name*`{ + '#' + }`*symbol* − *name*

For example,

`(plus, arith1, http://www.openmath.org/cd)`

is mapped to

`{http://www.openmath.org/cd/arith1#plus}`

Editor's note: MiKo: I thought we got rid of cdbase (David: it's not an attribute, but is in the abstract openmath model)

The resulting URI is specified as the value of the `definitionURL` attribute.

This form of reference is useful for backwards compatibility with MathML2 and to facilitate the use of Content MathML within URI-based frameworks (such as RDF [rdf] in the Semantic Web or OMDoc [OMDoc1.2]). Another benefit is that the symbol name in the CD does not need to correspond to the content of the `csymbol` element. However, in general, this method results in much longer MathML instances. Also, in situations where CDs are under development, the use of a CD Group file allows the locations of CDs to change without a change to the markup. A third drawback to `definitionURL` is that unlike the `cd` attribute, it is not limited to referencing symbol definitions in OpenMath content dictionaries. Hence, it is not in general possible for a user agent to automatically determine the proper interpretation for `definitionURL` values without further information about the context and community of practice in which the MathML instance occurs.

Both the `cd` and `definitionURL` mechanisms of external reference may be used within a single MathML instance. However, when both a `cd` and a `definitionURL` attribute are specified on a single `csymbol`, the `cd` attribute takes precedence.

4.2.3.3 Rendering Symbols

If the content of a `csymbol` element is tagged using presentation tags, that presentation is used. If no such tagging is supplied then the text content is rendered as if it were the content of an `mi` element. In particular if an application supports bidirectional text rendering, then the rendering follows the Unicode bidirectional rendering.

4.2.4 String Literals `<cs>`

	Schema Fragment
Class	Cs
Attributes	CommonAtt
Content	text

The `cs` element encodes 'string literals' which may be used in Content MathML expressions.

The content of `cs` is text. Unlike other token elements `cs` may not contain `mglyph` or other Presentation MathML constructs, and the content does not undergo white space normalisation.

Content MathML

```
<set>
  <cs>A</cs><cs>B</cs><cs>  </cs>
</set>
```

Sample Presentation

```
<mrow>
 <mo>{</mo>
 <ms>A</ms>
 <mo>,</mo>
 <ms>B</ms>
 <mo>,</mo>
 <ms>  </ms>
 <mo>}</mo>
</mrow>
```

{"A","B"," "}

4.2.5 Function Application `<apply>`

	Schema Fragment (Strict)	Schema Fragment (Full)
Class	Apply	Apply
Attributes	CommonAtt	CommonAtt
Content	ContExp+	ContExp+ | ContExp, BVar, Qualifier?, ContExp+

The most fundamental way of building a compound object in mathematics is by applying a function or an operator to some arguments.

4.2.5.1 Strict Content MathML

In MathML, the `apply` element is used to build an expression tree that represents the result of applying a function or operator to its arguments. The resulting tree corresponds to a complete mathematical expression. Roughly speaking, this means a piece of mathematics that could be surrounded by parentheses or 'logical brackets' without changing its meaning.

For example, $(x + y)$ might be encoded as

```
<apply><csymbol cd="arith1">plus</csymbol><ci>x</ci><ci>y</ci></apply>
```

The opening and closing tags of `apply` specify exactly the scope of any operator or function. The most typical way of using `apply` is simple and recursive. Symbolically, the content model can be described as:

<apply> *op* [*a* *b* ...] </apply>

where the *operands* *a*, *b*, ... are MathML expression trees themselves, and *op* is a MathML expression tree that represents an operator or function. Note that `apply` constructs can be nested to arbitrary depth.

An `apply` may in principle have any number of operands. For example, $(x + y + z)$ can be encoded as

```
<apply><csymbol cd="arith1">plus</csymbol>
  <ci>x</ci>
  <ci>y</ci>
  <ci>z</ci>
</apply>
```

Note that MathML also allows applications without operands, e.g. to represent functions like `random()`, or `current-date()`.

Mathematical expressions involving a mixture of operations result in nested occurrences of `apply`. For example, $a\,x + b$ would be encoded as

```
<apply><csymbol cd="arith1">plus</csymbol>
  <apply><csymbol cd="arith1">times</csymbol>
    <ci>a</ci>
    <ci>x</ci>
  </apply>
  <ci>b</ci>
</apply>
```

There is no need to introduce parentheses or to resort to operator precedence in order to parse expressions correctly. The `apply` tags provide the proper grouping for the re-use of the expressions within other constructs. Any expression enclosed by an `apply` element is well-defined, coherent object whose interpretation does not depend on the surrounding context. This is in sharp contrast to presentation markup, where the same expression may have very different meanings in different contexts. For example, an expression with a visual rendering such as $(F+G)(x)$ might be a product, as in

```
<apply><csymbol cd="arith1">times</csymbol>
  <apply><csymbol cd="arith1">plus</csymbol>
    <ci>F</ci>
    <ci>G</ci>
  </apply>
  <ci>x</ci>
</apply>
```

or it might indicate the application of the function $F + G$ to the argument x. This is indicated by constructing the sum

```
<apply><csymbol cd="arith1">plus</csymbol><ci>F</ci><ci>G</ci></apply>
```

and applying it to the argument x as in

```
<apply>
  <apply><csymbol cd="arith1">plus</csymbol>
    <ci>F</ci>
    <ci>G</ci>
  </apply>
  <ci>x</ci>
</apply>
```

In both cases, the interpretation of the outer `apply` is explicit and unambiguous, and does not change regardless of where the expression may be reused.

The preceding example also illustrates that in an `apply` construct, both the function and the arguments may be simple identifiers or more complicated expressions.

The `apply` element is conceptually necessary in order to distinguish between a function or operator, and an instance of its use. The expression constructed by applying a function to 0 or more arguments is always an element from the codomain of the function. Proper usage depends on the operator that is being applied. For example, the `plus` operator may have zero or more arguments, while the `minus` operator requires one or two arguments in order to be properly formed.

4.2.5.2 Rendering Applications

Strict Content MathML applications are rendered as mathematical function applications:

If F is the rendering of f and Ai those of ai .

```
<apply> f
   a1
   a2
   ...
   an
</apply>
```

Sample Presentation

```
<mrow>
  F
 <mo>&#x2061;</mo>
 <mrow>
  <mo fence="true">(</mo>
   A1
  <mo separator="true">,</mo>
   ...
  <mo separator="true">,</mo>
   A2
  <mo separator="true">,</mo>
   An
  <mo fence="true">)</mo>
 </mrow>
</mrow>
```

MathML applications may be used with qualifiers. In the absence of any more specific rendering rules, the proposed presentation in such cases is to follow the layout used for `sum`. So for example:

Content MathML

```
<apply> op
  <bvar> x </bvar>
  <domainofapplication> d </domainofapplication>
   expression-in-x
</apply>
```

Sample Presentation

```
<mrow>
 <munder>
   OP
  <mrow> X <mo>&#x2208;</mo> D </mrow>
 </munder>
 <mo>&#x2061;</mo>
 <mrow>
  <mo fence="true">(</mo>
   Expression-in-X
  <mo fence="true">)</mo>
 </mrow>
</mrow>
```

4.2.6 Bindings and Bound Variables <bind> and <bvar>

Many complex mathematical expressions are constructed with the use of bound variables, and bound variables are an important concept of logic and formal languages. Variables become *bound* in the scope of an expression through the use of a quantifier. Informally, they can be thought of as the "dummy variables" in expressions such as integrals, sums, products, and the logical quantifiers "for all" and "there exists". A bound variable is characterized by the property that systematically renaming the variable (to a name not already appearing in the expression) does not change the meaning of the expression.

4.2.6.1 Bindings

	Schema Fragment (Strict)	Schema Fragment (Full)
Class	Bind	Bind
Attributes	CommonAtt	CommonAtt
Content	ContExp, BVar*, ContExp	ContExp, BVar*, Qualifier*, ContExp+

Binding expressions are represented as MathML expression trees using the `bind` element. Its first child is a MathML expression that represents a binding operator (the integral operator in our example). This is followed by a non-empty list of `bvar` elements denoting the bound variables, and then the final child which is a general Content MathML expression, known as the *body* of the binding.

4.2.6.2 Bound Variables

	Schema Fragment (Strict)	Schema Fragment (Full)
Class	BVar	BVar
Attributes	CommonAtt	CommonAtt
Content	AnnVar	AnnVar,degree | degree,AnnVar

The `bvar` element is used to denote the bound variable of a binding expression, e.g. in sums, products, and quantifiers or user defined functions.

The content of a bvar element is an *annotated variable*, i.e. either a content identifier represented by a ci element or a semantics element whose first child is an annotated variable. The *name* of an annotated variable of the second kind is the name of its first child. The *name* of a bound variable is that of the annotated variable in the bvar element.

Bound variables are identified by comparing their names. Such identification can be made explicit by placing an id on the ci element in the bvar element and referring to it using the xref attribute on all other instances. An example of this approach is

```
<bind><csymbol cd="quant1">forall</csymbol>
  <bvar><ci id="var-x">x</ci></bvar>
  <apply><csymbol cd="relation1">lt</csymbol>
    <ci xref="var-x">x</ci>
    <cn>1</cn>
  </apply>
</bind>
```

This id based approach is especially helpful when constructions involving bound variables are nested.

It is sometimes necessary to associate additional information with a bound variable. The information might be something like a detailed mathematical type, an alternative presentation or encoding or a domain of application. Such associations are accomplished in the standard way by replacing a ci element (even inside the bvar element) by a semantics element containing both the ci and the additional information. Recognition of an instance of the bound variable is still based on the actual ci elements and not the semantics elements or anything else they may contain. The id based-approach outlined above may still be used.

The following example encodes forall x. $x+y=y+x$.

```
<bind><csymbol cd="quant1">forall</csymbol>
  <bvar><ci>x</ci></bvar>
  <apply><csymbol cd="relation1">eq</csymbol>
    <apply><csymbol cd="arith1">plus</csymbol><ci>x</ci><ci>y</ci></apply>
    <apply><csymbol cd="arith1">plus</csymbol><ci>y</ci><ci>x</ci></apply>
  </apply>
</bind>
```

In non-Strict Content markup, the bvar element is used in a number of idiomatic constructs. These are described in Section 4.3.3 and Section 4.4.

4.2.6.3 Renaming Bound Variables

It is a defining property of bound variables that they can be renamed consistently in the scope of their parent bind element. This operation, sometimes known as α-*conversion*, preserves the semantics of the expression.

A bound variable x may be renamed to say y so long as y does not occur free in the body of the binding, or in any annotations of the bound variable, x to be renamed, or later bound variables.

If a bound variable x is renamed, all free occurrences of x in annotations in its bvar element, any following bvar children of the bind and in the expression in the body of the bind should be renamed.

In the example in the previous section, note how renaming x to z produces the equivalent expression forall z. $z+y=y+z$, whereas x may not be renamed to y, as y is free in the body of the binding and would be *captured*, producing the expression forall y. $y+y=y+y$ which is not equivalent to the original expression.

4.2.6.4 Rendering Binding Constructions

If *b* and *s* are Content MathML expressions that render as the Presentation MathML expressions *B* and *S* then the sample rendering of a binding element is as follows:

Content MathML

```
<bind> b
  <bvar> x1 </bvar>
  <bvar> ... </bvar>
  <bvar> xn </bvar>
   s
</bind>
```

Sample Presentation

```
<mrow>
  B
 <mrow>
   x1
  <mo separator="true">,</mo>
   ...
  <mo separator="true">,</mo>
   xn
 </mrow>
 <mo separator="true">.</mo>
  S
</mrow>
```

4.2.7 Structure Sharing <share>

To conserve space in the XML encoding, MathML expression trees can make use of structure sharing.

4.2.7.1 The share element

	Schema Fragment
Class	Share
Attributes	CommonAtt, href
href Attribute Values	*URI*
Content	Empty

The share element has an href attribute used to to reference a MathML expression tree. The value of the href attribute is a URI specifying the id attribute of the root node of the expression tree. When building a MathML expression tree, the share element is replaced by a copy of the MathML expression tree referenced by the href attribute. Note that this copy is *structurally equal*, but not identical to the element referenced. The values of the share will often be relative URI references, in which case they are resolved using the base URI of the document containing the share element.

Issue (): In order to get parallel markup working, we might want to introduce a sharing element for Presentation MathML as well. That would also potentially give us size benefits.

Resolution: The WG decided on the Boston F2F that we do not want sharing in presentation (too complicated with all the inherited elements

For instance, the mathematical object $f(f(f(a,a),f(a,a)),f(f(a,a),f(a,a)))$ can be encoded as either one of the following representations (and some intermediate versions as well).

```
<apply><ci>f</ci>
  <apply><ci>f</ci>
    <apply><ci>f</ci>
      <ci>a</ci>
      <ci>a</ci>
    </apply>
    <apply><ci>f</ci>
      <ci>a</ci>
      <ci>a</ci>
    </apply>
  </apply>
  <apply><ci>f</ci>
    <apply><ci>f</ci>
      <ci>a</ci>
      <ci>a</ci>
    </apply>
    <apply><ci>f</ci>
      <ci>a</ci>
      <ci>a</ci>
    </apply>
  </apply>
</apply>
```

```
<apply><ci>f</ci>
  <apply id="t1"><ci>f</ci>
    <apply id="t11"><ci>f</ci>
      <ci>a</ci>
      <ci>a</ci>
    </apply>
    <share href="#t11"/>
  </apply>
  <share href="#t1"/>
</apply>
```

4.2.7.2 An Acyclicity Constraint

Say that an element *dominates* all its children and all elements they dominate. Say also that a `share` element dominates its target, i.e. the element that carries the `id` attribute pointed to by the `href` attribute. For instance in the representation on the right above, the `apply` element with `id="t1"` and also the second `share` (with `href="t11"`) both dominate the `apply` element with `id="t11"`.

The occurrences of the `share` element must obey the following global *acyclicity constraint*: An element may not dominate itself. For example, the following representation violates this constraint:

```
<apply id="badid1"><csymbol cd="arith1">divide</csymbol>
  <cn>1</cn>
  <apply><csymbol cd="arith1">plus</csymbol>
    <cn>1</cn>
    <share href="#badid1"/>
  </apply>
</apply>
```

Here, the `apply` element with `id="foo"` dominates its third child, which dominates the `share` element, which dominates its target: the element with `id="foo"`. So by transitivity, this element dominates itself. By the acyclicity constraint, the example is not a valid MathML expression tree. It might be argued that such an expression could be given the interpretation of the continued fraction $\frac{1}{1+\frac{1}{1+\frac{1}{1+\ddots}}}$. However, the procedure of building an expression tree by replacing `share` element does not terminate for such an expression, and hence such expressions are not allowed by Content MathML.

Note that the acyclicity constraints is not restricted to such simple cases, as the following example shows:

```
<apply id="bar">                          <apply id="baz">
  <csymbol cd="arith1">plus</csymbol>       <csymbol cd="arith1">plus</csymbol>
  <cn>1</cn>                                <cn>1</cn>
  <share href="#baz"/>                      <share href="#bar"/>
</apply>                                  </apply>
```

Here, the `apply` with `id="bar"` dominates its third child, the `share` with `href="#baz"`. That element dominates its target `apply` (with `id="baz"`), which in turn dominates its third child, the `share` with `href="#bar"`. Finally, the `share` with `href="#bar"` dominates its target, the original `apply` element with `id="bar"`. So this pair of representations ultimately violates the acyclicity constraint.

4.2.7.3 *Structure Sharing and Binding*

Note that the `share` element is a *syntactic* referencing mechanism: a `share` element stands for the exact element it points to. In particular, referencing does not interact with binding in a semantically intuitive way, since it allows a phenomenon called *variable capture* to occur. Consider an example:

```
<bind id="outer"><csymbol cd="fns1">lambda</csymbol>
  <bvar><ci>x</ci></bvar>
  <apply><ci>f</ci>
    <bind id="inner"><csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <share id="copy" href="#orig"/>
    </bind>
    <apply id="orig"><ci>g</ci><ci>x</ci></apply>
  </apply>
</bind>
```

This represents a term $\lambda x.f(\lambda x.g(x), g(x))$ which has two sub-terms of the form $g(x)$, one with `id="orig"` (the one explicitly represented) and one with `id="copy"`, represented by the `share` element. In the original, explicitly-represented term, the variable x is bound by the *outer* `bind` element. However, in the copy, the variable x is bound by the *inner* `bind` element. One says that the inner `bind` has captured the variable x.

Using references that capture variables in this way can easily lead to representation errors, and is not recommended. For instance, using α-conversion to rename the inner occurrence of x into, say, y leads to the semantically equivalent expression $\lambda x.f(\lambda y.g(y), g(x))$. However, in this form, it is no longer possible to share the expression $g(x)$. Replacing x with y in the inner `bvar` without replacing the `share` element results in a change in semantics.

4.2.7.4 *Rendering Expressions with Structure Sharing*

The default rendering of a `share` is that of the MathML element pointed to by the URI in the `href` attribute.

4.2.8 Attribution via `semantics`

	Schema Fragment (Strict)	Schema Fragment (content MathML)
Class	Semantics	Semantics
Attributes		definitionURL?, encoding?
Content	ContExp, (annotation | annotation-xml)*	ContExp, (annotation | annotation-xml)*

Content elements can be adorned with additional information via the `semantics` element. An *annotation* decorates a Content MathML expression with a sequence of one or more semantic annotations.

MathML uses the `semantics` element to wrap the annotated element and the `annotation-xml` and `annotation` elements for representing the annotations themselves.

	Schema Fragment (Strict)	Schema Fragment (content MathML)
Class	Annotation	Annotation
Attributes	cd name href?	definitionURL? encoding? cd? name? href? clipboardflavor?
Content	text	text

	Schema Fragment (Strict)	Schema Fragment (content MathML)
Class	AnnotationXML	AnnotationXML
Attributes	cd name href?	definitionURL? encoding? cd? name? href? clipboardflavor?
Content	ANY	ANY

As such, the `semantics` element should be considered part of both presentation MathML and Content MathML. MathML considers a `semantics` element (strict) Content MathML, if and only if its first child is (strict) Content MathML. All MathML processors should process the `semantics` element, even if they only process one of those subsets.

Each annotation has `cd`, and `name` attributes to specify the *key*, i.e. a symbol that specifies the relation between the annotated object and the annotation; See Section 5.1 for details.

An annotation acts as either adornment annotation or as semantic annotation, depending on the role of the key symbol is given by its content dictionary

4.2.8.1 Semantic annotations

When the key has role `"semantic-attribution"` then the annotated object is modified by the annotation and dropping it changes the semantics.

An example of the use of a semantic attribution would be to indicate the type of an object. For example the following expression associates with an identifier F the information that it represents an operator that takes real numbers as input and returns natural numbers as values (the absolute value function is an example of such a function).

```
<semantics>
  <ci>F</ci>
  <annotation-xml cd="mathmltypes" name="type" encoding="MathML Content">
    <apply><csymbol cd="mathmltypes">fun_type</csymbol>
      <csymbol cd="setname1">Z</csymbol>
      <csymbol cd="setname1">N</csymbol>
    </apply>
  </annotation-xml>
</semantics>
```

Here we have assumed the existence of a content dictionary `types` that provides a key symbol `type` that specifies that the attributed expression is of the type specified by the Content MathML expression in the `annotation-xml` element. The key is specified by the `cd` and `name` attributes in the `attribution-xml` element. The `encoding` attribute on the `annotation-xml` element specifies the format of the XML data.

Issue (): The functionality of `semantics` together with `annotation` is very similar to the one given by the OpenMath style `attribution` and `foreign` elements. At least if we make the `definitionURL` attribute mandatory on `annotation`, as we had planned for MathML2(2e), but forgot (the types note depends on this). The Difference then is largely in the way the key is addressed, and what we say about the semantics of attributions (does the order play a role, how about duplicates,

interaction with alpha renaming,...); some of this is still not fully solved in OpenMath yet, but on the agenda. We should decide for one of the possibilities and consolidate the rest.

Resolution: We have decided to go only with semantics and upgrade it so that it is openmath-compatible.

4.2.8.2 Adornment annotations

When the key symbol has role `"attribution"` in the content dictionary, then an annotation with this key is an *adornment annotation* and dropping the annotation is not harmful and preserves the semantics. If the key symbol lacks the role specification then attribution is acting as adornment annotation.

An example of the use of an adornment attribution would be to indicate the color in which a Content MathML expression *A* should be displayed, for example

```
<semantics>
   A
  <annotation-xml cd="display" name="color" encoding="MathML Presentation">
     red
  </annotation-xml>
</semantics>
```

Note *red* are arbitrary representations whereas the key is a symbol.

4.2.8.3 Rendering Annotations

The default rendering of a `semantics` element is the default rendering of its first child possibly augmented with default renderings of the semantic annotations depending on the key symbol; adornment annotations are not rendered by default.

When a Presentation MathML annotation is provided, a MathML renderer may optionally use this information to render the MathML construct. This would typically be the case when the first child is a MathML content construct and the annotation is provided to give a preferred rendering differing from the default for the content elements.

4.2.9 Error Markup `<cerror>`

	Schema Fragment (Strict)
Class	Error
Attributes	CommonAtt
Content	Symbol, ContExp*

A content error expression is made up of a symbol and a sequence of zero or more MathML expression trees. The initial symbol indicates the kind of error. The `cerror` object has no direct mathematical meaning. Errors occur as the result of some action performed on an expression tree and are thus of real interest only when some sort of communication is taking place. Errors may occur inside other objects and also inside other errors.

As an example, to encode a division by zero error, one might employ a hypothetical `aritherror` Content Dictionary with a `DivisionByZero` symbol, as in the following expression tree:

```
<cerror>
  <csymbol cd="aritherror">DivisionByZero</csymbol>
  <apply><csymbol cd="arith1">divide</csymbol><ci>x</ci><cn>0</cn></apply>
</cerror>
```

Note that error markup generally should enclose only the smallest erroneous sub-expression. Thus a `cerror` will often be a sub-expression of a bigger one, e.g.

```
<apply><csymbol cd="relation1">eq</csymbol>
  <cerror>
    <csymbol cd="aritherror">DivisionByZero</csymbol>
    <apply><csymbol cd="arith1">divide</csymbol><ci>x</ci><cn>0</cn></apply>
  </cerror>
  <cn>0</cn>
</apply>
```

If an application wishes to signal that a Content MathML expression it has received is syntactically invalid or is not well-formed, the offending data must be encoded as a string. For example:

```
<cerror>
  <csymbol cd="parser">invalid_XML</csymbol>
  <mtext> &lt;apply&gt;&lt;cos&gt; &lt;ci&gt;v&lt;/ci&gt; &lt;/apply&gt; </mtext>
</cerror>
```

Note that the < and > characters have been escaped as is usual in an XML document.

The default presentation of a `cerror` element is a `merror` expression, where the first child of the `merror` is a presentation of the first child of the `cerror` expression and and the remaining children are passed on for reference. For instance the presentation of the example above could be

```
<merror>
  <mtext>Division by zero</mtext>
  <apply><csymbol cd="arith1">divide</csymbol><ci>x</ci><cn>0</cn></apply>
</merror>
```

Editor's note: David: shouldn't this be as below, with slight wording changes in the above para to match? should probably be made into a "boxed triple, cerror, merror and an image so the pmml and image can be mechanically checked.

```
<merror>
  <mtext>Division by zero:
  &lt;apply>&lt;csymbol cd="arith1">divide&lt;/csymbol>&lt;ci>x&lt;/ci>&lt;cn>0&lt;/cn>
 &lt;/apply>
</mtext>
</merror>
```

4.2.10 Encoded Bytes `<cbytes>`

	Schema Fragment
Class	Cbytes
Attributes	CommonAtt
Content	base64

The content of `cbytes` represents a stream of bytes as a sequence of characters in Base64 encoding, that is it matches the base64Binary data type defined in [XMLSchemaDatatypes]. All white space is ignored.

The `cbytes` element is mainly used for OpenMath compatibility, but may be used, as in OpenMath, to encapsulate output from a system that may be hard to encode in MathML, such as binary data relating to the internal state of a system, or image data.

The rendering of `cbytes` is not expected to represent the content and the proposed rendering is that of an empty `mrow`. Typically `cbytes` is used in an `annotation-xml` or is itself annotated with Presentation MathML, so this default rendering should rarely be used.

4.3 Content MathML for Specific Structures

The elements of Strict Content MathML described in the previous section are sufficient to encode logical assertions and expression structure, and they do so in a way that closely models the standard constructions of mathematical logic that underlie the foundations of mathematics. As a consequence, Strict markup can be used to represent all of mathematics, and is ideal for providing consistent mathematical semantics for all Content MathML expressions.

At the same time, many notational idioms of mathematics are not straightforward to represent directly with Strict Content markup. For example, standard notations for sums, integrals, sets, piecewise functions and many other common constructions require non-obvious technical devices, such as the introduction of lambda functions, to rigorously encode them using Strict markup. Consequently, in order to make Content MathML easier to use, a range of additional elements have been provided for encoding such idiomatic constructs more directly. This section discusses the general approach for encoding such idiomatic constructs, and their Strict Content equivalents. Specific constructions are discussed in detail in Section 4.4.

Most idiomatic constructions which Content markup addresses fall into about a dozen classes. Some of these classes, such as *container elements*, have their own syntax. Similarly, a small number of non-Strict constructions involve a single element with an exceptional syntax, for example `partialdiff`. These exceptional elements are discussed on a case-by-case basis in Section 4.4. However, the majority of constructs consist of classes of operator elements which all share a particular usage of *qualifiers*. These classes of operators are described in Section 4.3.4.

In all cases, non-Strict expressions may be rewritten using only Strict markup. In most cases, the transformation is completely algorithmic, and may be automated. Rewrite rules for classes of non-Strict constructions are introduced and discussed later in this section, and rewrite rules for exceptional constructs involving a single operator are given in Section 4.4. The complete algorithm for rewriting arbitrary Content MathML as Strict Content markup is summarized at the end of the Chapter in Section 4.6.

4.3.1 Container Markup

Many mathematical structures are constructed from subparts or parameters. The motivating example is a set. Informally, one thinks of a set as a certain kind of mathematical object that contains a collection of elements. Thus, it is intuitively natural for the markup for a set to contain, in the XML sense, the markup for its constituent elements. This style of representation is termed *container markup* in MathML. By contrast, Strict markup typically represents an instance of a set as the result of applying a function (or more generally a *constructor symbol*) to arguments.

While the two approaches are formally equivalent, container markup is generally more intuitive for non-expert authors to use, while Strict markup is preferable is contexts where semantic rigor is paramount. In addition, MathML 2 relied on container markup, and thus container markup is necessary in cases where backward compatibility is required.

MathML provides container markup for the following mathematical constructs: sets, lists, intervals, vectors, matrices (two elements), piecewise functions (three elements) and lambda functions. There are corresponding constructor symbols in Strict markup for each of these, with the exception of lambda

functions, which correspond to binding symbols in Strict markup. Note that in MathML 2, the term "container markup" was also taken to include token elements, and the deprecated `declare`, `fn` and `reln` elements, but MathML 3 limits usage of the term to the above constructs.

The rewrite rules for obtaining equivalent Strict Content markup from container markup depend on the operator class of the particular operator involved. For details about a specific container element, obtain its operator class (and any applicable special case information) by consulting the syntax table and discussion for that element in Section 4.4. Then apply the rewrite rules for that specific operator class as described in Section 4.3.4.

4.3.1.1 Container Markup for Constructor Symbols

The arguments to container elements corresponding to constructors may either be explicitly given as a sequence of child elements, or they may be specified by a rule using qualifiers. The only exceptions are the `piecewise`, `piece`, and `otherwise` elements used for representing functions with piecewise definitions. The arguments of these elements must always be specified explicitly.

Here is an example of container markup with explicitly specified arguments:

```
<set><ci>a</ci><ci>b</ci><ci>c</ci></set>
```

This is equivalent to the following Strict Content MathML expression:

```
<apply><csymbol cd="set1">set</csymbol><ci>a</ci><ci>b</ci><ci>c</ci></apply>
```

Another example of container markup, where the list of arguments is given indirectly as an expression with a bound variable. The container markup for the set of even integers is:

```
<set>
  <bvar><ci>x</ci></bvar>
  <domainofapplication><integers/></domainofapplication>
  <apply><times/><cn>2</cn><ci>x</ci></apply>
</set>
```

This may be written as follows in Strict Content MathML:

```
<apply><csymbol cd="set1">map</csymbol>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>x</ci></bvar>
    <apply><csymbol cd="arith1">times</csymbol>
      <cn>2</cn>
      <ci>x</ci>
    </apply>
  </bind>
  <csymbol cd="setname1">Z</csymbol>
</apply>
```

Issue (): Do we want to prescribe one of the representations for the DOM? That would make the processing much simpler.

Resolution: We have decided to keep the MathML DOM directly in equivalent to the XML DOM of this, then this becomes a non-issue

4.3.1.2 Container Markup for Binding Constructors

The `lambda` element is a container element corresponding to the lambda symbol in the fns1 Content Dictionary. However, unlike the container elements of the preceding section, which purely construct

mathematical objects from arguments, the `lambda` element performs variable binding as well. Therefore, the child elements of `lambda` have distinguished roles. In particular, a `lambda` element must have at least one `bvar` child, optionally followed by qualifier elements, followed by a Content MathML element. This basic difference between the `lambda` container and the other constructor container elements is also reflected in the OpenMath symbols to which they correspond. The constructor symbols have an OpenMath role of "application", while the lambda symbol has a role of "bind".

This example shows the use of `lambda` container element and the equivalent use of `bind` in Strict Content MathML

```
<lambda><bvar><ci>x</ci></bvar><ci>x</ci></lambda>
```

```
<bind><csymbol cd="fns1">lambda</csymbol>
 <bvar><ci>x</ci></bvar><ci>x</ci>
</bind>
```

4.3.2 Bindings with `<apply>`

MathML allows the use of the `apply` element to perform variable binding in non-Strict constructions instead of the `bind` element. This usage conserves backwards compatibility with MathML 2. It also simplifies the encoding of several constructs involving bound variables with qualifiers as described below.

Use of the `apply` element to bind variables is allowed in two situations. First, when the operator to be applied is itself a binding operator, the `apply` element merely substitutes for the `bind` element. The logical quantifiers `<forall/>`, `<exists/>` and the container element `lambda` are the primary examples of this type.

The second situation arises when the operator being applied allows the use of bound variables with qualifiers. The most common examples are sums and integrals. In most of these cases, the variable binding is to some extent implicit in the notation, and the equivalent Strict representation requires the introduction of auxiliary constructs such as lambda expressions for formal correctness.

Because expressions using bound variables with qualifiers are idiomatic in nature, and do not always involve true variable binding, one cannot expect systematic renaming (alpha-conversion) of variables "bound" with `apply` to preserve meaning in all cases. An example for this is the `diff` element where the `bvar` term is technically not bound at all.

The following example illustrates the use of `apply` with a binding operator. In these cases, the corresponding Strict equivalent merely replaces the `apply` element with a `bind` element:

```
<apply><forall/>
  <bvar><ci>x</ci></bvar>
  <apply><geq/><ci>x</ci><ci>x</ci></apply>
</apply>
```

The equivalent Strict expression is:

```
<bind><csymbol cd="logic1">forall</csymbol>
  <bvar><ci>x</ci></bvar>
  <apply><csymbol cd="relation1">geq</csymbol><ci>x</ci><ci>x</ci></apply>
</bind>
```

In this example, the sum operator is not itself a binding operator, but bound variables with qualifiers are implicit in the standard notation, which is reflected in the non-Strict markup. In the equivalent Strict representation, it is necessary to convert the summand into a lambda expression, and recast the qualifiers as an argument expression:

```
<apply><sum/>
  <bvar><ci>i</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <uplimit><cn>100</cn></uplimit>
  <apply><power/><ci>x</ci><ci>i</ci></apply>
</apply>
```

The equivalent Strict expression is:

```
<apply><csymbol cd="arith1">sum</csymbol>
  <apply><csymbol cd="interval1">integer_interval</csymbol>
    <cn>0</cn>
    <cn>100</cn>
  </apply>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>i</ci></bvar>
    <apply><csymbol cd="arith1">power</csymbol>
      <ci>x</ci>
      <ci>i</ci>
    </apply>
  </bind>
</apply>
```

4.3.3 Qualifiers

Many common mathematical constructs involve an operator together with some additional data. The additional data is either implicit in conventional notation, such as a bound variable, or thought of as part of the operator, as is the case with the limits of a definite integral. MathML 3 uses *qualifier* elements to represent the additional data in such cases.

Qualifier elements are always used in conjunction with operator or container elements. Their meaning is idiomatic, and depends on the context in which they are used. When used with an operator, qualifiers always follow the operator and precede any arguments that are present. In all cases, if more than one qualifier is present, they appear in the order `bvar`, `lowlimit`, `uplimit`, `interval`, `condition`, `domainofapplication`, `degree`, `momentabout`, `logbase`.

The precise function of qualifier elements depends on the operator or container that they modify. The majority of use cases fall into one of several categories, discussed below, and usage notes for specific operators and qualifiers are given in Section 4.4.

4.3.3.1 *Uses of <domainofapplication>, <interval>, <condition>, <lowlimit> and <uplimit>*

The primary use of `domainofapplication`, `interval`, `uplimit`, `lowlimit` and `condition` is to restrict the values of a bound variable. The most general qualifier is `domainofapplication`. It is used to specify a set (perhaps with additional structure, such as an ordering or metric) over which an operation is to take place. The `interval` qualifier, and the pair `lowlimit` and `uplimit` also restrict

a bound variable to a set in the special case where the set is an interval. The `condition` qualifier, like `domainofapplication`, is general, and can be used to restrict bound variables to arbitrary sets. However, unlike the other qualifiers, it restricts the bound variable by specifying a Boolean-valued function of the bound variable. Thus, `condition` qualifiers always contain instances of the bound variable, while the other qualifier usually do not. The other qualifiers may even be used when no variables are being bound, e.g. to indicate the restriction of a function to a subdomain.

In most cases, any of the qualifiers capable of representing the domain of interest can be used interchangeably. The most qualifier general is `domainofapplication`, and it has a priveledged role. It is the preferred form, unless there are particular idiomatic reasons to use one of the other qualifier, e.g. limits for an integral. In MathML 3, the other forms are treated as shorthand notations `domainofapplication`, because they may all be rewritten as equivalent `domainofapplication` constructions. The rewrite rules to do this given below. The other qualifer elements are provided because they correspond to common notations and map more easily to familiar presentations. Therefore, in the situations where they naturally arise, they may be more convenient and direct than `domainofapplication`. Note, however, that only one of `domainofapplication`, `interval`,`condition` or the pair `uplimit` and `lowlimit` should be used in a single expression, since these qualifiers all serve essentially the same purpose.

To illustrate these ideas, consider the following examples showing alternative representations of a definite integral. Let C denote the interval from 0 to 1, and $f(x) = x^2$. Then `domainofapplication` could be used express the integral of a f over C in this way:

```
<apply><int/>
  <domainofapplication>
    <ci type="set">C</ci>
  </domainofapplication>
  <ci type="function">f</ci>
</apply>
```

Note that no explicit bound variable is identified in this encoding. Alternatively, the `interval` qualifier could be used with an explicit bound variable:

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <interval><cn>0</cn><cn>1</cn></interval>
  <apply><power/><ci>x</ci><cn>2</cn></apply>
</apply>
```

The pair `lowlimit` and `uplimit` can also be used. This is perhaps the most "standard" representation of this integral:

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <uplimit><cn>1</cn></uplimit>
  <apply><power/><ci>x</ci><cn>2</cn></apply>
</apply>
```

Finally, here is the same integral, represented using a `condition` on the bound variable:

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><and/>
      <apply><leq/><cn>0</cn><ci>x</ci></apply>
```

```
      <apply><leq/><ci>x</ci><cn>1</cn></apply>
    </apply>
  </condition>
  <apply><power/><ci>x</ci><cn>2</cn></apply>
</apply>
```

Note the use of the explicit bound variable within the `condition` term.

The general technique of using a `condition` element together with `domainofapplication` is quite powerful. For example, to extend the previous example to a multivariate domain, one may use an extra bound variable and a domain of application corresponding to a cartesian product:

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <domainofapplication>
    <set>
      <bvar><ci>t</ci></bvar>
      <bvar><ci>u</ci></bvar>
      <condition>
        <apply><and/>
          <apply><leq/><cn>0</cn><ci>t</ci></apply>
          <apply><leq/><ci>t</ci><cn>1</cn></apply>
          <apply><leq/><cn>0</cn><ci>u</ci></apply>
          <apply><leq/><ci>u</ci><cn>1</cn></apply>
        </apply>
      </condition>
      <list><ci>t</ci><ci>u</ci></list>
    </set>
  </domainofapplication>
  <apply><times/>
    <apply><power/><ci>x</ci><cn>2</cn></apply>
    <apply><power/><ci>y</ci><cn>3</cn></apply>
  </apply>
</apply>
```

Note that the order of the inner and outer bound variables is significant.

Mappings to Strict Content MathML

When rewriting expressions to Strict Content MathML, qualifier elements are removed via a series of rules described in this section. The general algorithm for rewriting a MathML expression involving qualifiers proceeds in two steps. First, constructs using the `interval`, `condition`, `uplimit` and `lowlimit` qualifiers are converted to constructs using only `domainofapplication`. Second, `domainofapplication` expressions are then rewritten as Strict Content markup.

Rewrite: interval qualifier

```
<apply> H
  <bvar> x </bvar>
  <lowlimit> a </lowlimit>
  <uplimit> b </uplimit>
   C
</apply>
```

```
<apply> H
  <bvar> x </bvar>
  <domainofapplication>
    <apply><csymbol cd="interval1">interval</csymbol>
       a
       b
    </apply>
  </domainofapplication>
   C
</apply>
```

The symbol used in this translation depends on the head of the application, denoted by `H` here. By default interval should be used (which is explictly for intervals of underdefined properties). However for the predefined eleents on MathML, more specific interval symbols can be used. If the head is `int` then ordered_interval, for `sum` and `product` integer_interval should be used.

The above technique for replacing `lowlimit` and `uplimit` qualifiers with a `domainofapplication` element is also used for replacing the `interval` qualifier.

The `condition` qualifier restricts a bound variable by specifying a Boolean-valued expression on a larger domain, specifying whether a given value is in the restricted domain. The `condition` element contains a single child that represents the truth condition. Compound conditions are formed by applying Boolean operators such as `and` in the condition.

Rewrite: condition

To rewrite an expression using the `condition` qualifier as one using `domainofapplication`,

```
<bvar> x1 </bvar>
<bvar> xn </bvar>
<condition> P </condition>
```

is rewritten to

```
<domainofapplication>
  <apply><csymbol cd="set1">suchthat</csymbol>
     R
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar> x1 </bvar>
      <bvar> xn </bvar>
       P
    </bind>
  </apply>
</domainofapplication>
```

If the `apply` has a `domainofapplication` (perhaps originally expressed as `interval` or an `uplimit`/`lowlimit` pair) then that is used for R. Otherwise R is a set determined by the `type` attribute of the bound variable as specified in Section 4.2.2.2, if that is present. If the type is unspecified, the translation introduces an unspecified domain via content identifier `<ci>R</ci>`.

By applying the rules above, expression using the `interval`, `condition`, `uplimit` and `lowlimit` can be rewritten using only `domainofapplication`. Once a `domainofapplication` has been obtained, the final mapping to Strict markup is accomplished using the following rules:

Rewrite: restriction

An application of a function that is qualified by the `domainofapplication` qualifier (expressed by an `apply` element without bound variables) is converted to an application of a function term constructed with the restriction symbol.

```
<apply> F
  <domainofapplication>
     C
  </domainofapplication>
   a1
   an
</apply>
```

may be written as:

```
<apply>
  <apply><csymbol cd="fns1">restriction</csymbol>
     F
     C
  </apply>
   a1
   an
</apply>
```

In general, an application involving bound variables and (possibly) `domainofapplication` is rewritten using the following rule, which makes the domain the first positional argument of the application,

and uses the lambda symbol to encode the variable bindings. Certain classes of operator have alternative rules, as described below.

Rewrite: apply bvar domainofappliction

A content MathML expression with bound variables and `domainofapplication`

```
<apply> H
  <bvar> v1 </bvar>
  ...
  <bvar> vn </bvar>
  <domainofapplication> D </domainofapplication>
     A1
    ...
     Am
</apply>
```

is rewritten to

```
<apply> H
   D
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> v1 </bvar>
    ...
    <bvar> vn </bvar>
     A1
  </bind>
  ...
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> v1 </bvar>
    ...
    <bvar> vn </bvar>
     Am
  </bind>
</apply>
```

If there is no `domainofapplication` qualifier the *D* child is omitted.

4.3.3.2 Uses of <degree>

The `degree` element is a qualifier used to specify the 'degree' or 'order' of an operation. MathML uses the `degree` element in this way in three contexts: to specify the degree of a root, a moment, and in various derivatives. Rather than introduce special elements for each of these families, MathML provides a single general construct, the `degree` element in all three cases.

Note that the `degree` qualifier is not used to restrict a bound variable in the same sense of the qualifiers discussed above. Indeed, with roots and moments, no bound variable is involved at all, either explicitly or implicitly. In the case of differentiation, the `degree` element is used in conjunction with a `bvar`, but even in these cases, the variable may not be genuinely bound.

For the usage of `degree` with the `root` and `moment` operators, see the discussion of those operators below. The usage of `degree` in differentiation is more complex. In general, the `degree` element indicates the order of the derivative with respect to that variable. The degree element is allowed as the second

child of a bvar element identifying a variable with respect to which the derivative is being taken. Here is an example of a second derivative using the degree qualifier:

```
<apply><diff/>
  <bvar>
    <ci>x</ci>
    <degree><cn>2</cn></degree>
  </bvar>
  <apply><power/><ci>x</ci><cn>4</cn></apply>
</apply>
```

For details see Section 4.4.4.2 and Section 4.4.4.3.

4.3.3.3 *Uses of <momentabout> and <logbase>*

The qualifiers momentabout and logbase are specialized elements specifically for use with the moment and log operators respectively. See the descriptions of those operators below for their usage.

4.3.4 Operator Classes

The Content MathML elements described in detail in the next section may be broadly separated into *classes*. The class of each element is shown in the syntax table that introduces the element in Section 4.4. The class gives an indication of the general intended mathematical usage of the element, and also determines its usage as determined by the schema. The class also determines the applicable rewrite rules for mapping to Strict Content MathML. This section presents the rewrite rules for each of the operator classes.

The rules in this section cover the use cases applicable to specific operator classes. Special-case rewrite rules for individual elements are discussed in the sections below. However, the most common usage pattern is generic, and is used by operators from almost all operator classes. It consists of applying an operator to an explicit list of arguments using an apply element. In these cases, rewriting to Strict Content MathML is simply a matter of replacing the empty element with an appropriate csymbol, as listed in the syntax tables in Section 4.4. This is summarized in the following rule.

Rewrite: element

For example,

<plus/>

is equivalent to the Strict form

<csymbol cd="*arith1*">*plus*</csymbol>

The corresponding OpenMath symbols for elements in these classes also take an arbitrary number of arguments.

4.3.4.1 *N-ary Operators (classes nary-arith, nary-functional, nary-logical, nary-linag, nary-set, nary-constructor)*

Many MathML operators may be used with an arbitrary number of arguments. In all such cases, either the arguments my be given explictly as children of the apply or bind element, or the list may be specified implictly via the use of qualifier elements.

If the argument list is given explictly, the Rewrite: element rule applies.

Any use of qualifier elements is expressed in Strict Content MathML, via explictly applying the function to a list of arguments using the apply_to_list symbol as shown in the following rule. The rule only considers the `domainofapplication` qualifier as other qualifiers may be rewritten to `domainofapplication` as described earlier.

Rewrite: n-ary domainofapplication

An expression of the following form, where *<union/>* represents any element of the relevant class and *expression-in-x* is an arbitrary expression involving the bound variable(s)

```
<apply><union/>
  <bvar> x </bvar>
  <domainofapplication> D </domainofapplication>
   expression-in-x
</apply>
```

is rewritten to

```
<apply><csymbol cd="fns2">apply_to_list</csymbol>
  <csymbol cd="set1">union</csymbol>
  <apply><csymbol cd="list1">map</csymbol>
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar> x </bvar>
       expression-in-x
    </bind>
     D
  </apply>
</apply>
```

The above rule applies to all symbols in the listed classes. In the case of `nary-set` the choice of Content Dictionary to use depends on the `type` attribute on the symbol, defaulting to set1, but multiset1 should be used if `type="multiset"`.

Note: The above rules apply to n-ary constructors such as `vector` with the syntactic variation that the MathML element uses *constructor* syntax where the arguments and qualifiers are given as children of the element rather than as children of a containing `apply`.

4.3.4.2 N-ary Constructors for set and list (class nary-setlist-constructor)

The use of `set` and `list` follows the same format as other n-ary constructors, however when rewriting to Strict Content MathML a variant of the above rule is used. This is because the map symbol implicitly constructs the required set or list, and apply_to_list is not needed in this case.

Rewrite: n-ary setlist domainofapplication

An expression of the following form, where *<set/>* is either of the elements `set` or `list` and *expression-in-x* is an arbitrary expression involving the bound variable(s)

```
<set>
  <bvar> x </bvar>
  <domainofapplication> D </domainofapplication>
   expression-in-x
</set>
```

is rewritten to

```
<apply><csymbol cd="set1">map</csymbol>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> x </bvar>
     expression-in-x
  </bind>
   D
</apply>
```

4.3.4.3 N-ary Relations (classes nary-reln, nary-set-reln)

MathML allows allows transative relations to be used with multiple arguments, to give a natural expression to 'chains' of relations such as $a < b < c < d$. However unlike the case of the arithmetic operators, the underlying symbols used in the Strict Content MathML are classed as binary, so it is not possible to use apply_to_list as in the previous section, but instead a similar function predicate_on_list is used, the semantics of which is essentially to take the conjunction of applying the predicate to elements of the domain two at a time.

Rewrite: n-ary relations

An expression of the form

```
<apply><lt/>
   a   b   c   d
</apply>
```

rewrites to Strict Content MathML

```
<apply><csymbol cd="fns2">predicate_on_list</csymbol>
 <csymbol cd="reln1">lt</csymbol>
 <apply><csymbol cd="list1">list</csymbol>
   a   b   c   d
 </apply>
</apply>
```

Rewrite: n-ary relations bvar

An expression of the form

```
<apply><lt/>
 <bvar> x </bvar>
 <domainofapplication> R </domainofapplication>
  expression-in-x
</apply>
```

where *expression-in-x* is an arbitrary expression invloving the bound variable, rewrites to the Strict Content MathML

```
<apply><csymbol cd="fns2">predicate_on_list</csymbol>
 <csymbol cd="reln1">lt</csymbol>
 <apply><csymbol cd="list1">map</csymbol>
    R
   <bind><csymbol cd="fns1">lambda</csymbol>
     <bvar> x </bvar>
      expression-in-x
   </bind>
  </apply>
</apply>
```

The above rules apply to all symbols in classes `nary-reln` and `nary-set-reln`. In the latter case the choice of Content Dictionary to use depends on the `type` attribute on the symbol, defaulting to set1, but multiset1 should be used if `type="multiset"`.

4.3.4.4 N-ary/Unary Operators (classes nary-minmax, nary-stats)

The MathML elements, `max`, `min` and some satistical elements such as `mean` may be used as a n-ary function as in the above classes, however a special interpretation is given in the case that a single argument is supplied. If a single argument is supplied the function is applied to the elements represented by the argument.

The underlying symbol used in Strict Content MathML for these elements is *Unary* and so if the MathML is used with 0 or more than 1 arguments, the function is applied to the set constructed from the explictly supplied arguments acording to the following rule.

Rewrite: n-ary unary set

When an element, *<max/>*, of class nary-stats or nary-minmax is applied to an explicit list of 0 or 2 or more arguments, *a1 a2 an*

```
<apply><max/> a1  a2  an </apply>
```

It is is translated to the unary application of the symbol `<csymbol cd="minmax1" name="max"/>` as specified in the syntax table for the element to the set of arguments, constructed using the `<csymbol cd="set1" name="set"/>` symbol.

```
<apply><csymbol cd="minmax1">max</csymbol>
  <apply><csymbol cd="set1">set</csymbol>
     a1  a2  an
  </apply>
</apply>
```

Like all MathML n-ary operators, The list of arguments may be specified implictly using qualifier elements. This is expressed in Strict Content MathML using the following rule, which is similar to the rule Rewrite: n-ary domainofapplication but differs in that the symbol can be directly applied to the constructed set of arguments and it is not necessary to use apply_to_list.

Rewrite: n-ary unary domainofapplication

An expression of the following form, where *<max/>* represents any element of the relevant class and *expression-in-x* is an arbitrary expression involving the bound variable(s)

```
<apply><max/>
  <bvar> x </bvar>
  <domainofapplication> D </domainofapplication>
   expression-in-x
</apply>
```

is rewritten to

```
<apply><csymbol cd="minmax1">max</csymbol>
  <apply><csymbol cd="set1">map</csymbol>
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar> x </bvar>
       expression-in-x
    </bind>
     D
  </apply>
</apply>
```

If the element is applied to a single argument the set symbol is not used and the symbol is applied directly to the argument.

Rewrite: n-ary unary single

When an element, *<max/>*, of class nary-stats or nary-minmax is applied to a single argument,

```
<apply><max/> a </apply>
```

It is is translated to the unary application of the symbol in the syntax table for the element.

```
<apply><csymbol cd="minmax1">max</csymbol>  a  </apply>
```

Note: Earlier versions of MathML were not explict about the correct interpretation of elements in this class, and left it undefined as to whether an expression such as max(X) was a trivial application of max to a singleton, or whether it should be interpretted as meaning the maximum of values of the set X. Applications finding that the rule Rewrite: n-ary unary single can not be applied as the supplied argument is a scalar may wish to use the rule Rewrite: n-ary unary set as an error recovery. As a further complication, in the case of the statistical functions the Content Dictionary to use in this case depends on the desired interpretation of the argument as a set of explict data or a random variable representing a distribution.

4.3.4.5 Binary Operators (classes binary-arith, binary-logical, binary-reln, binary-linalg, binary-set, binary-constructor)

Binary operators take two arguments and simply map to OpenMath symbols without the need of any special rewrite rules. The binary constructor `interval` is similar but uses constructor syntax in which

the arguments are children of the element, and the symbol used depends on the type element as described in Section 4.4.1.1

4.3.4.6 Unary Operators (classes unary-arith, unary-functional, unary-set, unary-elementary, unary-veccalc)

Binary operators take a single arguments and map to OpenMath symbols without the need of any special rewrite rules.

4.3.4.7 Constants (classes constant-arith, constant-set)

Constant symbols relate to mathematical constants such as e and true and also to names of sets such as the Real Numbers, and Integers. In most cases they rewrite simply to a single symbol in Strict Content MathML.

4.3.4.8 Quantifiers (class quantifier)

The Quantifier class is used for the forall and exists quantifiers of predicate calculus. If used with `bind` and no qualifiers, then the interpretation in Strict Content MathML is simple. In general if used with `apply` or qualifiers, the interpretation in Strict Content MathML is via the following rule.

Rewrite: quantifier

An expression of following form where *<exists/>* denotes an element of class `quantifier` and *expression-in-x* is an arbitrary expression involving the bound variable(s)

```
<apply><exists/>
  <bvar> x </bvar>
  <domainofapplication> D </domainofapplication>
   expression-in-x
</apply>
```

is rewritten to an expression

```
<bind><csymbol cd="quant1">exists</csymbol>
  <bvar> x </bvar>
  <apply><csymbol cd="logic1">and</csymbol>
    <apply><csymbol cd="set1">in</csymbol> x  D </apply>
   expression-in-x
  </apply>
</bind>
```

where the symbols `<csymbol cd="quant1">exists</csymbol>` and `<csymbol cd="logic1">and</csymbol>` are as specified in the syntax table of the element. (The additional symbol being and in the case of `exists` and implies in the case of `forall`.)

4.3.4.9 Other Operators (classes lambda, interval, int, partialdiff, sum, product, limit)

Special purpose classes, described in the sections for the appropriate elements

4.4 Content MathML for Specific Operators and Constants

This section presents elements representing a core set of mathematical operators, functions and constants. Most are empty elements, covering the subject matter of standard mathematics curricula up to the level of calculus. The remaining elements are container elements for sets, intervals, vectors and so on. For brevity, all elements defined in this section are sometimes called *operator elements*.

Each subsection below discusses a specific operator element, beginning with a syntax table, giving the elements operator class. Special case rules for rewritting as Strict Markup are introduced as needed. However, in most cases, the generic rewrite rules for the appropriate operator class is sufficient. In particular, unless otherwise indicated, elements are to be rewritten using the default Rewrite: element rule. Note, however, that all elements in this section must be rewritten in some fashion, since they are not allowed in Strict Content markup.

In MathML 2, the `definitionURL` attribute could be used to redefine or modify the meaning of an operator element. This use of the `definitionURL` attribute is deprecated in MathML 3. Instead a `csymbol` element should be used. In general, the value of `cd` attribute on the `csymbol` will correspond to the `definitionURL` value.

Issue (): In MathML 2, the meaning of various operator elements could be specialized via various attributes, usually the `type` attribute. Strict Content MathML does not have this possibility

Resolution: We pass these attributes as extra arguments in the `apply` (or `bind` elements), or add new symbols for the non-default case to the respective content dictionaries.

4.4.1 Functions and Inverses

4.4.1.1 Interval `<interval>`

Class	interval
Attributes	CommonAtt,closure?
Content	ContExp,ContExp
OM Symbols	interval_cc, interval_oc, interval_co, interval_oo

The `interval` element is a container element used to represent simple mathematical intervals of the real number line. It takes an optional attribute `closure`, with a default value of `"closed"`.

Content MathML

```
<interval closure="open"><ci>x</ci><cn>1</cn></interval>
<interval closure="closed"><cn>0</cn><cn>1</cn></interval>
<interval closure="open-closed"><cn>0</cn><cn>1</cn></interval>
<interval closure="closed-open"><cn>0</cn><cn>1</cn></interval>
```

Sample Presentation

```
<mfenced><mi>x</mi><mn>1</mn></mfenced>
```

$(x, 1)$

```
<mfenced open="[" close="]"><mn>0</mn><mn>1</mn></mfenced>
```

$[0, 1]$

```
<mfenced open="(" close="]"><mn>0</mn><mn>1</mn></mfenced>
```

$(0, 1]$

```
<mfenced open="[" close=")"><mn>0</mn><mn>1</mn></mfenced>
```

$[0, 1)$

Mapping to Strict Content MathML

In Strict markup, the `interval` element corresponds to one of four symbols from the interval1 content dictionary. If `closure` has the value `"open"` then `interval` corresponds to the interval_oo. With the value `"closed"` `interval` corresponds to the symbol interval_cc, with value `"open-closed"` to interval_oc, and with `"closed-open"` to interval_co.

4.4.1.2 Inverse `<inverse>`

Class	unary-functional
Attributes	CommonAtt
Content	Empty
OM Symbols	inverse

The `inverse` element is applied to a function in order to construct a generic expression for the functional inverse of that function. The `inverse` element may either be applied to arguments, or it may appear alone, in which case it represents an abstract inversion operator acting on other functions.

Content MathML

```
<apply><inverse/>
  <ci> f </ci>
</apply>
```

Sample Presentation

```
<msup><mi>f</mi><mrow><mo>(</mo><mn>-1</mn><mo>)</mo></mrow></msup>
```

$f^{(-1)}$

Content MathML

```
<apply>
  <apply><inverse/><ci type="matrix">A</ci></apply>
  <ci>a</ci>
</apply>
```

Sample Presentation

```
<mrow>
 <msup><mi>A</mi><mrow><mo>(</mo><mn>-1</mn><mo>)</mo></mrow></msup>
 <mo>&#x2061;</mo>
 <mfenced><mi>a</mi></mfenced>
</mrow>
```

$$A^{(-1)}\,(a)$$

4.4.1.3 *Lambda* <lambda>

Class	lambda
Attributes	CommonAtt
Content	BvarQ, DomainQ, ContExp
Qualifiers	BvarQ,DomainQ
OM Symbols	lambda

The `lambda` element is used to construct a user-defined function from an expression, bound variables, and qualifiers. In a lambda construct with n (possibly 0) bound variables, the first n children are `bvar` elements that identify the variables that are used as placeholders in the last child for actual parameter values. The bound variables can be restricted by an optional `domainofapplication` qualifier or one of its shorthand notations. The meaning of the `lambda` construct is an n-ary function that returns the expression in the last child where the bound variables are replaced with the respective arguments.

The `domainofapplication` child restricts the possible values of the arguments of the constructed function. For instance, the following `lambda` construct represents a function on the integers.

```
<lambda>
  <bvar><ci> x </ci></bvar>
  <domainofapplication><integers/></domainofapplication>
  <apply><sin/><ci> x </ci></apply>
</lambda>
```

If a `lambda` construct does not contain bound variables, then the `lambda` construct is superfluous and may be removed, unless it also contains a `domainofapplication` construct. In that case, if the last child of the `lambda` construct is itself a function, then the `domainofapplication` restricts it's existing functional arguments, as in this example, which is a variant representation for the function above.

```
<lambda>
  <domainofapplication><integers/></domainofapplication>
  <sin/>
</lambda>
```

Otherwise, if the last child of the `lambda` construct is not a function, say a number, then the `lambda` construct will not be a function, but the same number, and any `domainofapplication` is ignored.

Content MathML

```
<lambda>
  <bvar><ci>x</ci></bvar>
  <apply><sin/>
    <apply><plus/><ci>x</ci><cn>1</cn></apply>
  </apply>
</lambda>
```

Sample Presentation

```
<mrow>
 <mi>&#x3bb;</mi>
 <mi>x</mi>
 <mo>.</mo>
 <mfenced>
  <mrow>
   <mi>sin</mi>
   <mo>&#x2061;</mo>
   <mrow><mo>(</mo><mi>x</mi><mo>+</mo><mn>1</mn><mo>)</mo></mrow>
  </mrow>
 </mfenced>
</mrow>
```

$$\lambda x.(\sin\,(x+1)\,)$$

```
<mrow>
 <mi>x</mi>
 <mo>&#x21a6;</mo>
  <mrow>
   <mi>sin</mi>
   <mo>&#x2061;</mo>
   <mrow><mo>(</mo><mi>x</mi><mo>+</mo><mn>1</mn><mo>)</mo></mrow>
  </mrow>
</mrow>
```

$$x \mapsto \sin\,(x+1)$$

Mapping to Strict Markup

Rewrite: lambda

If the `lambda` element does not contain qualifiers, the lambda expression is directly translated into a `bind` expression.

```
<lambda>
  <bvar> x1 </bvar><bvar> xn </bvar>
   expression-in-x1-xn
</lambda>
```

rewrites to the Strict Content MathML

```
<bind><csymbol cd="fns1">lambda</csymbol>
  <bvar> x1 </bvar><bvar> xn </bvar>
   expression-in-x1-xn
</bind>
```

Rewrite: lambda domainofappliction

If the `lambda` element does contain qualifiers, the qualifier may be rewritten to `domainofapplication` and then the lambda expression is translated to a function term constructed with lambda and restriction.

```
<lambda>
  <bvar> x1 </bvar><bvar> xn </bvar>
  <domainofapplication> D </domainofapplication>
   expression-in-x1-xn
</lambda>
```

rewrites to the Strict Content MathML

```
<apply><csymbol cd="fns1">restriction</csymbol>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> x1 </bvar><bvar> xn </bvar>
     expression-in-x1-xn
  </bind>
   D
</apply>
```

4.4.1.4 Function composition <compose/>

Class	nary-functional
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	left_compose

The `compose` element represents the function composition operator. Note that MathML makes no assumption about the domain and codomain of the constituent functions in a composition; the domain of the resulting composition may be empty.

The `compose` element is a commutative n-ary operator. Consequently, it may be lifted to the induced operator defined on a collection of arguments indexed by a (possibly infinite) set by using qualifier elements as described in Section 4.3.4.1.

Content MathML

```
<apply><compose/><ci>f</ci><ci>g</ci><ci>h</ci></apply>
```

Sample Presentation

```
<mrow><mi>f</mi><mo>&#x2218;</mo><mi>g</mi><mo>&#x2218;</mo><mi>h</mi></mrow>
```

$$f \circ g \circ h$$

Content MathML

```
<apply><eq/>
  <apply>
    <apply><compose/><ci>f</ci><ci>g</ci></apply>
    <ci>x</ci>
  </apply>
  <apply><ci>f</ci><apply><ci>g</ci><ci>x</ci></apply></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow>
  <mrow><mo>(</mo><mi>f</mi><mo>&#x2218;</mo><mi>g</mi><mo>)</mo></mrow>
  <mo>&#x2061;</mo>
  <mfenced><mi>x</mi></mfenced>
 </mrow>
 <mo>=</mo>
 <mrow>
  <mi>f</mi>
  <mo>&#x2061;</mo>
  <mfenced>
   <mrow>
    <mi>g</mi>
    <mo>&#x2061;</mo>
    <mfenced><mi>x</mi></mfenced>
  </mrow>
 </mfenced>
 </mrow>
</mrow>
```

$$(f \circ g)(x) = f(g(x))$$

4.4.1.5 *Identity function* `<ident/>`

Class	unary-functional
Attributes	CommonAtt
Content	Empty
OM Symbols	identity

The `ident` element represents the identity function. Note that MathML makes no assumption about the domain and codomain of the represented identity function, which depends on the context in which it is used.

Content MathML

```
<apply><eq/>
  <apply><compose/>
    <ci type="function">f</ci>
    <apply><inverse/>
      <ci type="function">f</ci>
    </apply>
  </apply>
  <ident/>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow>
  <mi>f</mi>
  <mo>&#x2218;</mo>
  <msup><mi>f</mi><mrow><mo>(</mo><mn>-1</mn><mo>)</mo></mrow></msup>
 </mrow>
 <mo>=</mo>
 <mi>id</mi>
</mrow>
```

$$f \circ f^{(-1)} = \mathrm{id}$$

4.4.1.6 Domain <domain/>

Class	unary-functional
Attributes	CommonAtt
Content	Empty
OM Symbols	domain

The domain element represents the domain of the function to which it is applied. The domain is the set of values over which the function is defined.

Content MathML

```
<apply><eq/>
  <apply><domain/><ci>f</ci></apply>
  <reals/>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mi>domain</mi><mo>&#x2061;</mo><mfenced><mi>f</mi></mfenced></mrow>
 <mo>=</mo>
 <mi mathvariant="double-struck">R</mi>
</mrow>
```

$$\mathrm{domain}\,(f) = \mathbb{R}$$

4.4.1.7 codomain <codomain/>

Class	unary-functional
Attributes	CommonAtt
Content	Empty
OM Symbols	range

The codomain represents the codomain, or range, of the function to which is is applied. Note that the codomain is not necessarily equal to the image of the function, it is merely required to contain the image.

Content MathML

```
<apply><eq/>
  <apply><codomain/><ci>f</ci></apply>
  <rationals/>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mi>codomain</mi><mo>&#x2061;</mo><mfenced><mi>f</mi></mfenced></mrow>
 <mo>=</mo>
 <mi mathvariant="double-struck">Q</mi>
</mrow>
```

$$\text{codomain}\,(f) = \mathbb{Q}$$

4.4.1.8 Image <image/>

Class	unary-functional
Attributes	CommonAtt
Content	Empty
OM Symbols	image

The image element represent the image of the function to which it is applied. The image of a function is the set of values taken by the function. Every point in the image is generated by the function applied to some point of the domain.

Content MathML

```
<apply><eq/>
  <apply><image/><sin/></apply>
  <interval><cn>-1</cn><cn> 1</cn></interval>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mi>image</mi><mo>&#x2061;</mo><mfenced><mi>sin</mi></mfenced></mrow>
 <mo>=</mo>
 <mfenced open="[" close="]"><mn>-1</mn><mn>1</mn></mfenced>
</mrow>
```

$$\text{image}\,(\sin) = [-1, 1]$$

4.4.1.9 Piecewise declaration (<piecewise>, <piece>, <otherwise>)

Class	Constructor
Attributes	CommonAtt
Content	piece* otherwise?
OM Symbols	piecewise

Syntax Table for Piecewise

Class	Constructor
Attributes	CommonAtt
Content	ContExp ContExp
OM Symbols	piece

Syntax Table for `piece`

Class	Constructor
Attributes	CommonAtt
Content	ContExp
OM Symbols	otherwise

Syntax Table for `otherwise`

The `piecewise`, `piece`, and `otherwise` elements are used to represent 'piecewise' function definitions of the form ' $H(x) = 0$ if x less than 0, $H(x) = 1$ otherwise'.

The declaration is constructed using the `piecewise` element. This contains zero or more `piece` elements, and optionally one `otherwise` element. Each `piece` element contains exactly two children. The first child defines the value taken by the `piecewise` expression when the condition specified in the associated second child of the `piece` is true. The degenerate case of no `piece` elements and no `otherwise` element is treated as undefined for all values of the domain.

The `otherwise` element allows the specification of a value to be taken by the `piecewise` function when none of the conditions (second child elements of the `piece` elements) is true, i.e. a default value.

It should be noted that no 'order of execution' is implied by the ordering of the `piece` child elements within `piecewise`. It is the responsibility of the author to ensure that the subsets of the function domain defined by the second children of the `piece` elements are disjoint, or that, where they overlap, the values of the corresponding first children of the `piece` elements coincide. If this is not the case, the meaning of the expression is undefined.

Mapping to Strict Markup

In Strict Content MathML, the container elements `piecewise`, `piece` and `otherwise` are mapped to applications of the constructor symbols of the same names in the piece1 CD. Apart from the fact that these three elements (respectively symbols) are used together, the mapping to Strict markup is straightforward:

Content MathML

```
<piecewise>
  <piece>
    <apply><cn>0</cn></apply>
    <apply><lt/><ci>x</ci><cn>0</cn></apply>
  </piece>
  <piece>
    <cn>1</cn>
    <apply><gt/><ci>x</ci><cn>1</cn></apply>
  </piece>
  <otherwise>
    <ci>x</ci>
  </otherwise>
</piecewise>
```

Strict Content MathML equivalent

```
<apply><csymbol cd="piece1">piecewise</csymbol>
  <apply><csymbol cd="piece1">piece</csymbol>
    <cn>0</cn>
    <apply><csymbol cd="relation1">lt</csymbol><ci>x</ci><cn>0</cn></apply>
  </apply>
  <apply><csymbol cd="piece1">piece</csymbol>
    <cn>1</cn>
    <apply><csymbol cd="relation1">gt</csymbol><ci>x</ci><cn>1</cn></apply>
  </apply>
  <apply><csymbol cd="piece1">otherwise</csymbol>
    <ci>x</ci>
  </apply>
</apply>
```

Here is an example that doesn't use the optional `otherwise` element:

Content MathML

```
<piecewise>
  <piece>
    <apply><minus/><ci>x</ci></apply>
    <apply><lt/><ci>x</ci><cn>0</cn></apply>
  </piece>
  <piece>
    <cn>0</cn>
    <apply><eq/><ci>x</ci><cn>0</cn></apply>
  </piece>
  <piece>
    <ci>x</ci>
    <apply><gt/><ci>x</ci><cn>0</cn></apply>
  </piece>
</piecewise>
```

Sample Presentation

```
<mrow>
 <mo>{</mo>
 <mtable>
  <mtr>
   <mtd><mrow><mo>&#x2212;</mo><mi>x</mi></mrow></mtd>
   <mtd columnalign="left"><mtext>  if  </mtext></mtd>
   <mtd><mrow><mi>x</mi><mo>&lt;</mo><mn>0</mn></mrow></mtd>
  </mtr>
  <mtr>
   <mtd><mn>0</mn></mtd>
   <mtd columnalign="left"><mtext>  if  </mtext></mtd>
   <mtd><mrow><mi>x</mi><mo>=</mo><mn>0</mn></mrow></mtd>
  </mtr>
  <mtr>
   <mtd><mi>x</mi></mtd>
   <mtd columnalign="left"><mtext>  if  </mtext></mtd>
   <mtd><mrow><mi>x</mi><mo>&gt;</mo><mn>0</mn></mrow></mtd>
  </mtr>
 </mtable>
</mrow>
```

$$\begin{cases} -x & \text{if} \quad x < 0 \\ 0 & \text{if} \quad x = 0 \\ x & \text{if} \quad x > 0 \end{cases}$$

4.4.2 Arithmetic, Algebra and Logic

4.4.2.1 Quotient `<quotient/>`

Class	binary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	quotient

The `quotient` element represents the integer division operator. When the operator is applied to integer

arguments a and b, the result is the 'quotient of a divided by b'. That is, the quotient of integers a and b, is the integer q such that $a = b * q + r$, with $|r|$ less than $|b|$ and $a * r$ positive. In common usage, q is called the quotient and r is the remainder.

Content MathML

```
<apply><quotient/><ci>a</ci><ci>b</ci></apply>
```

Sample Presentation

```
<mrow><mo>&#x230a;</mo><mi>a</mi><mo>/</mo><mi>b</mi><mo>&#x230b;</mo></mrow>
```

$\lfloor a/b \rfloor$

4.4.2.2 Factorial `<factorial/>`

Class	unary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	factorial

This element represents the unary factorial operator on non-negative integers.

The factorial of an integer n is given by $n! = n*(n\text{-}1)* \ldots * 1$

Content MathML

```
<apply><factorial/><ci>n</ci></apply>
```

Sample Presentation

```
<mrow><mi>n</mi><mo>!</mo></mrow>
```

$n!$

4.4.2.3 Division `<divide/>`

Class	binary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	divide

The `divide` element represents the division operator in a number field.

Content MathML

```
<apply><divide/>
  <ci>a</ci>
  <ci>b</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>/</mo><mi>b</mi></mrow>
```

a/b

4.4.2.4 Maximum <max/>

Class	nary-minmax
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ, DomainQ
OM Symbols	max

The max element denotes the maximum function, which returns the largest of the arguments to which it is applied. Its arguments may be explicitly specified in the enclosing apply element, or specified using qualfier elements as described in Section 4.3.4.4. Note that when applied to infinite sets of arguments, no maximal argument may exist.

Content MathML

```
<apply><max/><cn>2</cn><cn>3</cn><cn>5</cn></apply>
```

Sample Presentation

```
<mrow>
 <mi>max</mi>
 <mrow>
  <mo>{</mo><mn>2</mn><mo>,</mo><mn>3</mn><mo>,</mo><mn>5</mn><mo>}</mo>
 </mrow>
</mrow>
```

$$\max\{2,3,5\}$$

Content MathML

```
<apply><max/>
  <bvar><ci>y</ci></bvar>
  <condition>
    <apply><in/>
      <ci>y</ci>
      <interval><cn>0</cn><cn>1</cn></interval>
    </apply>
  </condition>
  <apply><power/><ci>y</ci><cn>3</cn></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>max</mi>
 <mrow>
  <mo>{</mo><mi>y</mi><mo>|</mo>
  <mrow>
   <mi>y</mi>
   <mo>&#x2208;</mo>
   <mfenced open="[" close="]"><mn>0</mn><mn>1</mn></mfenced>
  </mrow>
  <mo>}</mo>
 </mrow>
</mrow>
```

$$\max\{y|y \in [0,1]\}$$

4.4.2.5 Minimum <min/>

Class	nary-minmax
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	min

The `min` element denotes the minimum function, which returns the smallest of the arguments to which it is applied. Its arguments may be explicitly specified in the enclosing `apply` element, or specified using qualfier elements as described in Section 4.3.4.4. Note that when applied to infinite sets of arguments, no minimal argument may exist.

Content MathML

```
<apply><min/><ci>a</ci><ci>b</ci></apply>
```

Sample Presentation

```
<mrow>
 <mi>min</mi>
 <mrow><mo>{</mo><mi>a</mi><mo>,</mo><mi>b</mi><mo>}</mo></mrow>
</mrow>
```

$$\min\{a,b\}$$

Content MathML

```
<apply><min/>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><notin/><ci>x</ci><ci type="set">B</ci></apply>
  </condition>
  <apply><power/><ci>x</ci><cn>2</cn></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>min</mi>
 <mrow><mo>{</mo><mi>x</mi><mo>|</mo>
  <mrow><mi>x</mi><mo>&#x2209;</mo><mi>B</mi></mrow>
  <mo>}</mo>
</mrow>
</mrow>
```

$$\min\{x|x \notin B\}$$

4.4.2.6 Subtraction <minus/>

Class	unary-arith, binary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	unary_minus, minus

The `minus` element can be used as a *unary arithmetic operator* (e.g. to represent - x), or as a *binary arithmetic operator* (e.g. to represent x- y).

If it is used with one argument, `minus` corresponds to the unary_minus symbol.

Content MathML

```
<apply><minus/><cn>3</cn></apply>
```

Sample Presentation

```
<mrow><mo>&#x2212;</mo><mn>3</mn></mrow>
```

-3

If it is used with two arguments, `minus` corresponds to the minus symbol

Content MathML

```
<apply><minus/><ci>x</ci><ci>y</ci></apply>
```

Sample Presentation

```
<mrow><mi>x</mi><mo>&#x2212;</mo><mi>y</mi></mrow>
```

$x-y$

In both cases, the translation to Strict Content markup is direct, as described in Rewrite: element. It is merely a matter of choosing the symbol that reflects the actual usage.

4.4.2.7 *Addition* `<plus/>`

Class	nary-arith
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	plus

The `plus` element represents the addition operator. Its arguments are normally specified explicitly in the enclosing `apply` element. As an n-ary commutative operator, it can be used with qualifiers to specify arguments, however, this is discouraged, and the `sum` operator should be used to represent such expressions instead.

Content MathML

```
<apply><plus/><ci>x</ci><ci>y</ci><ci>z</ci></apply>
```

Sample Presentation

```
<mrow><mi>x</mi><mo>+</mo><mi>y</mi><mo>+</mo><mi>z</mi></mrow>
```

$x+y+z$

4.4.2.8 *Exponentiation* `<power/>`

Class	binary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	power

The `power` element represents the exponentiation operator. The first argument is raised to the power of the second argument.

Content MathML

```
<apply><power/><ci>x</ci><cn>3</cn></apply>
```

Sample Presentation

```
<msup><mi>x</mi><mn>3</mn></msup>
```

x^3

4.4.2.9 Remainder `<rem/>`

Class	binary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	remainder

The `rem` element represents the modulus operator, which returns the remainder that results from dividing the first argument by the second. That is, when applied to integer arguments a and b, it returns the unique integer r such that $a = b * q + r$, with $|r|$ less than $|b|$ and $a * r$ positive.

Content MathML

```
<apply><rem/><ci> a </ci><ci> b </ci></apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>mod</mo><mi>b</mi></mrow>
```

$a \bmod b$

4.4.2.10 Multiplication `<times/>`

Class	nary-arith
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	times

The `times` element represents the n-ary multiplication operator. Its arguments are normally specified explicitly in the enclosing `apply` element. As an n-ary commutative operator, it can be used with qualifiers to specify arguments by rule, however, this is discouraged, and the `product` operator should be used to represent such expressions instead.

Content MathML

```
<apply><times/><ci>a</ci><ci>b</ci></apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>&#x2062;</mo><mi>b</mi></mrow>
```

ab

4.4.2.11 Root `<root/>`

Class	unary-arith, binary-arith
Attributes	CommonAtt
Content	Empty
Qualifiers	degree
OM Symbols	root

The `root` element is used to extract roots. The kind of root to be taken is specified by a 'degree' element, which should be given as the second child of the `apply` element enclosing the `root` element. Thus, square roots correspond to the case where `degree` contains the value 2, cube roots correspond to 3, and so on. If no `degree` is present, a default value of 2 is used.

Content MathML

```
<apply><root/>
  <degree><ci type="integer">n</ci></degree>
  <ci>a</ci>
</apply>
```

Sample Presentation

```
<mroot><mi>a</mi><mi>n</mi></mroot>
```

$\sqrt[n]{a}$

Mapping to Strict Content Markup

In Strict Content markup, the root symbol is always used with two arguments, with the second indicating the degree of the root being extracted.

Content MathML

```
<apply><root/><ci>x</ci></apply>
```

Strict Content MathML equivalent

```
<apply><csymbol cd="arith1">root</csymbol>
  <ci>x</ci>
  <cn type="integer">2</cn>
</apply>
```

Content MathML

```
<apply><root/>
  <degree><ci type="integer">n</ci></degree>
  <ci>a</ci>
</apply>
```

Strict Content MathML equivalent

```
<apply><csymbol cd="arith1">root</csymbol>
  <ci>a</ci>
  <cn type="integer">n</cn>
</apply>
```

4.4.2.12 Greatest common divisor <gcd/>

Class	nary-arith
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	gcd

The gcd element represents the n-ary operator which returns the greatest common divisor of its arguments. Its arguments may be explicitly specified in the enclosing apply element, or specified by rule as described in Section 4.3.4.1.

Content MathML

```
<apply><gcd/><ci>a</ci><ci>b</ci><ci>c</ci></apply>
```

Sample Presentation

```
<mrow>
 <mi>gcd</mi>
 <mo>&#x2061;</mo>
 <mfenced><mi>a</mi><mi>b</mi><mi>c</mi></mfenced>
</mrow>
```

$$\gcd(a,b,c)$$

This default rendering is English-language locale specific: other locales may have different default renderings.

When the gcd element is applied to an explicit list of arguments, the translation to Strict Content markup is direct, using the gcd symbol, as described in Rewrite: element. However, when qualifiers are used, the equivalent Strict markup is computed via Rewrite: n-ary domainofapplication.

4.4.2.13 And <and/>

Class	nary-logical
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	and

The and element represents the logical ‘and’ function which is an n-ary function taking Boolean arguments and returning a Boolean value. It is true if all arguments are true, and false otherwise. Its arguments may be explicitly specified in the enclosing apply element, or specified by rule as described in Section 4.3.4.1.

Content MathML

```
<apply><and/><ci>a</ci><ci>b</ci></apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>&#x2227;</mo><mi>b</mi></mrow>
```

$$a \wedge b$$

Content MathML

```
<apply><and/>
  <bvar><ci>i</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <uplimit><ci>n</ci></uplimit>
  <apply><gt/>
    <apply><selector/><ci>a</ci><ci>i</ci></apply>
    <cn>0</cn>
  </apply>
</apply>
```

Strict Content MathML

```
<apply><csymbol cd="fns2">apply_to_list</csymbol>
  <csymbol cd="logic1">and</csymbol>
  <apply><csymbol cd="list1">map</csymbol>
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>i</ci></bvar>
      <apply><csymbol cd="relation1">gt</csymbol>
<apply><csymbol cd="linalg1">vector_selector</csymbol><ci>i</ci><ci>a</ci></apply>
<cn>0</cn>
      </apply>
    </bind>
    <apply><csymbol cd="interval1">integer_interval</csymbol>
      <cn type="integer">0</cn>
      <ci>n</ci>
    </apply>
  </apply>
</apply>
```

Sample Presentation

```
<mrow>
 <munderover>
  <mo>&#x22C0;</mo>
  <mrow><mi>i</mi><mo>=</mo><mn>0</mn></mrow>
  <mi>n</mi>
 </munderover>
 <mrow>
  <mo>(</mo>
  <msub><mi>a</mi><mi>i</mi></msub>
  <mo>&gt;</mo>
  <mn>0</mn>
  <mo>)</mo>
 </mrow>
</mrow>
```

$$\bigwedge_{i=0}^{n} (a_i > 0)$$

4.4.2.14 Or <or/>

Class	nary-logical
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	or

The or element represents the logical 'or' function. It is true if any of the arguments are true, and false otherwise.

Content MathML

```
<apply><or/><ci>a</ci><ci>b</ci></apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>&#x2228;</mo><mi>b</mi></mrow>
```

$a \vee b$

4.4.2.15 Exclusive Or <xor/>

Class	nary-logical
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	xor

The xor element represents the logical 'xor' function. It is true if there are an odd number of true arguments or false otherwise.

Content MathML

```
<apply><xor/><ci>a</ci><ci>b</ci></apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>xor</mo><mi>b</mi></mrow>
```

$a \operatorname{xor} b$

4.4.2.16 Not <not/>

Class	unary-logical
Attributes	CommonAtt
Content	Empty
OM Symbols	not

The note element represents the logical not function which takes one Boolean argument, and returns the opposite Boolean value.

Content MathML

```
<apply><not/><ci>a</ci></apply>
```

Sample Presentation

```
<mrow><mo>&#xac;</mo><mi>a</mi></mrow>
```

$\neg a$

4.4.2.17 Implies `<implies/>`

Class	binary-logical
Attributes	CommonAtt
Content	Empty
OM Symbols	implies

The `implies` element represents the logical implication function which takes two Boolean expressions as arguments. It evaluates to false if the first argument is true and the second argument is false, otherwise it evaluates to true.

Content MathML

```
<apply><implies/><ci>A</ci><ci>B</ci></apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x21d2;</mo><mi>B</mi></mrow>
```

$$A \Rightarrow B$$

4.4.2.18 Universal quantifier `<forall/>`

Class	quantifier
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	forall, implies

The `forall` element represents the universal ("for all") quantifier which takes one or more bound variables, and an argument which specifies the asserion being quantified. In addition, `condition` or other qualifiers may be used as described in Section 4.3.4.8 to limit the domain of the bound variables.

Content MathML

```
<bind><forall/>
  <bvar><ci>x</ci></bvar>
  <apply><eq/>
    <apply><minus/><ci>x</ci><ci>x</ci></apply>
    <cn>0</cn>
  </apply>
</bind>
```

Sample Presentation

```
<mrow>
 <mo>&#x2200;</mo>
 <mi>x</mi>
 <mo>.</mo>
 <mfenced>
  <mrow>
   <mrow><mi>x</mi><mo>&#x2212;</mo><mi>x</mi></mrow>
   <mo>=</mo>
   <mn>0</mn>
  </mrow>
 </mfenced>
</mrow>
```

$$\forall x.(x - x = 0)$$

When the `forall` element is used with a `condition` qualifier the strict equivalent is constructed with the help of logical implication. Thus by the rules above:

```
<bind><forall/>
  <bvar><ci>p</ci></bvar>
  <bvar><ci>q</ci></bvar>
  <condition>
    <apply><and/>
      <apply><in/><ci>p</ci><rationals/></apply>
      <apply><in/><ci>q</ci><rationals/></apply>
      <apply><lt/><ci>p</ci><ci>q</ci></apply>
    </apply>
  </condition>
  <apply><lt/>
    <ci>p</ci>
    <apply><power/><ci>q</ci><cn>2</cn></apply>
  </apply>
</bind>
```

translates to

```
<bind><csymbol cd="quant1">forall</csymbol>
  <bvar><ci>p</ci></bvar>
  <bvar><ci>q</ci></bvar>
  <apply><csymbol cd="logic1">implies</csymbol>
    <apply><csymbol cd="logic1">and</csymbol>
      <apply><csymbol cd="set1">in</csymbol>
        <ci>p</ci>
        <csymbol cd="setname1">Q</csymbol>
        </apply>
      <apply><csymbol cd="set1">in</csymbol>
        <ci>q</ci>
        <csymbol cd="setname1">Q</csymbol>
      </apply>
      <apply><csymbol cd="relation1">lt</csymbol><ci>p</ci><ci>q</ci></apply>
    </apply>
    <apply><csymbol cd="relation1">lt</csymbol>
      <ci>p</ci>
      <apply><csymbol cd="arith1">power</csymbol>
        <ci>q</ci>
        <cn>2</cn>
      </apply>
    </apply>
  </apply>
</bind>
```

Sample Presentation

```
<mrow>
 <mo>&#x2200;</mo>
 <mrow>
  <mrow><mi>p</mi><mo>&#x2208;</mo><mi mathvariant="double-struck">Q</mi></mrow>
  <mo>&#x2227;</mo>
  <mrow><mi>q</mi><mo>&#x2208;</mo><mi mathvariant="double-struck">Q</mi></mrow>
  <mo>&#x2227;</mo>
  <mrow><mo>(</mo><mi>p</mi><mo>&lt;</mo><mi>q</mi><mo>)</mo></mrow>
 </mrow>
 <mo>.</mo>
 <mfenced>
```

4.4.2.19 Existential quantifier `<exists/>`

Class	quantifier
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	exists, and

The `exists` element represents the existential ("there exists") quantifier which takes one or more bound variables, and an argument which specifies the assertion being quantified. In addition, `condition` or other qualifiers may be used as described in Section 4.3.4.8 to limit the domain of the bound variables.

Content MathML

```
<bind><exists/>
  <bvar><ci>x</ci></bvar>
  <apply><eq/>
    <apply><ci>f</ci><ci>x</ci></apply>
    <cn>0</cn>
  </apply>
</bind>
```

Sample Presentation

```
<mrow>
 <mo>&#x2203;</mo>
 <mi>x</mi>
 <mo>.</mo>
 <mfenced>
  <mrow>
   <mrow><mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi></mfenced></mrow>
   <mo>=</mo>
   <mn>0</mn>
  </mrow>
 </mfenced>
</mrow>
```

$$\exists x.(f(x) = 0)$$

Content MathML

```
<apply><exists/>
  <bvar><ci>x</ci></bvar>
  <domainofapplication>
    <integers/>
  </domainofapplication>
   <apply><eq/>
    <apply><ci>f</ci><ci>x</ci></apply>
    <cn>0</cn>
  </apply>
</apply>
```

Strict MathML equivalent:

```
<bind><csymbol cd="quant1">exists</csymbol>
  <bvar><ci>x</ci></bvar>
  <apply><csymbol cd="logic1">and</csymbol>
    <apply><csymbol cd="set1">in</csymbol>
      <ci>x</ci>
      <csymbol cd="setname1">Z</csymbol>
    </apply>
    <apply><csymbol cd="relation1">eq</csymbol>
      <apply><ci>f</ci><ci>x</ci></apply>
      <cn>0</cn>
    </apply>
  </apply>
</bind>
```

Sample Presentation

```
<mrow>
 <mo>&#x2203;</mo>
 <mi>x</mi>
 <mo>.</mo>
 <mfenced separators="">
  <mrow><mi>x</mi><mo>&#x2208;</mo><mi mathvariant="double-struck">Z</mi></mrow>
  <mo>&#x2227;</mo>
  <mrow>
   <mrow><mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi></mfenced></mrow>
   <mo>=</mo>
   <mn>0</mn>
  </mrow>
 </mfenced>
</mrow>
```

$$\exists x.(x \in \mathbb{Z} \wedge f(x) = 0)$$

4.4.2.20 Absolute Value <abs/>

Class	unary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	abs

The abs element represents the absolute value function. The argument should be numerically valued. When the argument is a complex number, the absolute value is often referred to as the modulus.

Content MathML

```
<apply><abs/><ci>x</ci></apply>
```

Sample Presentation

```
<mrow><mo>|</mo><mi>x</mi><mo>|</mo></mrow>
```

$$|x|$$

4.4.2.21 Complex conjugate <conjugate/>

Class	unary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	conjugate

The conjugate element represents the function defined over the complex numbers with returns the complex conjugate of its argument.

Content MathML

```
<apply><conjugate/>
  <apply><plus/>
    <ci>x</ci>
    <apply><times/><cn>&#x2148;</cn><ci>y</ci></apply>
  </apply>
</apply>
```

Sample Presentation

```
<mover>
 <mrow>
  <mi>x</mi>
  <mo>+</mo>
  <mrow><mn>&#x2148;</mn><mo>&#x2062;</mo><mi>y</mi></mrow>
 </mrow>
 <mo>&#xaf;</mo>
</mover>
```

$$\overline{x+\mathrm{i}y}$$

4.4.2.22 Argument <arg/>

Class	unary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	argument

The arg element represents the unary function which returns the angular argument of a complex number, namely the angle which a straight line drawn from the number to zero makes with the real line (measured anti-clockwise).

Content MathML

```
<apply><arg/>
  <apply><plus/>
    <ci> x </ci>
    <apply><times/><imaginaryi/><ci>y</ci></apply>
  </apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>arg</mi>
 <mo>&#x2061;</mo>
 <mfenced>
  <mrow>
   <mi>x</mi>
   <mo>+</mo>
   <mrow><mi>i</mi><mo>&#x2062;</mo><mi>y</mi></mrow>
  </mrow>
 </mfenced>
</mrow>
```

$$\arg(x+iy)$$

4.4.2.23 Real part `<real/>`

Class	unary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	real

The `real` element represents the unary operator used to construct an expression representing the "real" part of a complex number, that is, the x component in $x + iy$.

Content MathML

```
<apply><real/>
  <apply><plus/>
    <ci>x</ci>
    <apply><times/><imaginaryi/><ci>y</ci></apply>
  </apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mo>&#x211b;</mo>
 <mo>&#x2061;</mo>
 <mfenced>
  <mrow>
   <mi>x</mi>
   <mo>+</mo>
   <mrow><mi>i</mi><mo>&#x2062;</mo><mi>y</mi></mrow>
  </mrow>
 </mfenced>
</mrow>
```

$$\mathcal{R}(x+iy)$$

4.4.2.24 *Imaginary part* `<imaginary/>`

Class	unary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	imaginary

The `imaginary` element represents the unary operator used to construct an expression representing the "imaginary" part of a complex number, that is, the y component in $x + iy$.

Content MathML

```
<apply><imaginary/>
  <apply><plus/>
    <ci>x</ci>
    <apply><times/><imaginaryi/><ci>y</ci></apply>
  </apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mo>&#x2111;</mo>
 <mo>&#x2061;</mo>
 <mfenced>
  <mrow>
   <mi>x</mi>
   <mo>+</mo>
   <mrow><mi>i</mi><mo>&#x2062;</mo><mi>y</mi></mrow>
  </mrow>
 </mfenced>
</mrow>
```

$$\Im(x+iy)$$

4.4.2.25 Lowest common multiple <lcm/>

Class	nary-arith
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	lcm

The lcm element represents the n-ary operator used to construct an expression which represents the least common multiple of its arguments. If no argument is provided, the lcm is 1. If one argument is provided, the lcm is that argument. The least common multiple of x and 1 is x.

Content MathML

```
<apply><lcm/><ci>a</ci><ci>b</ci><ci>c</ci></apply>
```

Sample Presentation

```
<mrow>
 <mi>lcm</mi>
 <mo>&#x2061;</mo>
 <mfenced><mi>a</mi><mi>b</mi><mi>c</mi></mfenced>
</mrow>
```

$$\mathrm{lcm}\,(a,b,c)$$

This default rendering is English-language locale specific: other locales may have different default renderings.

4.4.2.26 Floor <floor/>

Class	unary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	floor

The floor element represents the operation that rounds down (towards negative infinity) to the nearest integer. This function takes one real number as an argument and returns an integer.

Content MathML

```
<apply><floor/><ci>a</ci></apply>
```

Sample Presentation

```
<mrow><mo>&#x230a;</mo><mi>a</mi><mo>&#x230b;</mo></mrow>
```

$\lfloor a \rfloor$

4.4.2.27 Ceiling <ceiling/>

Class	unary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	ceiling

The ceiling element represents the operation that rounds up (towards positive infinity) to the nearest integer. This function takes one real number as an argument and returns an integer.

Content MathML

```
<apply><ceiling/><ci>a</ci></apply>
```

Sample Presentation

```
<mrow><mo>&#x2308;</mo><mi>a</mi><mo>&#x2309;</mo></mrow>
```

$\lceil a \rceil$

4.4.3 Relations

4.4.3.1 Equals <eq/>

Class	nary-reln
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	eq

The eq elements represents the equality relation.

Content MathML

```
<apply><eq/>
  <cn type="rational">2<sep/>4</cn>
  <cn type="rational">1<sep/>2</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mn>2</mn><mo>/</mo><mn>4</mn></mrow>
 <mo>=</mo>
 <mrow><mn>1</mn><mo>/</mo><mn>2</mn></mrow>
</mrow>
```

$$2/4 = 1/2$$

4.4.3.2 *Not Equals <neq/>*

Class	binary-reln
Attributes	CommonAtt
Content	Empty
OM Symbols	neq

The neq element represents the binary inequality relation, i.e. the relation "not equal to" which returns true unless the two arguments are equal.

Content MathML

```
<apply><neq/><cn>3</cn><cn>4</cn></apply>
```

Sample Presentation

```
<mrow><mn>3</mn><mo>&#x2260;</mo><mn>4</mn></mrow>
```

$$3 \neq 4$$

4.4.3.3 *Greater than <gt/>*

Class	nary-reln
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	gt

The gt element represents the "greater than" function which returns true if the first argument is greater than the second, and returns false otherwise. While this is a binary relation, gt may be used with more than two arguments, denoting a chain of inequalities, as described in Section 4.3.4.3.

Content MathML

```
<apply><gt/><cn>3</cn><cn>2</cn></apply>
```

Sample Presentation

```
<mrow><mn>3</mn><mo>&gt;</mo><mn>2</mn></mrow>
```

$$3 > 2$$

4.4.3.4 Less Than <lt/>

Class	nary-reln
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	lt

The lt element represents the "less than" function which returns true if the first argument is less than the second, and returns false otherwise. While this is a binary relation, lt may be used with more than two arguments, denoting a chain of inequalities, as described in Section 4.3.4.3.

Content MathML

```
<apply><lt/><cn>2</cn><cn>3</cn><cn>4</cn></apply>
```

Sample Presentation

```
<mrow><mn>2</mn><mo>&lt;</mo><mn>3</mn><mo>&lt;</mo><mn>4</mn></mrow>
```

$$2 < 3 < 4$$

4.4.3.5 Greater Than or Equal <geq/>

Class	nary-reln
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	geq

The geq element represents the "greater than or equal to" function which returns true if the first argument is greater than or equal to the second, and returns false otherwise. While this is a binary relation, geq may be used with more than two arguments, denoting a chain of inequalities, as described in Section 4.3.4.3.

Content MathML

```
<apply><geq/><cn>4</cn><cn>3</cn><cn>3</cn></apply>
```

Strict Content MathML

```
<apply><csymbol cd="fns2">predicate_on_list</csymbol>
 <csymbol cd="reln1">geq</csymbol>
 <apply><csymbol cd="list1">list</csymbol>
  <cn>4</cn><cn>3</cn><cn>3</cn>
 </apply>
</apply>
```

Sample Presentation

```
<mrow><mn>4</mn><mo>&#x2265;</mo><mn>3</mn><mo>&#x2265;</mo><mn>3</mn></mrow>
```

$$4 \geq 3 \geq 3$$

4.4.3.6 Less Than or Equal <leq/>

Class	nary-reln
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	leq

The leq element represents the "less than or equal to" function which returns true if the first argument is less than or equal to the second, and returns false otherwise. While this is a binary relation, leq may be used with more than two arguments, denoting a chain of inequalities, as described in Section 4.3.4.3.

Content MathML

```
<apply><leq/><cn>3</cn><cn>3</cn><cn>4</cn></apply>
```

Sample Presentation

```
<mrow><mn>3</mn><mo>&#x2264;</mo><mn>3</mn><mo>&#x2264;</mo><mn>4</mn></mrow>
```

$$3 \leq 3 \leq 4$$

4.4.3.7 Equivalent <equivalent/>

Class	binary-logical
Attributes	CommonAtt
Content	Empty
OM Symbols	equivalent

The equivalent element represents the relation that asserts two Boolean expressions are logically equivalent, that is have the same Boolean value for any inputs.

Content MathML

```
<apply><equivalent/>
  <ci>a</ci>
  <apply><not/><apply><not/><ci>a</ci></apply></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>a</mi>
 <mo>&#x2261;</mo>
 <mrow><mo>&#xac;</mo><mrow><mo>&#xac;</mo><mi>a</mi></mrow></mrow>
</mrow>
```

$$a \equiv \neg\neg a$$

4.4.3.8 Approximately <approx/>

Class	binary-reln
Attributes	CommonAtt
Content	Empty
OM Symbols	approx

The approx element represent the relation that asserts the approximate equality of its arguments.

Content MathML

```
<apply><approx/>
  <pi/>
  <cn type="rational">22<sep/>7</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>&#x3c0;</mi>
 <mo>&#x2243;</mo>
 <mrow><mn>22</mn><mo>/</mo><mn>7</mn></mrow>
</mrow>
```

$$\pi \simeq 22/7$$

4.4.3.9 Factor Of `<factorof/>`

Class	binary-reln
Attributes	CommonAtt
Content	Empty
OM Symbols	factorof

The `factorof` element is used to indicate the mathematical relationship that the first argument "is a factor of" the second. This relationship is true if and only if $b \bmod a = 0$.

Content MathML

```
<apply><factorof/><ci>a</ci><ci>b</ci></apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>|</mo><mi>b</mi></mrow>
```

$$a|b$$

4.4.4 Calculus and Vector Calculus

4.4.4.1 Integral `<int/>`

Class	int
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	int defint

The `int` element is the operator element for a definite or indefinite integral over a function or a definite over an expression with a bound variable.

Content MathML

```
<apply><eq/>
  <apply><int/><sin/></apply>
  <cos/>
</apply>
```

Sample Presentation

```
<mrow><mrow><mi>&#x222b;</mi><mi>sin</mi></mrow><mo>=</mo><mi>cos</mi></mrow>
```

$$\int \sin = \cos$$

Content MathML

```
<apply><int/>
  <interval><ci>a</ci><ci>b</ci></interval>
  <cos/>
</apply>
```

Sample Presentation

```
<mrow><msubsup><mi>&#x222b;</mi><mi>a</mi><mi>b</mi></msubsup><mi>cos</mi></mrow>
```

$$\int_a^b \cos$$

The `int` element can also be used with bound variables serving as the integration variables.

Content MathML

Here, definite integrals are indicated by providing qualifier elements specifying a domain of integration (here a `lowlimit`/`uplimit` pair). This is perhaps the most "standard" representation of this integral:

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <uplimit><cn>1</cn></uplimit>
  <apply><power/><ci>x</ci><cn>2</cn></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <msubsup><mi>&#x222b;</mi><mn>0</mn><mn>1</mn></msubsup>
 <msup><mi>x</mi><mn>2</mn></msup>
 <mi>d</mi>
 <mi>x</mi>
</mrow>
```

$$\int_0^1 x^2 dx$$

Mapping to Strict Markup

As an indefinite integral applied to a function, the `int` element corresponds to the int symbol from the calculus1 content dictionary. As a definite integral applied to a function, the `int` element corresponds to the defint symbol from the calculus1 content dictionary. For the case of bound variables the situation is more complicated in general, and the following rule is used.

Rewrite: int

Translate a definite integral, where *expression-in-x* is an arbitrary expression involving the bound variable(s) *x*

```
<apply><int/>
  <bvar> x </bvar>
  <domainofapplication> D </domainofapplication>
   expression-in-x
</apply>
```

to the expression

```
<apply>
  <apply><csymbol cd="calculus1">defint</csymbol>
     D
    <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> x </bvar>
     expression-in-x
    </bind>
  </apply>
   x
</apply>
```

For the indefinite integral, where the `domainofapplication` element is missing, the defint is used instead and the *D* is dropped. Note that as x is not bound in the original indefinite integral, the integrated function is applied to the variable x making it an explicit free variable in Strict Content Markup expression, even though it is bound in the subterm used as an argument to defint.

For instance, the expression

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <apply><cos/><ci>x</ci></apply>
</apply>
```

has the Strict Content MathML equivalent

```
<apply>
  <apply><csymbol cd="calculus1">int</csymbol>
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><cos/><ci>x</ci></apply>
    </bind>
  </apply>
  <ci>x</ci>
</apply>
```

But the definite integral with an `lowlimit`/`uplimit` pair carries the strong intuition that the range of integration is oriented, and thus swapping lower and upper limits will change the sign of the result. To accomodate this, use the following special translation rule:

Rewrite: int limits

```
<apply><int/>
  <bvar> x </bvar>
  <lowlimit> a </lowlimit>
  <uplimit> b </uplimit>
   expression-in-x
</apply>
```

where *expression-in-x* is an expression in the variable x is translated to to the expression:

```
<apply>
  <apply><csymbol cd="calculus1">defint</csymbol>
    <apply><csymbol cd="interval1">ordered_interval</csymbol>
       a   b
    </apply>
    <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> x </bvar>
     E
    </bind>
  </apply>
   x
</apply>
```

The case for multiple integrands is treated analogously.

Note that use of the `condition` element extends to multivariate domains by using extra bound variables and a domain corresponding to a cartesian product as in:

```
<bind><int/>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <condition>
    <apply><and/>
      <apply><leq/><cn>0</cn><ci>x</ci></apply>
      <apply><leq/><ci>x</ci><cn>1</cn></apply>
      <apply><leq/><cn>0</cn><ci>y</ci></apply>
      <apply><leq/><ci>y</ci><cn>1</cn></apply>
    </apply>
  </condition>
  <apply><times/>
    <apply><power/><ci>x</ci><cn>2</cn></apply>
    <apply><power/><ci>y</ci><cn>3</cn></apply>
  </apply>
</bind>
```

Strict Content MathML equivalent

```
<bind><csymbol cd="calculus1">defint</csymbol>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <apply><csymbol cd="set1">suchthat</csymbol>
    <apply><csymbol cd="set1">cartesianproduct</csymbol>
      <csymbol cd="setname1">R</csymbol>
      <csymbol cd="setname1">R</csymbol>
    </apply>
    <apply><csymbol cd="logic1">and</csymbol>
      <apply><csymbol cd="arith1">leq</csymbol><cn>0</cn><ci>x</ci></apply>
      <apply><csymbol cd="arith1">leq</csymbol><ci>x</ci><cn>1</cn></apply>
      <apply><csymbol cd="arith1">leq</csymbol><cn>0</cn><ci>y</ci></apply>
      <apply><csymbol cd="arith1">leq</csymbol><ci>y</ci><cn>1</cn></apply>
    </apply>
    <apply><csymbol cd="arith1">times</csymbol>
      <apply><csymbol cd="arith1">power</csymbol><ci>x</ci><cn>2</cn></apply>
      <apply><csymbol cd="arith1">power</csymbol><ci>y</ci><cn>3</cn></apply>
    </apply>
  </apply>
</bind>
```

4.4.4.2 *Differentiation* `<diff/>`

Class	Differential Operator
Attributes	CommonAtt
Content	Empty
OM Symbols	diff

The `diff` element is the differentiation operator element for functions or expressions of a single variable. It may be applied directly to an actual function thereby denoting a function which is the derivative of the original function, or it can be applied to an expression involving a single variable.

Content MathML

```
<apply><diff/><ci>f</ci></apply>
```

Sample Presentation

```
<msup><mi>f</mi><mo>&#x2032;</mo></msup>
```

$$f'$$

Content MathML

```
<apply><eq/>
  <apply><diff/>
    <bvar><ci>x</ci></bvar>
    <apply><sin/><ci>x</ci></apply>
  </apply>
  <apply><cos/><ci>x</ci></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mfrac>
  <mrow><mi>d</mi><mrow><mi>sin</mi><mo>&#x2061;</mo><mi>x</mi></mrow></mrow>
  <mrow><mi>d</mi><mi>x</mi></mrow>
 </mfrac>
 <mo>=</mo>
 <mrow><mi>cos</mi><mo>&#x2061;</mo><mi>x</mi></mrow>
</mrow>
```

$$\frac{d\sin x}{dx} = \cos x$$

The `bvar` element may also contain a `degree` element, which specifies the order of the derivative to be taken.

Content MathML

```
<apply><diff/>
  <bvar><ci>x</ci><degree><cn>2</cn></degree></bvar>
  <apply><power/><ci>x</ci><cn>4</cn></apply>
</apply>
```

Sample Presentation

```
<mfrac>
 <mrow>
  <msup><mi>d</mi><mn>2</mn></msup>
  <msup><mi>x</mi><mn>4</mn></msup>
 </mrow>
 <mrow><mi>d</mi><msup><mi>x</mi><mn>2</mn></msup></mrow>
</mfrac>
```

Mapping to Strict Markup

For the translation to strict Markup it is crucial to realize that in the expression case, the variable is actually not bound by the differentiation operator.

Rewrite: diff

Translate an expression

```
<apply><diff/>
  <bvar> x </bvar>
   expression-in-x
</apply>
```

where *expression-in-x* is an expression in the variable x to the expression

```
<apply>
  <apply><csymbol cd="calculus1">diff</csymbol>
    <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> x </bvar>
     E
    </bind>
  </apply>
   x
</apply>
```

Note that the the differentiated function is applied to the variable x making its status as a free variable explicit in strict markup. Thus the strict equivalent of

```
<apply><diff/>
  <bvar><ci>x</ci></bvar>
  <apply><sin/><ci>x</ci></apply>
</apply>
```

is

```
<apply>
  <apply><csymbol cd="calculus1">diff</csymbol>
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="transc1">sin</csymbol><ci>x</ci></apply>
    </bind>
  </apply>
  <ci>x</ci>
</apply>
```

If the `bvar` element contains a `degree` element, use the `nthdiff` symbol.

Rewrite: nthdiff

```
<apply><diff/>
  <bvar> x <degree> n </degree></bvar>
   expression-in-x
</apply>
```

where *expression-in-x* is an is an expression in the variable x is translated to to the expression:

```
<apply>
  <apply><csymbol cd="calculus1">nthdiff</csymbol>
     n
    <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> x </bvar>
     expression-in-x
    </bind>
  </apply>
   x
</apply>
```

For example

```
<apply><diff/>
  <bvar><degree><cn>2</cn></degree><ci>x</ci></bvar>
  <apply><sin/><ci>x</ci></apply>
</apply>
```

Strict Content MathML equivalent

```
<apply>
  <apply><csymbol cd="calculus1">nthdiff</csymbol>
    <cn>2</cn>
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="transc1">sin</csymbol><ci>x</ci></apply>
    </bind>
  </apply>
  <ci>x</ci>
</apply>
```

4.4.4.3 Partial Differentiation <partialdiff/>

Class	partialdiff
Attributes	CommonAtt
Content	Empty
OM Symbols	partialdiff partialdiffdegree

The `partialdiff` element is the partial differentiation operator element for functions or expressions in several variables.

For the case of partial differentiation of a function, the containing `partialdiff` takes two arguments: firstly a list of indices indicating by position which function arguments are involved in constructing the partial derivatives, and secondly the actual function to be partially differentiated. The indices may be repeated.

Content MathML

```
<apply><partialdiff/>
  <list><cn>1</cn><cn>1</cn><cn>3</cn></list>
  <ci type="function">f</ci>
</apply>
```

Sample Presentation

```
<mrow>
 <msub>
  <mi>D</mi>
  <mrow><mn>1</mn><mo>,</mo><mn>1</mn><mo>,</mo><mn>3</mn></mrow>
 </msub>
 <mi>f</mi>
</mrow>
```

$$D_{1,1,3}\, f$$

Content MathML

```
<apply><partialdiff/>
  <list><cn>1</cn><cn>1</cn><cn>3</cn></list>
  <lambda>
   <bvar><ci>x</ci></bvar>
   <bvar><ci>y</ci></bvar>
   <bvar><ci>f</ci></bvar>
   <apply><ci>f</ci><ci>x</ci><ci>y</ci><ci>f</ci></apply>
  </lambda>
</apply>
```

Sample Presentation

```
<mfrac>
 <mrow>
  <msup><mo>&#x2202;</mo><mn>3</mn></msup>
  <mrow><mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi><mi>y</mi><mi>z</mi></mfenced></mrow
 </mrow>
 <mrow>
  <mrow><mo>&#x2202;</mo><msup><mi>x</mi><mn>2</mn></msup></mrow>
  <mrow><mo>&#x2202;</mo><mi>z</mi></mrow>
 </mrow>
</mfrac>
```

$$\frac{\partial^3 f(x,y,z)}{\partial x^2 \partial z}$$

In the case of algebraic expressions, the bound variables are given by `bvar` elements, which are children of the containing `apply` element. The `bvar` elements may also contain `degree` element, which specify the order of the partial derivative to be taken in that variable.

Content MathML

```
<apply><partialdiff/>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <apply><ci type="function">f</ci><ci>x</ci><ci>y</ci></apply>
</apply>
```

Sample Presentation

```
<mfrac>
 <mrow>
  <msup><mo>&#x2202;</mo><mn>2</mn></msup>
  <mrow>
   <mi>f</mi>
   <mo>&#x2061;</mo>
   <mfenced><mi>x</mi><mi>y</mi></mfenced>
  </mrow>
 </mrow>
 <mrow>
  <mrow><mo>&#x2202;</mo><mi>x</mi></mrow>
  <mrow><mo>&#x2202;</mo><mi>y</mi></mrow>
 </mrow>
</mfrac>
```

$$\frac{\partial^2 f(x,y)}{\partial x \partial y}$$

Where a total degree of differentiation must be specified, this is indicated by use of a `degree` element at the top level, i.e. without any associated `bvar`, as a child of the containing `apply` element.

Content MathML

```
<apply><partialdiff/>
  <bvar><ci>x</ci><degree><ci>m</ci></degree></bvar>
  <bvar><ci>y</ci><degree><ci>n</ci></degree></bvar>
  <degree><ci>k</ci></degree>
  <apply><ci type="function">f</ci>
    <ci>x</ci>
    <ci>y</ci>
  </apply>
</apply>
```

Sample Presentation

```
<mfrac>
 <mrow>
  <msup><mo>&#x2202;</mo><mi>k</mi></msup>
  <mrow><mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi><mi>y</mi></mfenced></mrow>
 </mrow>
 <mrow>
  <mrow><mo>&#x2202;</mo><msup><mi>x</mi><mi>m</mi></msup></mrow>
  <mrow><mo>&#x2202;</mo><msup><mi>y</mi><mi>n</mi></msup></mrow>
 </mrow>
</mfrac>
```

$$\frac{\partial^k f(x,y)}{\partial x^m \partial y^n}$$

Mapping to Strict Markup

When applied to a function, the `partialdiff` element corresponds to the partialdiff symbol from the calculus1 content dictionary. No special rules are necessary as the two arguments of `partialdiff` translate directly to the two arguments of partialdiff.

Rewrite: partialdiffdegree

If `partialdiff` is used with an expression and `bvar` qualifiers it is rewritten to Strict Content MathML using the partialdiffdegree symbol.

```
<apply><partialdiff/>
  <bvar> x1 <degree> n1 </degree></bvar>
  <bvar> xk <degree> nk </degree></bvar>
  <degree> total-n1-nk </degree>
   expression-in-x1-xk
</apply>
```

expression-in-x1-xk is an arbitrary expression involving the bound variables.

```
<apply>
  <apply><csymbol cd="calculus1">partialdiffdegree</csymbol>
    <apply><csymbol cd="list1">list</csymbol>
       n1   nk
    </apply>
     total-n1-nk
    <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> x1 </bvar>
    <bvar> xk </bvar>
     A
   </bind>
  </apply>
   x1
   xk
</apply>
```

If any of the bound variables do not use a `degree` qualifier, `<cn>1</cn>` should be used in place of the degree. If the original expression did not use the total degree qualifier then the second argument to partialdiffdegree should be the sum of the degrees, for example

```
<apply><csymbol cd="arith1">plus</csymbol>
   n1   nk
</apply>
```

With this rule, the expression

```
<apply><partialdiff/>
  <bvar><ci>x</ci><degree><ci>n</ci></degree></bvar>
  <bvar><ci>y</ci><degree><ci>m</ci></degree></bvar>
  <apply><sin/>
    <apply><times/><ci>x</ci><ci>y</ci></apply>
  </apply>
</apply>
```

is translated into

```
<apply>
  <apply><csymbol cd="calculus1">partialdiffdegree</csymbol>
    <apply><csymbol cd="list1">list</csymbol>
      <ci>n</ci><ci>m</ci>
    </apply>
    <apply><csymbol cd="arith1">plus</csymbol>
      <ci>n</ci><ci>m</ci>
    </apply>
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <bvar><ci>y</ci></bvar>
      <apply><csymbol cd="transc1">sin</csymbol>
        <apply><csymbol cd="arith1">times</csymbol>
          <ci>x</ci><ci>y</ci>
        </apply>
      </apply>
    </bind>
    <ci>x</ci>
    <ci>y</ci>
  </apply>
</apply>
```

4.4.4.4 *Divergence* `<divergence/>`

Class	unary-veccalc
Attributes	CommonAtt
Content	Empty
OM Symbols	divergence

The `divergence` element is the vector calculus divergence operator, often called div. It represents the divergence function which takes one argument which should be a vector of scalar-valued functions, intended to represent a vector-valued function, and returns the scalar-valued function giving the divergence of the argument.

Content MathML

```
<apply><divergence/><ci>a</ci></apply>
```

Sample Presentation

```
<mrow><mi>div</mi><mo>&#x2061;</mo><mfenced><mi>a</mi></mfenced></mrow>
```

$$\operatorname{div}(a)$$

Content MathML

```
<apply><divergence/>
  <ci type="vector">E</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>div</mi><mo>&#x2061;</mo><mfenced><mi>E</mi></mfenced></mrow>
```

$$\operatorname{div}(E)$$

```
<mrow><mo>&#x2207;</mo><mo>&#x22c5;</mo><mi>E</mi></mrow>
```

$$\nabla \cdot E$$

The functions defining the coordinates may be defined implicitly as expressions defined in terms of the coordinate names, in which case the coordinate names must be provided as bound variables.

Content MathML

```
<apply><divergence/>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <bvar><ci>z</ci></bvar>
  <vector>
    <apply><plus/><ci>x</ci><ci>y</ci></apply>
    <apply><plus/><ci>x</ci><ci>z</ci></apply>
    <apply><plus/><ci>z</ci><ci>y</ci></apply>
  </vector>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>div</mi>
 <mo>&#x2061;</mo>
 <mo>(</mo>
 <mtable>
  <mtr><mtd>
   <mi>x</mi>
   <mo>&#x21a6;</mo>
   <mrow><mi>x</mi><mo>+</mo><mi>y</mi></mrow>
  </mtd></mtr>
  <mtr><mtd>
   <mi>y</mi>
   <mo>&#x21a6;</mo>
   <mrow><mi>x</mi><mo>+</mo><mi>z</mi></mrow>
  </mtd></mtr>
  <mtr><mtd>
   <mi>z</mi>
   <mo>&#x21a6;</mo>
   <mrow><mi>z</mi><mo>+</mo><mi>y</mi></mrow>
  </mtd></mtr>
 </mtable>
 <mo>)</mo>
</mrow>
```

$$\operatorname{div}\begin{pmatrix} x \mapsto x+y \\ y \mapsto x+z \\ z \mapsto z+y \end{pmatrix}$$

4.4.4.5 *Gradient* `<grad/>`

Class	unary-veccalc
Attributes	CommonAtt
Content	Empty
OM Symbols	grad

The `grad` element is the vector calculus gradient operator, often called grad. It is used to represent the grad function, which takes one argument which should be a scalar-valued function and returns a vector of functions.

Content MathML

```
<apply><grad/><ci type="function">f</ci></apply>
```

Sample Presentation

```
<mrow><mi>grad</mi><mo>&#x2061;</mo><mfenced><mi>f</mi></mfenced></mrow>
```

$$\operatorname{grad}(f)$$

```
<mrow><mo>&#x2207;</mo><mo>&#x2061;</mo><mfenced><mi>f</mi></mfenced></mrow>
```

$$\nabla(f)$$

The functions defining the coordinates may be defined implicitly as expressions defined in terms of the coordinate names, in which case the coordinate names must be provided as bound variables.

Content MathML

```
<apply><grad/>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <bvar><ci>z</ci></bvar>
  <apply><times/><ci>x</ci><ci>y</ci><ci>z</ci></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>grad</mi>
 <mo>&#x2061;</mo>
 <mrow>
  <mo>(</mo>
  <mfenced><mi>x</mi><mi>y</mi><mi>z</mi></mfenced>
  <mo>&#x21a6;</mo>
  <mrow><mi>x</mi><mo>&#x2062;</mo><mi>y</mi><mo>&#x2062;</mo><mi>z</mi></mrow>
  <mo>)</mo>
 </mrow>
</mrow>
```

$$\operatorname{grad}\left((x,y,z)\mapsto xyz\right)$$

4.4.4.6 Curl `<curl/>`

Class	unary-veccalc
Attributes	CommonAtt
Content	Empty
OM Symbols	curl

The `curl` element is used to represent the curl function of vector calculus. It takes one argument which should be a vector of scalar-valued functions, intended to represent a vector-valued function, and returns a vector of functions.

Content MathML

```
<apply><curl/><ci>a</ci></apply>
```

Sample Presentation

```
<mrow><mi>curl</mi><mo>&#x2061;</mo><mfenced><mi>a</mi></mfenced></mrow>
```

$$\operatorname{curl}(a)$$

```
<mrow><mo>&#x2207;</mo><mo>&#xd7;</mo><mi>a</mi></mrow>
```

$$\nabla \times a$$

The functions defining the coordinates may be defined implicitly as expressions defined in terms of the coordinate names, in which case the coordinate names must be provided as bound variables.

4.4.4.7 *Laplacian* `<laplacian/>`

Class	unary-veccalc
Attributes	CommonAtt
Content	Empty
OM Symbols	Laplacian

The `laplacian` element represents the Laplacian operator of vector calculus. The Laplacian takes a single argument which is a vector of scalar-valued functions representing a vector-valued function, and returns a vector of functions.

Content MathML

```
<apply><laplacian/><ci type="vector">E</ci></apply>
```

Sample Presentation

```
<mrow>
 <msup><mo>&#x2207;</mo><mn>2</mn></msup>
 <mo>&#x2061;</mo>
 <mfenced><mi>E</mi></mfenced>
</mrow>
```

$$\nabla^2(E)$$

The functions defining the coordinates may be defined implicitly as expressions defined in terms of the coordinate names, in which case the coordinate names must be provided as bound variables.

Content MathML

```
<apply><laplacian/>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <bvar><ci>z</ci></bvar>
  <apply><ci>f</ci><ci>x</ci><ci>y</ci></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <msup><mo>&#x2207;</mo><mn>2</mn></msup>
 <mo>&#x2061;</mo>
 <mrow>
  <mo>(</mo>
  <mfenced><mi>x</mi><mi>y</mi><mi>z</mi></mfenced>
  <mo>&#x21a6;</mo>
  <mrow><mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi><mi>y</mi></mfenced></mrow>
  <mo>)</mo>
 </mrow>
</mrow>
```

$$\nabla^2 \left((x,y,z) \mapsto f\left(x,y\right)\right)$$

4.4.5 Theory of Sets

4.4.5.1 Set `<set>`

Class	nary-setlist-constructor
Attributes	CommonAtt
Content	ContExp*
Qualifiers	BvarQ,DomainQ
OM Symbols	set, multiset

The `set` represents a function which constructs mathematical sets from its arguments. It is an n-ary function. The members of the set to be constructed may be given explicitly as child elements of the constructor, or specified by rule as described in Section 4.3.1.1. There is no implied ordering to the elements of a set.

Content MathML

```
<set>
  <ci>a</ci><ci>b</ci><ci>c</ci>
</set>
```

Sample Presentation

```
<mrow>
 <mo>{</mo><mi>a</mi><mo>,</mo><mi>b</mi><mo>,</mo><mi>c</mi><mo>}</mo>
</mrow>
```

$$\{a,b,c\}$$

In general, a set can be constructed by providing a function and a domain of application. The elements of the set correspond to the values obtained by evaluating the function at the points of the domain.

Content MathML

```
<set>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><lt/><ci>x</ci><cn>5</cn></apply>
  </condition>
  <ci>x</ci>
</set>
```

Sample Presentation

```
<mrow>
 <mo>{</mo>
 <mi>x</mi>
 <mo>|</mo>
 <mrow><mi>x</mi><mo>&lt;</mo><mn>5</mn></mrow>
 <mo>}</mo>
</mrow>
```

$$\{x|x < 5\}$$

Content MathML

```
<set>
  <bvar><ci type="set">S</ci></bvar>
  <condition>
    <apply><in/><ci>S</ci><ci type="list">T</ci></apply>
  </condition>
  <ci>S</ci>
</set>
```

Sample Presentation

```
<mrow>
 <mo>{</mo>
 <mi>S</mi>
 <mo>|</mo>
 <mrow><mi>S</mi><mo>&#x2208;</mo><mi>T</mi></mrow>
 <mo>}</mo>
</mrow>
```

$$\{S|S \in T\}$$

Content MathML

```
<set>
  <bvar><ci> x </ci></bvar>
  <condition>
    <apply><and/>
      <apply><lt/><ci>x</ci><cn>5</cn></apply>
      <apply><in/><ci>x</ci><naturalnumbers/></apply>
    </apply>
  </condition>
  <ci>x</ci>
</set>
```

Sample Presentation

```
<mrow>
 <mo>{</mo>
 <mi>x</mi>
 <mo>|</mo>
 <mrow>
  <mrow><mo>(</mo><mi>x</mi><mo>&lt;</mo><mn>5</mn><mo>)</mo></mrow>
  <mo>&#x2227;</mo>
  <mrow><mi>x</mi><mo>&#x2208;</mo><mi mathvariant="double-struck">N</mi></mrow>
 </mrow>
 <mo>}</mo>
</mrow>
```

$$\{x|\,(x<5)\wedge x\in\mathbb{N}\}$$

4.4.5.2 List <list>

Class	nary-setlist-constructor
Attributes	CommonAtt
Content	ContExp*
Qualifiers	BvarQ,DomainQ
OM Symbols	interval_cc, list

The `list` elements represents the n-ary function which constructs a list from its arguments. Lists differ from sets in that there is an explicit order to the elements.

The list entries and order may be given explicitly.

Content MathML

```
<list>
  <ci>a</ci><ci>b</ci><ci>c</ci>
</list>
```

Sample Presentation

```
<mrow>
 <mo>(</mo><mi>a</mi><mo>,</mo><mi>b</mi><mo>,</mo><mi>c</mi><mo>)</mo>
</mrow>
```

$$(a,b,c)$$

In general a list can be constructed by providing a function and a domain of application. The elements

of the list correspond to the values obtained by evaluating the function at the points of the domain. When this method is used, the ordering of the list elements may not be clear, so the kind of ordering may be specified by the `order` attribute. Two orders are supported: lexicographic and numeric.

Content MathML

```
<list order="numeric">
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><lt/><ci>x</ci><cn>5</cn></apply>
  </condition>
</list>
```

Sample Presentation

```
<mrow>
 <mo>(</mo>
 <mi>x</mi>
 <mo>|</mo>
 <mrow><mi>x</mi><mo>&lt;</mo><mn>5</mn></mrow>
 <mo>)</mo>
</mrow>
```

$$(x|x<5)$$

4.4.5.3 Union `<union/>`

Class	nary-set
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	union

The `union` element is used to denote the n-ary union of sets. It takes sets as arguments, and denotes the set that contains all the elements that occur in any of them.

Arguments may be explicitly specified.

Content MathML

```
<apply><union/><ci>A</ci><ci>B</ci></apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x222a;</mo><mi>B</mi></mrow>
```

$$A \cup B$$

Arguments may also be specified using qualfier elements as described in Section 4.3.4.1. operator element can be used as a binding operator to construct the union over a collection of sets.

Content MathML

```
<apply><union/>
  <bvar><ci type="set">S</ci></bvar>
  <domainofapplication>
    <ci type="list">L</ci>
  </domainofapplication>
  <ci type="set"> S</ci>
</apply>
```

Sample Presentation

```
<mrow><munder><mo>&#x22c3;</mo><mi>L</mi></munder><mi>S</mi></mrow>
```

$$\bigcup_{L} S$$

4.4.5.4 Intersect `<intersect/>`

Class	nary-set
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	intersect

The `intersect` element is used to denote the n-ary intersection of sets. It takes sets as arguments, and denotes the set that contains all the elements that occur in all of them. Its arguments may be explicitly specified in the enclosing `apply` element, or specified using qualfier elements as described in Section 4.3.4.1.

Content MathML

```
<apply><intersect/>
  <ci type="set"> A </ci>
  <ci type="set"> B </ci>
</apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x2229;</mo><mi>B</mi></mrow>
```

$$A \cap B$$

Content MathML

```
<apply><intersect/>
  <bvar><ci type="set">S</ci></bvar>
  <domainofapplication><ci type="list">L</ci></domainofapplication>
  <ci type="set"> S </ci>
</apply>
```

Sample Presentation

```
<mrow><munder><mo>&#x22c2;</mo><mi>L</mi></munder><mi>S</mi></mrow>
```

$$\bigcap_{L} S$$

4.4.5.5 Set inclusion <in/>

Class	binary-set
Attributes	CommonAtt
Content	Empty
OM Symbols	in

The in element represents the set inclusion relation. It has two arguments, an element and a set. It is used to denote that the element is in the given set.

Content MathML

```
<apply><in/><ci>a</ci><ci type="set">A</ci></apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>&#x2208;</mo><mi>A</mi></mrow>
```

$$a \in A$$

When translating to Strict Content Markup, if the type has value "multiset", then the in from the multiset1 should be used instead.

4.4.5.6 Set exclusion <notin/>

Class	binary-set
Attributes	CommonAtt
Content	Empty
OM Symbols	notin

The notin represents the negated set inclusion relation. It has two arguments, an element and a set. It is used to denote that the element is not in the given set.

Content MathML

```
<apply><notin/><ci>a</ci><ci type="set">A</ci></apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>&#x2209;</mo><mi>A</mi></mrow>
```

$$a \notin A$$

When translating to Strict Content Markup, if the type has value "multiset", then the in from the multiset1 should be used instead.

4.4.5.7 Subset <subset/>

Class	nary-set-reln
Attributes	CommonAtt
Content	Empty
OM Symbols	subset

The subset element represents the subset relation. It is used to denote that the first argument is a subset of the second. As described in Section 4.3.4.3, it may also be used as an n-ary operator to express that each argument is a subset of its predecessor.

Content MathML

```
<apply><subset/>
  <ci type="set">A</ci>
  <ci type="set">B</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x2286;</mo><mi>B</mi></mrow>
```

$A \subseteq B$

4.4.5.8 *Proper Subset <prsubset/>*

Class	nary-set-reln
Attributes	CommonAtt
Content	Empty
OM Symbols	prsubset

The `prsubset` element represents the proper subset relation, i.e. that the first argument is a proper subset of the second. As described in Section 4.3.4.3, it may also be used as an n-ary operator to express that each argument is a proper subset of its predecessor.

Content MathML

```
<apply><prsubset/>
  <ci type="set">A</ci>
  <ci type="set">B</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x2282;</mo><mi>B</mi></mrow>
```

$A \subset B$

4.4.5.9 *Not Subset <notsubset/>*

Class	binary-set
Attributes	CommonAtt
Content	Empty
OM Symbols	notsubset

The `notsubset` element represents the negated subset relation. It is used to denote that the first argument is not a subset of the second.

Content MathML

```
<apply><notsubset/>
  <ci type="set">A</ci>
  <ci type="set">B</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x2288;</mo><mi>B</mi></mrow>
```

$A \not\subseteq B$

When translating to Strict Content Markup, if the `type` has value `"multiset"`, then the in from the multiset1 should be used instead.

4.4.5.10 Not Proper Subset `<notprsubset/>`

Class	binary-set
Attributes	CommonAtt
Content	Empty
OM Symbols	notprsubset

The `notprsubset` element represents the negated proper subset relation. It is used to denote that the first argument is not a proper subset of the second.

Content MathML

```
<apply><notprsubset/>
  <ci type="set">A</ci>
  <ci type="set">B</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x2284;</mo><mi>B</mi></mrow>
```

$$A \not\subset B$$

When translating to Strict Content Markup, if the `type` has value `"multiset"`, then the in from the multiset1 should be used instead.

4.4.5.11 Set Difference `<setdiff/>`

Class	binary-set
Attributes	CommonAtt
Content	Empty
OM Symbols	setdiff, setdiff

The `setdiff` element represents set difference operator. It takes two sets as arguments, and denotes the set that contains all the elements that occur in the first set, but not in the second.

Content MathML

```
<apply><setdiff/>
  <ci type="set">A</ci>
  <ci type="set">B</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x2216;</mo><mi>B</mi></mrow>
```

$$A \setminus B$$

When translating to Strict Content Markup, if the `type` has value `"multiset"`, then the in from the multiset1 should be used instead.

4.4.5.12 Cardinality <card/>

Class	unary-set
Attributes	CommonAtt
Content	Empty
OM Symbols	size, size

The card element represents the cardinality function, which takes a set argument and returns its cardinality, i.e. the number of elements in the set. The cardinality of a set is a non-negative integer, or an infinite cardinal number.

Content MathML

```
<apply><eq/>
  <apply><card/><ci>A</ci></apply>
  <cn>5</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mo>|</mo><mi>A</mi><mo>|</mo></mrow>
 <mo>=</mo>
 <mn>5</mn>
</mrow>
```

$$|A| = 5$$

When translating to Strict Content Markup, if the type has value "multiset", then the size from the multiset1 should be used instead.

4.4.5.13 Cartesian product <cartesianproduct/>

Class	nary-set
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	cartesian_product

The cartesianproduct element is used to represents the Cartesian product operator. It takes sets as arguments, which may be explicitly specified in the enclosing apply element, or specified using qualfier elements as described in Section 4.3.4.1.

Content MathML

```
<apply><cartesianproduct/><ci>A</ci><ci>B</ci></apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#xd7;</mo><mi>B</mi></mrow>
```

$$A \times B$$

4.4.6 Sequences and Series

4.4.6.1 Sum <sum/>

Class	sum
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	sum

The sum element represents the n-ary addition operator. The terms of the sum are normally specified by rule through the use of qualifiers. While it can be used with an explicit list of arguments, this is strongly discouraged, and the plus operator should be used instead in such situations.

The sum operator may be used either with or without explicit bound variables. When a bound variable is used, the sum element is followed by one or more bvar elements giving the index variables, followed by qualifiers giving the domain for the index variables. The final child in the enclosing apply is then an expression in the bound variables, and the terms of the sum are obtained by evaluating this expression at each point of the domain of the index variables. Depending on the structure of the domain, the domain of summation is often given by using uplimit and lowlimit to specify upper and lower limits for the sum.

When no bound variables are explicitly given, the final child of the enclosing apply element must be a function, and the terms of the sum are obtained by evaluating the function at each point of the domain specified by qualifiers.

Content MathML

```
<apply><sum/>
  <bvar><ci>x</ci></bvar>
  <lowlimit><ci>a</ci></lowlimit>
  <uplimit><ci>b</ci></uplimit>
  <apply><ci>f</ci><ci>x</ci></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <munderover>
  <mo>&#x2211;</mo>
  <mrow><mi>x</mi><mo>=</mo><mi>a</mi></mrow>
  <mi>b</mi>
 </munderover>
 <mrow><mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi></mfenced></mrow>
</mrow>
```

$$\sum_{x=a}^{b} f(x)$$

Content MathML

```
<apply><sum/>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><in/><ci>x</ci><ci type="set">B</ci></apply>
  </condition>
  <apply><ci type="function">f</ci><ci>x</ci></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <munder>
  <mo>&#x2211;</mo>
  <mrow><mi>x</mi><mo>&#x2208;</mo><mi>B</mi></mrow>
 </munder>
 <mrow><mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi></mfenced></mrow>
</mrow>
```

$$\sum_{x \in B} f(x)$$

Content MathML

```
<apply><sum/>
  <domainofapplication>
    <ci type="set">B</ci>
  </domainofapplication>
  <ci type="function">f</ci>
</apply>
```

Sample Presentation

```
<mrow><munder><mo>&#x2211;</mo><mi>B</mi></munder><mi>f</mi></mrow>
```

$$\sum_{B} f$$

Mapping to Strict Content MathML

When no explicit bound variables are used, no special rules are required to rewrite sums as Strict Content beyond the generic rules for rewriting expressions using qualifiers. However, when bound variables are used, it is necessary to introduce a `lambda` construction to rewrite the expression in the bound variables as a function.

Content MathML

```
<apply><sum/>
  <bvar><ci>i</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <uplimit><cn>100</cn></uplimit>
  <apply><power/><ci>x</ci><ci>i</ci></apply>
</apply>
```

Strict Content MathML equivalent

```
<apply><csymbol cd="arith1">sum</csymbol>
  <apply><csymbol cd="interval1">integer_interval</csymbol>
    <cn>0</cn>
    <cn>100</cn>
  </apply>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>i</ci></bvar>
    <apply><csymbol cd="arith1">power</csymbol><ci>x</ci><ci>i</ci></apply>
  </bind>
</apply>
```

4.4.6.2 *Product* <product/>

Class	product
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	product

The product element represents the n-ary multiplication operator. The terms of the product are normally specified by rule through the use of qualifiers. While it can be used with an explicit list of arguments, this is strongly discouraged, and the times operator should be used instead in such situations.

The product operator may be used either with or without explicit bound variables. When a bound variable is used, the product element is followed by one or more bvar elements giving the index variables, followed by qualifiers giving the domain for the index variables. The final child in the enclosing apply is then an expression in the bound variables, and the terms of the product are obtained by evaluating this expression at each point of the domain. Depending on the structure of the domain, it is commonly given using uplimit and lowlimit qualifiers.

When no bound variables are explicitly given, the final child of the enclosing apply element must be a function, and the terms of the product are obtained by evaluating the function at each point of the domain specified by qualifiers.

Content MathML

```
<apply><product/>
  <bvar><ci>x</ci></bvar>
  <lowlimit><ci>a</ci></lowlimit>
  <uplimit><ci>b</ci></uplimit>
  <apply><ci type="function">f</ci>
    <ci>x</ci>
  </apply>
</apply>
```

Sample Presentation

```
<mrow>
 <munderover>
  <mo>&#x220f;</mo>
  <mrow><mi>x</mi><mo>=</mo><mi>a</mi></mrow>
  <mi>b</mi>
 </munderover>
 <mrow><mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi></mfenced></mrow>
</mrow>
```

$$\prod_{x=a}^{b} f(x)$$

Content MathML

```
<apply><product/>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><in/>
      <ci>x</ci>
      <ci type="set">B</ci>
    </apply>
  </condition>
  <apply><ci>f</ci><ci>x</ci></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <munder>
  <mo>&#x220f;</mo>
  <mrow><mi>x</mi><mo>&#x2208;</mo><mi>B</mi></mrow>
 </munder>
 <mrow><mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi></mfenced></mrow>
</mrow>
```

$$\prod_{x \in B} f(x)$$

Mapping to Strict Content MathML

When no explicit bound variables are used, no special rules are required to rewrite products as Strict Content beyond the generic rules for rewriting expressions using qualifiers. However, when bound variables are used, it is necessary to introduce a `lambda` construction to rewrite the expression in the bound variables as a function.

Content MathML

```
<apply><product/>
  <bvar><ci>i</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <uplimit><cn>100</cn></uplimit>
  <apply><power/><ci>x</ci><ci>i</ci></apply>
</apply>
```

Strict Content MathML equivalent

```
<apply><csymbol cd="arith1">product</csymbol>
  <apply><csymbol cd="interval1">integer_interval</csymbol>
    <cn>0</cn>
    <cn>100</cn>
  </apply>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>i</ci></bvar>
    <apply><csymbol cd="arith1">power</csymbol><ci>x</ci><ci>i</ci></apply>
  </bind>
</apply>
```

4.4.6.3 *Limits* `<limit/>`

Class	limit
Attributes	CommonAtt
Content	Empty
Qualifiers	lowlimit, condition
OM Symbols	limit, both_sides, above, below, null

The `limit` element represents the operation of taking a limit of a sequence. The limit point is expressed by specifying a `lowlimit` and a `bvar`, or by specifying a `condition` on one or more bound variables.

Content MathML

```
<apply><limit/>
  <bvar><ci>x</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <apply><sin/><ci>x</ci></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <munder><mi>lim</mi><mrow><mi>x</mi><mo>&#x2192;</mo><mn>0</mn></mrow></munder>
 <mrow><mi>sin</mi><mo>&#x2061;</mo><mi>x</mi></mrow>
</mrow>
```

$$\lim_{x \to 0} \sin x$$

Content MathML

```
<apply><limit/>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><tendsto/><ci>x</ci><cn>0</cn></apply>
  </condition>
  <apply><sin/><ci>x</ci></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <munder><mi>lim</mi><mrow><mi>x</mi><mo>&#x2192;</mo><mn>0</mn></mrow></munder>
 <mrow><mi>sin</mi><mo>&#x2061;</mo><mi>x</mi></mrow>
</mrow>
```

$$\lim_{x \to 0} \sin x$$

Content MathML

```
<apply><limit/>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><tendsto type="above"/><ci>x</ci><ci>a</ci></apply>
  </condition>
  <apply><sin/><ci>x</ci></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <munder><mi>lim</mi><mrow><mi>x</mi><mo>&#x2192;</mo><mi>a</mi></mrow></munder>
 <mrow><mi>sin</mi><mo>&#x2061;</mo><mi>x</mi></mrow>
</mrow>
```

$$\lim_{x \to a} \sin x$$

The direction from which a limiting value is approached is given as an argument limit in Strict Content MathML, which supplies the direction specifier symbols both_sides, above, and below for this purpose. The first correspond to the values `"all"`, `"above"`, and `"below"` of the `type` attribute of the `tendsto` element below. The null symbol corresponds to the case where no `type` attribute is present. We translate

Rewrite: limits condition

```
<apply><limit/>
  <bvar> x </bvar>
  <condition>
    <apply><tendsto/> x <cn>0</cn></apply>
  </condition>
   expression-in-x
</apply>
```

Strict Content MathML equivalent

```
<apply><csymbol cd="limit1">limit</csymbol>
  <cn>0</cn>
  <csymbol cd="limit1">null</csymbol>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar> x </bvar>
     expression-in-x
  </bind>
</apply>
```

where *expression-in-x* is an arbitrary expression involving the bound variable(s), and the choice of symbol, null depends on the `type` attribute of the the `tendsto` element as described above.

4.4.6.4 Tends To `<tendsto/>`

Class	binary-reln
Attributes	CommonAtt
Content	Empty
OM Symbols	limit

The `tendsto` element is used to express the relation that a quantity is tending to a specified value. While this is used primarily as part of the statement of a mathematical limit, it exists as a construct on its own to allow one to capture mathematical statements such as "As x tends to y," and to provide a building block to construct more general kinds of limits.

The `tendsto` element takes the attributes `type` to set the direction from which the limiting value is approached.

Content MathML

```
<apply><tendsto type="above"/>
  <apply><power/><ci>x</ci><cn>2</cn></apply>
   <apply><power/><ci>a</ci><cn>2</cn></apply>
</apply>
```

Sample Presentation

```
<mrow>
 <msup><mi>x</mi><mn>2</mn></msup>
 <mo>&#x2192;</mo>
 <msup><mi>a</mi><mn>2</mn></msup>
</mrow>
```

$$x^2 \to a^2$$

Content MathML

```
<apply><tendsto/>
  <vector><ci>x</ci><ci>y</ci></vector>
   <vector>
     <apply><ci type="function">f</ci><ci>x</ci><ci>y</ci></apply>
     <apply><ci type="function">g</ci><ci>x</ci><ci>y</ci></apply>
   </vector>
</apply>
```

Sample Presentation

```
<mfenced><mtable>
  <mtr><mtd><mi>x</mi></mtd></mtr>
  <mtr><mtd><mi>y</mi></mtd></mtr>
</mtable></mfenced>
<mo>&#x2192;</mo>
<mfenced><mtable>
  <mtr><mtd>
    <mi>f</mi><mo>&#x2061;</mo><mfenced><mi>x</mi><mi>y</mi></mfenced>
  </mtd></mtr>
  <mtr><mtd>
    <mi>g</mi><mo>&#x2061;</mo><mfenced><mi>x</mi><mi>y</mi></mfenced>
  </mtd></mtr>
</mtable></mfenced>
```

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$$

Mapping to Strict Content MathML

The usage of `tendsto` to qualify a limit is formally defined by writing the expression in Strict Content MathML via the rule Rewrite: limits condition. The meanings of other more idiomatic uses of `tendsto` are not formally defined by this specification. When rewriting these cases to Strict Content MathML, `tendsto` should be rewritten to an annotated identifier as shown below.

Rewrite: tendsto

```
<tendsto/>
```

Strict Content MathML equivalent:

```
<semantics>
 <ci>tendsto</ci>
 <annotation-xml cd="altenc" encoding="MathML_encoding">
  <tendsto/>
 </annotation-xml>
</semantics>
```

4.4.7 Elementary classical functions

4.4.7.1 Common trigonometric functions

Class	unary-elementary
Attributes	CommonAtt
Content	Empty
OM Symbols	sin

sin	cos	tan	sec	csc	cot
sinh	cosh	tanh	sech	csch	coth
arcsin	arccos	arctan	arccosh	arccot	arccoth
arccsc	arccsch	arcsec	arcsech	arcsinh	arctanh

These operator elements denote the standard trigonometric and hyperbolic functions and their inverses. Since their standard interpretations are widely known, they are discussed as a group. In the case of inverse functions there are differing definitions in use. For maximum interoperability applications evaluating such expressions should follow the definitions in [Abramowitz1997].

Content MathML

```
<apply><sin/><ci>x</ci></apply>
```

Sample Presentation

```
<mrow><mi>sin</mi><mo>&#x2061;</mo><mi>x</mi></mrow>
```

$$\sin x$$

Content MathML

```
<apply><sin/>
  <apply><plus/>
    <apply><cos/><ci>x</ci></apply>
    <apply><power/><ci>x</ci><cn>3</cn></apply>
  </apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>sin</mi>
 <mo>&#x2061;</mo>
 <mrow>
  <mo>(</mo>
  <mrow><mi>cos</mi><mo>&#x2061;</mo><mi>x</mi></mrow>
  <mo>+</mo>
  <msup><mi>x</mi><mn>3</mn></msup>
  <mo>)</mo>
 </mrow>
</mrow>
```

$$\sin\left(\cos x + x^3\right)$$

4.4.7.2 Exponential <exp/>

Class	unary-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	exp

The exp element represents the exponentiation function associated with the inverse of the ln function. It takes one argument.

Content MathML

```
<apply><exp/><ci>x</ci></apply>
```

Sample Presentation

```
<msup><mi>e</mi><mi>x</mi></msup>
```

e^x

4.4.7.3 Natural Logarithm <ln/>

Class	unary-functional
Attributes	CommonAtt
Content	Empty
OM Symbols	ln

The ln element represents the natural logarithm function.

Content MathML

```
<apply><ln/><ci>a</ci></apply>
```

Sample Presentation

```
<mrow><mi>ln</mi><mo>&#x2061;</mo><mi>a</mi></mrow>
```

$\ln a$

4.4.7.4 Logarithm <log/>

Class	unary-functional
Attributes	CommonAtt
Content	Empty
Qualifiers	logbase
OM Symbols	log

The log elements represents the logarithm function relative to a given base. When present, the logbase qualifier specifies the base. Otherwise, the base is assumed to be 10. apply.

Content MathML

```
<apply><log/>
  <logbase><cn>3</cn></logbase>
  <ci>x</ci>
</apply>
```

Sample Presentation

```
<mrow><msub><mi>log</mi><mn>3</mn></msub><mo>&#x2061;</mo><mi>x</mi></mrow>
```

$$\log_3 x$$

Content MathML

```
<apply><log/><ci>x</ci></apply>
```

Sample Presentation

```
<mrow><mi>log</mi><mo>&#x2061;</mo><mi>x</mi></mrow>
```

$$\log x$$

Mapping to Strict Content MathML

When mapping `log` to Strict Content, one uses the log, symbol denoting the function that returns the log of it's second argument with respect to the base specified by the first argument. When `logbase` is present, it determines the base. Otherwise, the default base of 10 must be explicitly provided in Strict markup. See the following example.

```
<apply><plus/>
  <apply>
    <log/>
    <logbase><cn>2</cn></logbase>
    <ci>x</ci>
  </apply>
  <apply>
    <log/>
    <ci>y</ci>
  </apply>
</apply>
```

Strict Content MathML equivalent:

```
<apply>
  <csymbol cd="arith1">plus</csymbol>
  <apply>
    <csymbol cd="transc1">log</csymbol>
    <ci>x</ci>
    <cn>2</cn>
  </apply>
  <apply>
    <csymbol cd="transc1">log</csymbol>
    <ci>y</ci>
    <cn>10</cn>
  </apply>
</apply>
```

4.4.8 Statistics

4.4.8.1 Mean <mean/>

Class	nary-stats
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	mean, mean

The mean element represents the function returning arithmetic mean or average of a data set or random variable. If it is used on a data set, then the mean element corresponds to the mean symbol from the s_data1 content dictionary. When it is applied to a random variable, then it corresponds to the mean symbol from the s_dist1 CD.

Content MathML

```
<apply><mean/>
  <cn>3</cn><cn>4</cn><cn>3</cn><cn>7</cn><cn>4</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mo>&#x2329;</mo>
 <mn>3</mn><mo>,</mo><mn>4</mn><mo>,</mo><mn>3</mn>
 <mo>,</mo><mn>7</mn><mo>,</mo><mn>4</mn>
 <mo>&#x232a;</mo>
</mrow>
```

$$\langle 3,4,3,7,4 \rangle$$

Content MathML

```
<apply><mean/><ci>X</ci></apply>
```

Sample Presentation

```
<mrow><mo>&#x2329;</mo><mi>X</mi><mo>&#x232a;</mo></mrow>
```

$$\langle X \rangle$$

```
<mover><mi>X</mi><mo>&#xaf;</mo></mover>
```

$$\overline{X}$$

Mapping to Strict Markup

When the mean element is applied to an explicit list of arguments, the translation to Strict Content markup is direct, using the mean symbol from the s_data1 content dictionary, as described in Rewrite: element. When it is applied to a distribution, then the mean symbol from the s_dist1 content dictionary should be used. In the case with qualifiers use Rewrite: n-ary domainofapplication with the same caveat.

4.4.8.2 Standard Deviation <sdev/>

Class	nary-stats
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	sdev, sdev

sdev is the operator element representing the standard deviation of a data set or random variable.

Content MathML

```
<apply><sdev/>
  <cn>3</cn><cn>4</cn><cn>2</cn><cn>2</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mo>&#x3c3;</mo>
 <mo>&#x2061;</mo>
 <mfenced><mn>3</mn><mn>4</mn><mn>2</mn><mn>2</mn></mfenced>
</mrow>
```

$$\sigma(3,4,2,2)$$

Content MathML

```
<apply><sdev/>
  <ci type="discrete_random_variable">X</ci>
</apply>
```

Sample Presentation

```
<mrow><mo>&#x3c3;</mo><mo>&#x2061;</mo><mfenced><mi>X</mi></mfenced></mrow>
```

$$\sigma(X)$$

Mapping to Strict Markup

When the sdev element is applied to an explicit list of arguments, the translation to Strict Content markup is direct, using the sdev symbol from the s_data1 content dictionary, as described in Rewrite: element. When it is applied to a distribution, then the sdev symbol from the s_dist1 content dictionary should be used. In the case with qualifiers use Rewrite: n-ary domainofapplication with the same caveat.

4.4.8.3 *Variance* <variance/>

Class	nary-stats
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	variance, variance

variance is the operator element representing the standard deviation of a data set or random variable. If it is used on a data set, then the variance element corresponds to the variance from the s_data1 content dictionary, if it is used on a random variable, then it corresponds to the variance from the s_dist1 CD.

Content MathML

```
<apply><variance/>
  <cn>3</cn><cn>4</cn><cn>2</cn><cn>2</cn>
</apply>
```

Sample Presentation

```
<msup>
 <mrow>
  <mo>&#x3c3;</mo>
  <mo>&#x2061;</mo>
  <mfenced><mn>3</mn><mn>4</mn><mn>2</mn><mn>2</mn></mfenced>
 </mrow>
 <mn>2</mn>
</msup>
```

$$\sigma(3,4,2,2)^2$$

Content MathML

```
<apply><variance/>
  <ci type="discrete_random_variable"> X</ci>
</apply>
```

Sample Presentation

```
<msup>
 <mrow><mo>&#x3c3;</mo><mo>&#x2061;</mo><mfenced><mi>X</mi></mfenced></mrow>
 <mn>2</mn>
</msup>
```

$$\sigma(X)^2$$

Mapping to Strict Markup

When the `variance` element is applied to an explicit list of arguments, the translation to Strict Content markup is direct, using the variance symbol from the s_data1 content dictionary, as described in Rewrite: element. When it is applied to a distribution, then the variance symbol from the s_dist1 content dictionary should be used. In the case with qualifiers use Rewrite: n-ary domainofapplication with the same caveat.

4.4.8.4 Median `<median/>`

Class	nary-stats
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	median

This symbol represents an n-ary function denoting the median of its arguments. That is, if the data were placed in ascending order then it denotes the middle one (in the case of an odd amount of data) or the average of the middle two (in the case of an even amount of data).

Content MathML

```
<apply><median/>
  <cn>3</cn><cn>4</cn><cn>2</cn><cn>2</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>median</mi>
 <mo>&#x2061;</mo>
 <mfenced><mn>3</mn><mn>4</mn><mn>2</mn><mn>2</mn></mfenced>
</mrow>
```

$$\text{median}\,(3,4,2,2)$$

Mapping to Strict Markup

When the `median` element is applied to an explicit list of arguments, the translation to Strict Content markup is direct, using the median symbol from the s_data1 content dictionary, as described in Rewrite: element.

4.4.8.5 Mode `<mode/>`

Class	nary-stats
Attributes	CommonAtt
Content	Empty
Qualifiers	BvarQ,DomainQ
OM Symbols	mode

This symbol represents an n-ary function denoting the mode of its arguments. That is the value which occurs with the greatest frequency.

Content MathML

```
<apply><mode/>
  <cn>3</cn><cn>4</cn><cn>2</cn><cn>2</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>mode</mi>
 <mo>&#x2061;</mo>
 <mfenced><mn>3</mn><mn>4</mn><mn>2</mn><mn>2</mn></mfenced>
</mrow>
```

$$\text{mode}\,(3,4,2,2)$$

Mapping to Strict Markup

When the `mode` element is applied to an explicit list of arguments, the translation to Strict Content markup is direct, using the mode symbol from the s_data1 content dictionary, as described in Rewrite: element.

4.4.8.6 Moment (<moment/>, <momentabout>)

Class	unary-functional
Attributes	CommonAtt
Content	Empty
Qualifiers	degree, momentabout
OM Symbols	moment, moment

The `moment` element is used to denote the *i*th moment of a set of data set or random variable. The `moment` function accepts the `degree` and `momentabout` qualifiers. If present, the `degree` schema denotes the order of the moment. Otherwise, the moment is assumed to be the first order moment. When used with `moment`, the `degree` schema is expected to contain a single child. If present, the `momentabout` schema denotes the point about which the moment is taken. Otherwise, the moment is assumed to be the moment about zero.

Content MathML

```
<apply><moment/>
  <degree><cn>3</cn></degree>
  <momentabout><mean/></momentabout>
  <cn>6</cn><cn>4</cn><cn>2</cn><cn>2</cn><cn>5</cn>
</apply>
```

Sample Presentation

```
<msub>
 <mrow>
  <mo>&#x2329;</mo>
  <msup>
   <mfenced><mn>6</mn><mn>4</mn><mn>2</mn><mn>2</mn><mn>5</mn></mfenced>
   <mn>3</mn>
  </msup>
  <mo>&#x232a;</mo>
 </mrow>
 <mi>mean</mi>
</msub>
```

$$\langle (6,4,2,2,5)^3 \rangle_{\text{mean}}$$

Content MathML

```
<apply><moment/>
  <degree><cn>3</cn></degree>
  <momentabout><ci>p</ci></momentabout>
  <ci>X</ci>
</apply>
```

Sample Presentation

```
<msub>
 <mrow><mo>&#x2329;</mo><msup><mi>X</mi><mn>3</mn></msup><mo>&#x232a;</mo></mrow>
 <mi>p</mi>
</msub>
```

$$\langle X^3 \rangle_p$$

Mapping to Strict Markup

When rewriting to Strict Markup, the moment is used. It takes the degreee as the first argument, the point as the second, and the dataset or random variable.

```
<apply><moment/>
  <degree><cn>3</cn></degree>
  <momentabout><ci>p</ci></momentabout>
  <ci>X</ci>
</apply>
```

Strict Content MathML equivalent

```
<apply><csymbol cd="s_dist1">moment</csymbol>
  <cn>3</cn>
  <ci>p</ci>
  <ci>X</ci>
</apply>
```

4.4.9 Linear Algebra

4.4.9.1 Vector `<vector>`

Class	nary-constructor
Attributes	CommonAtt
Qualifiers	BvarQ,DomainQ
Content	ContExp*
OM Symbol	vector

A vector is an ordered n-tuple of values representing an element of an n-dimensional vector space. The "values" are all from the same ring, typically real or complex. Where orientation is important, such as for pre or post multiplication by a matrix a vector is treated as a row vector and its transpose is treated a column vector.

For purposes of interaction with matrices and matrix multiplication, vectors are regarded as equivalent to a matrix consisting of a single column, and the transpose of a vector behaves the same as a matrix consisting of a single row. Note that vectors may be rendered either as a single column or row.

`vector` is a *constructor* element (see ???).

Content MathML

```
<vector>
  <apply><plus/><ci>x</ci><ci>y</ci></apply>
  <cn>3</cn>
  <cn>7</cn>
</vector>
```

Sample Presentation

```
<mrow>
 <mo>(</mo>
 <mtable>
  <mtr><mtd><mrow><mi>x</mi><mo>+</mo><mi>y</mi></mrow></mtd></mtr>
  <mtr><mtd><mn>3</mn></mtd></mtr>
  <mtr><mtd><mn>7</mn></mtd></mtr>
 </mtable>
 <mo>)</mo>
</mrow>
```

$$\begin{pmatrix} x+y \\ 3 \\ 7 \end{pmatrix}$$

```
<mfenced>
  <mrow><mi>x</mi><mo>+</mo><mi>y</mi></mrow>
  <mn>3</mn>
  <mn>7</mn>
</mfenced>
```

$$(x+y,3,7)$$

The `vector` element constructs vectors from an n-dimensional vector space so that its n child elements typically represent real or complex valued scalars as in the three-element vector

In general a vector can be constructed by providing a function and a 1-dimensional domain of application. The entries of the vector correspond to the values obtained by evaluating the function at the points of the domain.

Content MathML

```
<vector>
  <bvar><ci>x</ci></bvar>
  <domainofapplication>
    <interval type="integer"><cn>1</cn><cn>7</cn></interval>
  </domainofapplication>
  <apply><power/><ci>x</ci><cn>2</cn>
  </apply>
</vector>
```

Sample Presentation

```
<mrow>
 <mo>[</mo>
 <msup><mi>x</mi><mn>2</mn></msup>
 <mo>|</mo>
 <mi>x</mi>
 <mo>&#x2208;</mo>
 <mfenced open="[" close="]"><mn>1</mn><mn>7</mn></mfenced>
 <mo>]</mo>
</mrow>
```

$$[x^2 | x \in [1,7]]$$

4.4.9.2 Matrix `<matrix>`

Class	nary-constructor
Attributes	CommonAtt
Qualifiers	BvarQ,DomainQ
Content	ContExp*
OM Symbol	matrix

A vector is an ordered n-tuple of values representing an element of an n-dimensional vector space. The "values" are all from the same ring, typically real or complex. Where orientation is important, such as for pre or post multiplication by a matrix a vector is treated as a row vector and its transpose is treated a column vector.

For purposes of interaction with matrices and matrix multiplication, vectors are regarded as equivalent to a matrix consisting of a single column, and the transpose of a vector behaves the same as a matrix consisting of a single row. Note that vectors may be rendered either as a single column or row.

Note that the behavior of the `matrix` and `matrixrow` elements is substantially different from the `mtable` and `mtr` presentation elements.

`matrix` is a *constructor* element (see ???).

In general a matrix can be constructed by providing a function and a 2-dimensional domain of application. The entries of the matrix correspond to the values obtained by evaluating the function at the points of the domain. The qualifications defined by a `domainofapplication` element can also be abbreviated in several ways including a `condition` element placing constraints directly on bound variables and an expression in those variables.

Content MathML

```
<matrix>
  <bvar><ci type="integer">i</ci></bvar>
  <bvar><ci type="integer">j</ci></bvar>
  <condition>
    <apply><and/>
      <apply><in/>
        <ci>i</ci>
        <interval><ci>1</ci><ci>5</ci></interval>
      </apply>
      <apply><in/>
        <ci>j</ci>
        <interval><ci>5</ci><ci>9</ci></interval>
      </apply>
    </apply>
  </condition>
  <apply><power/><ci>i</ci><ci>j</ci></apply>
</matrix>
```

Sample Presentation

```
<mrow>
 <mo>[</mo>
 <msub><mi>m</mi><mrow><mi>i</mi><mo>,</mo><mi>j</mi></mrow></msub>
 <mo>|</mo>
 <mrow>
  <msub><mi>m</mi><mrow><mi>i</mi><mo>,</mo><mi>j</mi></mrow></msub>
  <mo>=</mo>
  <msup><mi>i</mi><mi>j</mi></msup>
 </mrow>
 <mo>;</mo>
 <mrow>
  <mrow>
   <mi>i</mi>
   <mo>&#x2208;</mo>
   <mfenced open="[" close="]"><mi>1</mi><mi>5</mi></mfenced>
  </mrow>
  <mo>&#x2227;</mo>
  <mrow>
   <mi>j</mi>
   <mo>&#x2208;</mo>
   <mfenced open="[" close="]"><mi>5</mi><mi>9</mi></mfenced>
  </mrow>
 </mrow>
 <mo>]</mo>
</mrow>
```

$$[m_{i,j} \big| m_{i,j} = i^j; i \in [1,5] \wedge j \in [5,9]]$$

4.4.9.3 Matrix row <matrixrow>

Class	nary-constructor
Attributes	CommonAtt
Qualifiers	BvarQ,DomainQ
Content	ContExp*
OM Symbol	matrixrow

This symbol is an n-ary constructor used to represent rows of matrices. Its arguments should be members of a ring.

Matrix rows are not directly rendered by themselves outside of the context of a matrix.

4.4.9.4 Determinant <determinant/>

Class	unary-linalg
Attributes	CommonAtt
Content	Empty
OM Symbols	determinant

This symbol denotes the unary function which returns the determinant of its argument, the argument should be a square matrix.

Content MathML

```
<apply><determinant/>
  <ci type="matrix">A</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>det</mi><mo>&#x2061;</mo><mi>A</mi></mrow>
```

$\det A$

4.4.9.5 Transpose <transpose/>

Class	unary-linalg
Attributes	CommonAtt
Content	Empty
OM Symbols	transpose

This symbol represents a unary function that denotes the transpose of the given matrix or vector.

Content MathML

```
<apply><transpose/>
  <ci type="matrix">A</ci>
</apply>
```

Sample Presentation

```
<msup><mi>A</mi><mi>T</mi></msup>
```

A^T

4.4.9.6 Selector `<selector/>`

Class	nary-linalg
Attributes	CommonAtt
Content	Empty
OM Symbols	vector_selector, matrix_selector

The `selector` element is the operator for indexing into vectors matrices and lists. It accepts one or more arguments. The first argument identifies the vector, matrix or list from which the selection is taking place, and the second and subsequent arguments, if any, indicate the kind of selection taking place.

When `selector` is used with a single argument, it should be interpreted as giving the sequence of all elements in the list, vector or matrix given. The ordering of elements in the sequence for a matrix is understood to be first by column, then by row. That is, for a matrix $(a_{i,j})$, where the indices denote row and column, the ordering would be $a_{1,1}$, $a_{1,2}$, ... $a_{2,1}$, $a_{2,2}$ ... etc.

When three arguments are given, the last one is ignored for a list or vector, and in the case of a matrix, the second and third arguments specify the row and column of the selected element.

When two arguments are given, and the first is a vector or list, the second argument specifies an element in the list or vector.

Content MathML

```
<apply><selector/><ci type="vector">V</ci><cn>1</cn></apply>
```

Sample Presentation

```
<msub><mi>V</mi><mn>1</mn></msub>
```

V_1

Content MathML

```
<apply><eq/>
  <apply><selector/>
    <matrix>
      <matrixrow><cn>1</cn><cn>2</cn></matrixrow>
      <matrixrow><cn>3</cn><cn>4</cn></matrixrow>
    </matrix>
    <cn>1</cn>
  </apply>
  <matrixrow><cn>1</cn><cn>2</cn></matrixrow>
</apply>
```

Sample Presentation

```
<mrow>
 <msub>
  <mrow>
   <mo>(</mo>
   <mtable>
    <mtr><mtd><mn>1</mn></mtd><mtd><mn>2</mn></mtd></mtr>
    <mtr><mtd><mn>3</mn></mtd><mtd><mn>4</mn></mtd></mtr>
   </mtable>
   <mo>)</mo>
  </mrow>
  <mn>1</mn>
 </msub>
 <mo>=</mo>
 <mtable><mtr><mtd><mn>1</mn></mtd><mtd><mn>2</mn></mtd></mtr></mtable>
</mrow>
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}_1 = 1 \quad 2$$

4.4.9.7 *Vector product* `<vectorproduct/>`

Class	binary-linalg
Attributes	CommonAtt
Content	Empty
OM Symbols	vectorproduct

This symbol represents the vector product function. It takes two three dimensional vector arguments and returns a three dimensional vector.

Content MathML

```
<apply><eq/>
  <apply><vectorproduct/>
    <ci type="vector"> A </ci>
    <ci type="vector"> B </ci>
 </apply>
  <apply><times/>
    <ci>a</ci>
    <ci>b</ci>
    <apply><sin/><ci>&#x3b8;</ci></apply>
    <ci type="vector"> N </ci>
  </apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mi>A</mi><mo>&#xd7;</mo><mi>B</mi></mrow>
 <mo>=</mo>
 <mrow>
  <mi>a</mi>
  <mo>&#x2062;</mo>
  <mi>b</mi>
  <mo>&#x2062;</mo>
  <mrow><mi>sin</mi><mo>&#x2061;</mo><mi>&#x3b8;</mi></mrow>
  <mo>&#x2062;</mo>
  <mi>N</mi>
 </mrow>
</mrow>
```

$$A \times B = ab\sin\theta N$$

4.4.9.8 *Scalar product* `<scalarproduct/>`

Class	binary-linalg
Attributes	CommonAtt
Content	Empty
OM Symbols	scalarproduct

This symbol represents the scalar product function. It takes two vector arguments and returns a scalar value.

Content MathML

```
<apply><eq/>
  <apply><scalarproduct/>
    <ci type="vector">A</ci>
    <ci type="vector">B</ci>
  </apply>
  <apply><times/>
    <ci>a</ci>
    <ci>b</ci>
    <apply><cos/><ci>&#x3b8;</ci></apply>
  </apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mi>A</mi><mo>.</mo><mi>B</mi></mrow>
 <mo>=</mo>
 <mrow>
  <mi>a</mi>
  <mo>&#x2062;</mo>
  <mi>b</mi>
  <mo>&#x2062;</mo>
  <mrow><mi>cos</mi><mo>&#x2061;</mo><mi>&#x3b8;</mi></mrow>
 </mrow>
</mrow>
```

$$A.B = ab\cos\theta$$

4.4.9.9 *Outer product* `<outerproduct/>`

Class	binary-linalg
Attributes	CommonAtt
Content	Empty
OM Symbols	outerproduct

This symbol represents the outer product function. It takes two vector arguments and returns a matrix.

Content MathML

```
<apply><outerproduct/>
  <ci type="vector">A</ci>
  <ci type="vector">B</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x2297;</mo><mi>B</mi></mrow>
```

$$A \otimes B$$

4.4.10 Constant and Symbol Elements

This section explains the use of the Constant and Symbol elements.

4.4.10.1 integers <integers/>

Class	constant-set
Attributes	CommonAtt
Content	Empty
OM Symbols	Z

This symbol represents the set of integers, positive, negative and zero.

Content MathML

```
<apply><in/>
  <cn type="integer"> 42 </cn>
  <integers/>
</apply>
```

Sample Presentation

```
<mrow><mn>42</mn><mo>&#x2208;</mo><mi mathvariant="double-struck">Z</mi></mrow>
```

$$42 \in \mathbb{Z}$$

4.4.10.2 reals <reals/>

Class	constant-set
Attributes	CommonAtt
Content	Empty
OM Symbols	R

This symbol represents the set of real numbers.

Content MathML

```
<apply><in/>
  <cn type="real"> 44.997</cn>
  <reals/>
</apply>
```

Sample Presentation

```
<mrow><mn>44.997</mn><mo>&#x2208;</mo><mi mathvariant="double-struck">R</mi></mrow>
```

$$44.997 \in \mathbb{R}$$

4.4.10.3 Rational Numbers <rationals/>

Class	constant-set
Attributes	CommonAtt
Content	Empty
OM Symbols	Q

This symbol represents the set of rational numbers.

Content MathML

```
<apply><in/>
  <cn type="rational"> 22 <sep/>7</cn>
  <rationals/>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mn>22</mn><mo>/</mo><mn>7</mn></mrow>
 <mo>&#x2208;</mo>
 <mi mathvariant="double-struck">Q</mi>
</mrow>
```

$$22/7 \in \mathbb{Q}$$

4.4.10.4 Natural Numbers <naturalnumbers/>

Class	constant-set
Attributes	CommonAtt
Content	Empty
OM Symbols	N

This symbol represents the set of natural numbers (including zero).

Content MathML

```
<apply><in/>
  <cn type="integer">1729</cn>
  <naturalnumbers/>
</apply>
```

Sample Presentation

```
<mrow><mn>1729</mn><mo>&#x2208;</mo><mi mathvariant="double-struck">N</mi></mrow>
```

$$1729 \in \mathbb{N}$$

4.4.10.5 complexes <complexes/>

Class	constant-set
Attributes	CommonAtt
Content	Empty
OM Symbols	C

This symbol represents the set of complex numbers.

Content MathML

```
<apply><in/>
  <cn type="complex-cartesian">17<sep/>29</cn>
  <complexes/>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mn>17</mn><mo>+</mo><mn>29</mn><mo>&#x2062;</mo><mi>i</mi></mrow>
 <mo>&#x2208;</mo>
 <mi mathvariant="double-struck">C</mi>
</mrow>
```

$$17 + 29i \in \mathbb{C}$$

4.4.10.6 *primes* `<primes/>`

Class	constant-set
Attributes	CommonAtt
Content	Empty
OM Symbols	P

This symbol represents the set of positive prime numbers.

Content MathML

```
<apply><in/>
  <cn type="integer">17</cn>
  <primes/>
</apply>
```

Sample Presentation

```
<mrow><mn>17</mn><mo>&#x2208;</mo><mi mathvariant="double-struck">P</mi></mrow>
```

$$17 \in \mathbb{P}$$

4.4.10.7 *Exponential e* `<exponentiale/>`

Class	constant-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	e

This symbol represents the base of the natural logarithm, approximately 2.718.

Content MathML

```
<apply><eq/>
  <apply><ln/><exponentiale/></apply>
  <cn>1</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mi>ln</mi><mo>&#x2061;</mo><mi>e</mi></mrow>
 <mo>=</mo>
 <mn>1</mn>
</mrow>
```

$$\ln e = 1$$

4.4.10.8 Imaginary i <imaginaryi/>

Class	constant-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	i

This symbol represents the mathematical constant which is the square root of -1, commonly written i

Content MathML

```
<apply><eq/>
  <apply><power/><imaginaryi/><cn>2</cn></apply>
  <cn>-1</cn>
</apply>
```

Sample Presentation

```
<mrow><msup><mi>i</mi><mn>2</mn></msup><mo>=</mo><mn>-1</mn></mrow>
```

$$i^2 = -1$$

4.4.10.9 Not A Number <notanumber/>

Class	constant-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	NaN

A symbol to convey the notion of not-a-number. The result of an ill-posed floating computation. See IEEE standard for floating point representations.

Content MathML

```
<apply><eq/>
  <apply><divide/><cn>0</cn><cn>0</cn></apply>
  <notanumber/>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mn>0</mn><mo>/</mo><mn>0</mn></mrow>
 <mo>=</mo>
 <mi>NaN</mi>
</mrow>
```

$$0/0 = \text{NaN}$$

4.4.10.10 True `<true/>`

Class	constant-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	true

This symbol represents the Boolean value true, i.e. the logical constant for truth.

Content MathML

```
<apply><eq/>
  <apply><or/>
    <true/>
     <ci type="boolean">P</ci>
  </apply>
  <true/>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mi>true</mi><mo>&#x2228;</mo><mi>P</mi></mrow>
 <mo>=</mo>
 <mi>true</mi>
</mrow>
```

$$\text{true} \vee P = \text{true}$$

4.4.10.11 False `<false/>`

Class	constant-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	false

This symbol represents the Boolean value false, i.e. the logical constant for falsehood.

Content MathML

```
<apply><eq/>
  <apply><and/>
    <false/>
    <ci type="boolean">P</ci>
  </apply>
  <false/>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mi>false</mi><mo>&#x2227;</mo><mi>P</mi></mrow>
 <mo>=</mo>
 <mi>false</mi>
</mrow>
```

$$\text{false} \wedge P = \text{false}$$

4.4.10.12 Empty Set <emptyset/>

Class	constant-set
Attributes	CommonAtt
Content	Empty
OM Symbols	emptyset, emptyset

This symbol is used to represent the empty set, that is the set which contains no members. It takes no parameters.

The `emptyset` element takes an optional attribute `type`. If its value is `"multiset"`, then the `emptyset` corresponds to the emptyset symbol from the multiset1 CD.

Content MathML

```
<apply><neq/>
  <integers/>
  <emptyset/>
</apply>
```

Sample Presentation

```
<mrow><mi mathvariant="double-struck">Z</mi><mo>&#x2260;</mo><mi>&#x2205;</mi></mrow>
```

$$\mathbb{Z} \neq \varnothing$$

4.4.10.13 pi <pi/>

Class	constant-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	pi

A symbol to convey the notion of pi, approximately 3.142. The ratio of the circumference of a circle to its diameter.

Content MathML

```
<apply><approx/>
  <pi/>
  <cn type="rational">22<sep/>7</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mi>&#x3c0;</mi>
 <mo>&#x2243;</mo>
 <mrow><mn>22</mn><mo>/</mo><mn>7</mn></mrow>
</mrow>
```

$$\pi \simeq 22/7$$

4.4.10.14 Euler gamma <eulergamma/>

Class	constant-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	gamma

A symbol to convey the notion of the gamma constant. It is the limit of 1 + 1/2 + 1/3 + ... + 1/m - ln m as m tends to infinity, this is approximately 0.5772.

Content MathML

```
<apply><approx/>
  <eulergamma/>
  <cn>0.5772156649</cn>
</apply>
```

Sample Presentation

```
<mrow><mi>&#x3b3;</mi><mo>&#x2243;</mo><mn>0.5772156649</mn></mrow>
```

$$\gamma \simeq 0.5772156649$$

4.4.10.15 infinity <infinity/>

Class	constant-arith
Attributes	CommonAtt
Content	Empty
OM Symbols	infinity

A symbol to represent the notion of infinity.

Content MathML

```
<infinity/>
```

Sample Presentation

```
<mi>&#x221e;</mi>
```

$$\infty$$

4.5 Deprecated Content Elements

4.5.1 Declare `<declare>`

Editor's note: MiKoThis should maybe be moved into a general section about changes or deprecated elements. Also Stan thinks the text should be improved.

MathML2 provided the `declare` element to bind properties like types to symbols and variables and to define abbreviations for structure sharing. This element is deprecated in MathML 3. Structure sharing can obtained via the `share` element (see Section 4.2.7 for details).

4.6 The Strict Content MathML Translation

This section sketches the meaning-giving translation from full MathML3 into Strict Content MathML as a list of transformation rules which are supposed to be applied in order.

1. *Normalize non-strict `bind`*: Change the outer `bind` tags in binding expressions to `apply`, if they have qualifiers or multiple children. This simplifies the algorithm by allowing the subsequent rules to be applied to non-strict binding expressions without case distinction. Note that the following rules will change the `apply` elements introduced in this step back to `bind` elements.
2. *Normalize Container Markup*:
   (a) Sets and lists are rewritten by the rule Section 4.3.4.2.
   (b) Interval, vectors, matricies, and matrix rows are rewritten as described in Section 4.4.1.1, Section 4.4.9.1, Section 4.4.9.2 and Section 4.4.9.3.
   (c) Lambda expressions are rewritten by rules Rewrite: lambda
   (d) Piecewise funtions are rewritten, as described in Section 4.4.1.9.
3. *Special Case Operator Rules*: This step deals with the special cases for the operators introduced in Section 4.4. There are different classes of special cases to be taken into account here:
   (a) *Quantifiers with `condition`:*: The two quantifiers `forall` and `exists` are rewritten to expressions using implication and conjunction by the rule Rewrite: quantifier.
   (b) Derivatives are rewritten with rules Rewrite: diff, Rewrite: diff, Rewrite: partialdiffdegree that take special care of explicating the binding status of the variables involved.
   (c) Integrals are rewritten with rules Rewrite: int that take special care of bound/free variables and of the orientation of the range of integration if it is given as a `lowlimit`/`uplimit` pair.
   (d) Limits are rewritten as described in Rewrite: tendsto and Rewrite: limits condition.
   (e) Sums and products are rewritten as described in Section 4.4.6.1 and Section 4.4.6.2.
   (f) Logarithms are rewritten as described in Section 4.4.7.4.
   (g) Moments are rewriteen as described in Section 4.4.8.6.
4. *Rewriting Qualifiers*: This rule is applied to `apply` with `bvar` children and normalizes various cases of qualifiers.
   (a) *Rewriting Intervals*: Qualifiers given as `interval` and `lowlimit`/`uplimit` are rewritten to intervals of integers via Rewrite: interval qualifier.
   (b) *Multiple `conditions`*: Multiple `condition` qualifiers are rewritten to one, by taking their conjunction, then this is rewritten to `domainofapplication` according to rule Rewrite: condition.
   (c) *Multiple `domainofapplication`* Multiple `domainofapplication` qualifiers are rewritten to one by taking the intersection of the specified domains.

5. *Eliminating* `domainofapplication`: At this stage, any `apply` has at most one `domainofapplication` child. As `domainofapplication` is not Strict Content MathML, it is rewritten
   (a) into an application of a restricted function via the rule Rewrite: restriction if the `apply` does not contain a `bvar` child.
   (b) into an application of predicate_on_list via the rules Rewrite: n-ary relations and Rewrite: n-ary relations bvar if used with a relation.
   (c) into a construction with the apply_to_list symbol via the general rule Rewrite: n-ary domainofapplication for general n-ary operators..
   (d) into a construction using the suchthat symbol from the set1 content dictionary in `apply` with bound variables via the Rewrite: apply bvar domainofappliction rule
6. *Rewriting* `cn`: Numbers represented as `cn` elements with `type` is one of `"e-notation"`, `"rational"`, `"complex-cartesian"`, `"complex-polar"`, `"constant"` are rewritten as strict `cn` via rules Rewrite: cn sep, Rewrite: cn constant.
7. *Rewriting the* `type` *attribute*: `ci` and `csymbol` elements with a `type` attribute to a strict expression with `semantics` according to rule Rewrite: ci type annotation.
8. *Token Elements containing Presentation MathML*: Any `ci`, `csymbol` or `sep` segments of `cn` containing presentation MathML rewritten to `semantics` elements with rule Rewrite: ci presentation mathml and its analog for `csymbol`.
9. *rewriting* `definitionURL` *and* `encoding` *on* `csymbol`: If the `definitionURL` and `encoding` attributes on a `csymbol` element can be interpreted as a reference to a content dictionary (see Section 4.2.3.2 for details), then this content dictionary referenced by the `cd` attribute instead.
10. *rewriting attributes*: Any element with attributes that are not allowed in strict markup is rewritten to a `semantics` construction with the element without these attributes as the first child and the attributes in `annotation` elements.

Rewrite: attributes

For instance,

```
<ci class="foo" xmlns:other="http://example.com" other:att="bla">x</ci>
```

is rewritten to

```
<semantics>
  <ci>x</ci>
  <annotation cd="mathmlattr"
     name="class" encoding="text">foo</annotation>
  <annotation-xml cd="mathmlattr" name="foreign" encoding="MathML">
    <apply><csymbol cd="mathmlattr">foreign_attribute</csymbol>
      <cs>http://example.com</cs>
      <cs>other</cs>
      <cs>att</cs>
      <cs>bla</cs>
    </apply>
  </annotation-xml>
</semantics>
```

For MathML attributes not allowed in Strict Content MathML the content dictionary mathmlattr is referenced, which provides symbols for all attributes allowed on content MathML elements. For other attributes in other namespaces, the namespace URI is encoded in the `definitionURL` attribute instead.

11. *rewriting operators*: Any remaining operator defined in Section 4.4 is rewritten to a `csymbol` referencing the symbol identified in the syntax table by the rule Rewrite: element.

Chapter 5

Mixing Markup Languages

The semantic annotation elements provide an important tool for making associations between alternate representations of an expression, as well as for associating semantic adornments and other attributions with a MathML expression. These elements allow presentation markup and content markup to be combined in several different ways. One method, known as *mixed markup*, is to intersperse content and presentation elements in what is essentially a single tree. Another method, known as *parallel markup*, is to provide *both* explicit presentation markup and content markup in a pair of markup expressions, combined by a single `semantics` element.

5.1 Semantic Annotations

An important concern of MathML is to represent associations between presentation and content markup forms for an expression, and of associations between MathML markup forms and other representations for an expression. An additional concern is the preservation of semantic attributions that are associated with MathML presentation or content forms. These associations are known collectively as *semantic annotations*. A semantic annotation decorates a MathML expression with a sequence of pairs made up of a symbol, known as the *annotation key*, and an associated entity, the *annotation value*.

5.1.1 Annotation elements

MathML uses the `semantics`, `annotation`, and `annotation-xml` elements to represent semantic annotations. The `semantics` element provides the container for an annotated element and a sequence of annotations, represented by `annotation` elements, for character data annotations, and by `annotation-xml` elements, for XML markup annotations, that represent the annotation key/value pairs.

```
<semantics>
  <mrow>
    <mrow>
      <mo>sin</mo>
      <mfenced><mi>x</mi></mfenced>
    </mrow>
    <mo>+</mo>
    <mn>5</mn>
  </mrow>
  <annotation cd="TeX" name="plainTeXrep" encoding="TeX">
    \sin x + 5
  </annotation>
```

```
  <annotation-xml cd="openmath" name="XMLencoding" encoding="OpenMath">
    <OMA xmlns="http://www.openmath.org/OpenMath">
      <OMS cd="arith1" name="plus"/>
      <OMA><OMS cd="transc1" name="sin"/><OMV name="x"/></OMA>
      <OMI>5</OMI>
    </OMA>
  </annotation-xml>
</semantics>
```

A semantic annotation may provide an alternate representation for a MathML expression, either as another MathML or XML expression, or as character data represented in some other markup language. An annotation may provide an equivalent representation that captures all of the relevant semantic behavior of the expression, or it may extend the object with additional semantic properties that change the expression in an essential way, or it may simply provide additional rendering or other associations that are incidental to the semantics of the expression.

The relationship between the expression to be annotated and the annotation value is identified by a symbol, known as the *annotation key*. The annotation key is the primary identifier that an application should use to determine if it understands the associated annotation value. If the annotation key is not specified, it defaults to a distinguished annotation key that specifies that the annotation provides an alternate representation for the annotated expression. In this case, an application should use the value of the `encoding` attribute to determine if it understands the alternate representation.

Each annotation element provides a reference to its annotation key via the `cd` and `name` attributes. Taken together, these attributes identify a named symbol from a specific content dictionary that describes the nature of the annotation. The referencing mechanism is the same as for the `csymbol` element (see Section 4.2.3) only that the symbol name of the key symbol is directly given by the `name` attribute. The `definitionURL` attribute provides an alternative way to reference the key symbol for an annotation, for better compatibility with MathML 2. If none of these attributes are specified, the annotation key is assumed to be the symbol alternate-representation from the mathmlkeys content dictionary.

The `semantics` element is considered to be both a presentation element and a content element, and may be used in either context. All MathML processors should process the `semantics` element, even if they only process one of these two subsets of MathML.

5.1.2 Annotation references

In the usual case, each annotation element includes either character data content (in the case of `annotation`) or XML markup data (in the case of `annotation-xml`) that represents the *annotation value*. There is no restriction on the type of annotation that may appear within a `semantics` element. For example, an annotation could provide a TEX encoding, a linear input form for a computer algebra system, a rendered image, or detailed mathematical type information.

In some cases the alternative children of a `semantics` element are not an essential part of the behavior of the annotated expression, but may be useful to specialized processors. To enable the availability of several annotation formats in a more efficient manner, a `semantics` element may contain empty `annotation` and `annotation-xml` elements that provide `encoding` and `href` attributes to specify an external location for the annotation value associated with the annotation. This type of annotation is known as an *annotation reference*.

```
<semantics>
  <mfrac><mi>a</mi><mrow><mi>a</mi><mo>+</mo><mi>b</mi></mrow></mfrac>
  <annotation encoding="image/png" href="333/formula56.png"/>
```

```
  <annotation encoding="text/maple" href="333/formula56.ms"/>
</semantics>
```

Processing agents that anticipate that consumers of exported markup may not be able to retrieve the external entity referenced by such annotations should request the content of the external entity at the indicated location and replace the annotation with its expanded form.

An annotation reference follows the same rules as for other annotations to determine the annotation key that specifies the relationship between the annotated object and the annotation value.

5.1.3 Alternate representations

A semantic annotation may provide an alternate representation for a MathML expression. For example, in the MathML representation below, the `semantics` element binds together various representations of the sum of the sine function applied to a variable *x* and the number 5.

```
<semantics>
  <mrow>
    <mrow>
      <mo>sin</mo>
      <mfenced open="(" close=")"><mi>x</mi></mfenced>
    </mrow>
    <mo>+</mo>
    <mn>5</mn>
  </mrow>
  <annotation-xml cd="mathml" name="contentequiv" encoding="MathML&amp;#160;Content">
    <apply>
      <csymbol cd="algebra-logic" name="plus"/>
      <apply><sin/><ci>x</ci></apply>
      <cn>5</cn>
    </apply>
  </annotation-xml>
  <annotation cd="maple" name="nativerep" encoding="text/maple">sin(x) + 5</annotation>
  <annotation cd="mathematica" name="nativerep" encoding="Mathematica">Sin[x] + 5</annotati
  <annotation cd="TeX" name="plainTeXrep" encoding="TeX"> \sin x + 5</annotation>
  <annotation-xml cd="openmath" name="XMLencoding" encoding="OpenMath">
    <OMA xmlns="http://www.openmath.org/OpenMath">
      <OMA>
        <OMS cd="arith1" name="plus"/>
        <OMA><OMS cd="transc1" name="sin"/><OMV name="x"/></OMA>
        <OMI>5</OMI>
      </OMA>
    </OMA>
  </annotation-xml>
</semantics>
```

Here, the presentation element in the first child of the `semantics` element is annotated with various content-oriented representations. Each `annotation` and `annotation-xml` element specifies the nature of the annotation by referencing a key symbol in an appropriate content dictionary. For instance, the first `annotation-xml` element references the key symbol `"contentequiv"` from the `attribution-keys` content dictionary that specifies that the content MathML expression it provides is mathematically equivalent to the annotated presentation MathML expression.

5.1.4 Flattening semantic annotations

One consequence of the syntax for semantic annotation is that annotations may be applied to markup elements that are themselves annotations of other elements. In other words, a `semantics` element may contain another `semantics` element as its first child element, as in the sketch below:

```
<semantics>
  <semantics>A A_1 A_k</semantics>
  A_k+1 ... A_n
</semantics>
```

where the A_i represent `annotation` or `annotation-xml` elements. This expression is equivalent to a single `semantics` element that contains the union of the annotations from the original `semantics` elements.

```
<semantics>
  A
  A_1 ... A_n
</semantics>
```

The operation that produces an expression with a single layer of semantic annotations is called *flattening*. Multiple annotations with the same key symbol are allowed. While the order of the given attributes does not imply any notion of priority, it potentially could be significant.

5.2 Elements for Semantic Annotations

This section explains the semantic mapping elements `semantics`, `annotation`, and `annotation-xml`. These elements associate alternate representations for a presentation or content expression, or associate semantic or other attributions that may modify the meaning of the annotated expression.

5.2.1 The `semantics` element

The `semantics` element is the container element that associates annotations with a MathML expression. The `semantics` element has as its first child the expression to be annotated. Subsequent children provide the annotations.

An annotation whose representation is XML based is enclosed in an `annotation-xml` element. An annotation whose representation is text is enclosed in an `annotation` element.

The `semantics` element takes the `definitionURL` and `encoding` attributes, which reference an external source for some or all of the semantic information for the annotated element, as modified by the annotation. The use of these attributes are deprecated in MathML3. The `definitionURL` attribute should be added to the elements whose meaning is to be clarified.

Attributes of the `semantics` element

Name	values	default
definitionURL	a URI pointing to an equivalent formulation	
encoding	the encoding of that equivalent formulation	

```
<semantics>
  <mrow>
    <mrow>
      <mo>sin</mo>
      <mfenced><mi>x</mi></mfenced>
```

```
    </mrow>
    <mo>+</mo>
    <mn>5</mn>
  </mrow>
  <annotation-xml cd="mathml" name="contentequiv" encoding="MathML Content">
    <apply>
      <plus/>
      <apply><sin/><ci>x</ci></apply>
      <cn>5</cn>
    </apply>
  </annotation-xml>
  <annotation cd="maple" name="nativerep" encoding="Maple">
    sin(x) + 5
  </annotation>
  <annotation cd="mathematica" name="nativerep" encoding="Mathematica">
    Sin[x] + 5
  </annotation>
  <annotation cd="TeX" name="plainTeXrep" encoding="TeX">
    \sin x + 5
  </annotation>
  <annotation-xml cd="openmath" name="XMLencoding" encoding="OpenMath">
    <OMA xmlns="http://www.openmath.org/OpenMath">
      <OMS cd="arith1" name="plus"/>
      <OMA><OMS cd="transc1" name="sin"/><OMV name="x"/></OMA>
      <OMI>5</OMI>
    </OMA>
  </annotation-xml>
</semantics>
```

The default rendering of a `semantics` element is the default rendering of its first child. A renderer may use the information contained in the annotations to customize its rendering of the annotated element.

5.2.2 The `annotation` element

The `annotation` element is the container element for a semantic annotation whose representation is parsed character data in a non-XML format. The `annotation` element should contain the character data for the annotation, and should not contain XML markup elements. If the annotation contains one of the XML reserved characters `&`, `<`, `>`, `'`, or `"`, then these characters must be encoded using an XML entity reference or an XML `CDATA` section.

The `annotation` element takes the attributes `cd`, and `name`. Taken together, these attributes reference the key symbol that identifies the relation between the annotated element and the annotation.

The `annotation` element takes the `definitionURL` attribute, which provides an alternative way to reference the key symbol that identifies the relation between the annotated element and the annotation.

If none of these attributes are specified, the key symbol for the annotation is the symbol `alternate-representation` from the `attribution-keys` content dictionary.

The `annotation` element takes the `encoding` attribute, which describes the content type of the annotation. The value of the `encoding` attribute may contain a MIME type that identifies the data format for the encoding data. For data formats that do not have an associated MIME type, implementors may

choose a For data formats that do not have an associated media type, implementors may choose a self-describing character string to identify their content type.

The `annotation` element allows the `href` attribute, which provides a mechanism to attach external entities as annotations on MathML expressions.

```
<annotation cd="TeX" name="plainTeXrep" encoding="TeX">
  \sin x + 5
</annotation>

<annotation encoding="image/png" href="333/formula56.png"/>
```

The `annotation` element is a semantic mapping element that may only be used as a child of the `semantics` element. While there is no default rendering for the `annotation` element, a renderer may use the information contained in an annotation to customize its rendering of the annotated element.

Attributes of the `annotation` and `annotation-xml` elements

Name	values	default
definitionURL	a URI pointing to the meaning of the annotation relationship	
encoding	an encoding name of the alternate representation contained in the annotation	
cd	the content-dictionary name of the symbol denoting the annotation relationship	attribution-keys
name	the name of the equivalent symbol	alternate-representation
href	the (relative) URL to the content of the annotation	
clipboardflavor	the (standardized or platform specific) flavor name indicating that this annotation should provide a clipboard flavor, see Section 6.3.2	

5.2.3 The `annotation-xml` element

The `annotation-xml` element is the container element for a semantic annotation whose representation is structured markup in an XML format. The `annotation-xml` element should contain the markup elements, attributes, and character data for the annotation.

The `annotation-xml` element takes the attributes `cd`, and `name`. Taken together, these attributes reference the key symbol that identifies the relation between the annotated element and the annotation.

The `annotation-xml` element takes the `definitionURL` attribute, which provides an alternative way to reference the key symbol that identifies the relation between the annotated element and the element should contain the markup elements, attributes, and character data for the annotation.

The `annotation-xml` uses the same attributes as the `annotation` element for identifying the key symbol of the annotation. It uses the `encoding` attribute, which describes the content type of the annotation. The value of the `encoding` attribute may contain a media type that identifies the data format for the encoding data. For data formats that do not have an associated media type, implementors may choose a self-describing character string to identify their content type. For example, Section 6.2.3 identifies the strings `MathML`, `MathML Presentation`, and `MathML Content` as predefined values for the `encoding` attribute that may be used to identify MathML markup in an `annotation-xml` element. Finally the `annotation-xml` element allows the `href` attribute, which provides a mechanism to attach external XML entities as annotations on MathML expressions.

```
<annotation-xml cd="mathmlkeys" name="contentequiv" encoding="MathML Content">
  <apply>
```

```
    <plus/>
    <apply><sin/><ci>x</ci></apply>
    <cn>5</cn>
  </apply>
</annotation-xml>

<annotation-xml cd="openmath" name="XMLencoding" encoding="OpenMath">
  <OMA xmlns="http://www.openmath.org/OpenMath">
    <OMS cd="arith1" name="plus"/>
    <OMA><OMS cd="transc1" name="sin"/><OMV name="x"/></OMA>
    <OMI>5</OMI>
  </OMA>
</annotation-xml>
```

When the annotation value is represented in an XML dialect other than MathML, the namespace for the XML markup for the annotation should be identified by means of namespace attributes and/or namespace prefixes on the annotation value. For instance:

```
<annotation-xml encoding="application/xhtml+xml">
  <html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>E</title></head>
    <body><p>The base of the natural logarithms, approximately 2.71828.</p></body>
  </html>
</annotation-xml>
```

The `annotation-xml` element is a semantic mapping element that may only be used as a child of the `semantics` element. While there is no default rendering for the `annotation-xml` element, a renderer may use the information contained in an annotation to customize its rendering of the annotated element.

5.3 Combining Presentation and Content Markup

Presentation markup encodes the *notational structure* of an expression. Content markup encodes the *functional structure* of an expression. In certain cases, a particular application of MathML may require a combination of both presentation and content markup. This section describes specific constraints that govern the use of presentation markup within content markup, and vice versa.

5.3.1 Presentation Markup in Content Markup

Presentation markup may be embedded within content markup so long as the resulting expression retains an unambiguous function application structure. Specifically, presentation markup may only appear in content markup in three ways:

1. within `ci` and `cn` token elements
2. within the `csymbol` element
3. within the `semantics` element

Any other presentation markup occurring within content markup is a MathML error. More detailed discussion of these three cases follows:

Presentation markup within token elements. The token elements `ci` and `cn` are permitted to contain any sequence of MathML characters (defined in Chapter 7) and/or presentation elements. Contiguous blocks of MathML characters in `ci` or `cn` elements are treated as if wrapped in `mi` or `mn` elements, as appropriate, and the resulting collection of presentation elements is rendered as if wrapped in an implicit `mrow` element.

Presentation markup within the `csymbol` element. The `csymbol` element may contain either MathML characters interspersed with presentation markup, or content markup. It is a MathML error for a `csymbol` element to contain both presentation and content elements. When the `csymbol` element contains character data and presentation markup, the same rendering rules that apply to the token elements `ci` and `cn` should be used.

Presentation markup within the `semantics` element. One of the main purposes of the `semantics` element is to provide a mechanism for incorporating arbitrary MathML expressions into content markup in a semantically meaningful way. In particular, any valid presentation expression can be embedded in a content expression by placing it as the first child of a `semantics` element. The meaning of this wrapped expression should be indicated by one or more annotation elements also contained in the `semantics` element.

5.3.2 Content Markup in Presentation Markup

Content markup may be embedded within presentation markup so long as the resulting expression has an unambiguous rendering. That is, it must be possible, in principle, to produce a presentation markup fragment for each content markup fragment that appears in the combined expression. The replacement of each content markup fragment by its corresponding presentation markup should produce a well-formed presentation markup expression. A presentation engine should then be able to process this presentation expression without reference to the content markup bits included in the original expression.

In general, this constraint means that each embedded content expression must be well-formed, as a content expression, and must be able to stand alone outside the context of any containing content markup element. As a result, the following content elements may not appear as an immediate child of a presentation element: `annotation`, `annotation-xml`, `bvar`, `condition`, `degree`, `logbase`, `lowlimit`, `uplimit`.

In addition, within presentation markup, content markup may not appear within presentation token elements.

5.4 Parallel Markup

Some applications are able to use *both* presentation and content information. *Parallel markup* is a way to combine two or more markup trees for the same mathematical expression. Parallel markup is achieved with the `semantics` element. Parallel markup for an expression may appear on its own, or as part of a larger content or presentation tree.

5.4.1 Top-level Parallel Markup

In many cases, the goal is to provide presentation markup and content markup for a mathematical expression as a whole. A single `semantics` element may be used to pair two markup trees, where one child element provides the presentation markup, and the other child element provides the content markup.

The following example encodes the Boolean arithmetic expression $(a+b)(c+d)$ in this way.

```
<semantics>
  <mrow>
    <mrow><mo>(</mo><mi>a</mi> <mo>+</mo> <mi>b</mi><mo>)</mo></mrow>
    <mo>&InvisibleTimes;</mo>
    <mrow><mo>(</mo><mi>c</mi> <mo>+</mo> <mi>d</mi><mo>)</mo></mrow>
```

```
  </mrow>
  <annotation-xml encoding="MathML Content">
    <apply><and/>
      <apply><xor/><ci>a</ci> <ci>b</ci></apply>
      <apply><xor/><ci>c</ci> <ci>d</ci></apply>
    </apply>
  </annotation-xml>
</semantics>
```

Note that the above markup annotates the presentation markup as the first child element, with the content markup as part of the `annotation-xml` element. An equivalent form could be given that annotates the content markup as the first child element, with the presentation markup as part of the `annotation-xml` element.

5.4.2 Parallel Markup via Cross-References

To accommodate applications that must process sub-expressions of large objects, MathML supports cross-references between the branches of a `semantics` element to identify corresponding sub-structures. These cross-references are established by the use of the `id` and `xref` attributes within a `semantics` element. This application of the `id` and `xref` attributes within a `semantics` element should be viewed as best practice to enable a recipient to select arbitrary sub-expressions in each alternative branch of a `semantics` element. The `id` and `xref` attributes may be placed on MathML elements of any type.

The `id` and `xref` attributes are supported by MathML to provide cross-references for those applications that do not otherwise require the use of namespaces or validation. Those applications that support namespaces may use the `xml:id` attribute in the same manner as is described for the `id` attribute. Similarly, those applications that support validation may use other attributes declared of type ID and IDREF to establish cross-references between corresponding sub-expressions. Of course, cross-references that use custom attributes in this way rely on prior agreement between the producing and consuming applications to preserve the cross-references.

The following example demonstrates cross-references for the Boolean arithmetic expression $(a+b)(c+d)$.

```
<semantics>
  <mrow id="E">
    <mrow id="E.1">
      <mo id="E.1.1">(</mo>
      <mi id="E.1.2">a</mi>
      <mo id="E.1.3">+</mo>
      <mi id="E.1.4">b</mi>
      <mo id="E.1.5">)</mo>
    </mrow>
    <mo id="E.2">&InvisibleTimes;</mo>
    <mrow id="E.3">
      <mo id="E.3.1">(</mo>
      <mi id="E.3.2">c</mi>
      <mo id="E.3.3">+</mo>
      <mi id="E.3.4">d</mi>
      <mo id="E.3.5">)</mo>
    </mrow>
  </mrow>
```

```
  <annotation-xml encoding="MathML Content">
    <apply xref="E">
      <and xref="E.2"/>
      <apply xref="E.1">
        <xor xref="E.1.3"/><ci xref="E.1.2">a</ci><ci xref="E.1.4">b</ci>
      </apply>
      <apply xref="E.3">
        <xor xref="E.3.3"/><ci xref="E.3.2">c</ci><ci xref="E.3.4">d</ci>
      </apply>
    </apply>
  </annotation-xml>
</semantics>
```

An `id` attribute and associated `xref` attributes that appear within the same `semantics` element establish the cross-references between corresponding sub-expressions.

All of the `id` attributes referenced by any `xref` attribute must be in the *same* branch of an enclosing `semantics` element. This constraint guarantees that the cross-references do not create unintentional cycles. This restriction does *not* exclude the use of `id` attributes within other branches of the enclosing `semantics` element. It does, however, exclude references to these other `id` attributes originating from the same `semantics` element.

There is no restriction on which branch of the `semantics` element may contain the destination `id` attributes. It is up to the application to determine which branch to use.

In general, there will not be a one-to-one correspondence between nodes in parallel branches. For example, a presentation tree may contain elements, such as parentheses, that have no correspondents in the content tree. It is therefore often useful to put the `id` attributes on the branch with the finest-grained node structure. Then all of the other branches will have `xref` attributes to some subset of the `id` attributes.

In absence of other criteria, the first branch of the `semantics` element is a sensible choice to contain the `id` attributes. Applications that add or remove annotations will then not have to re-assign these attributes as the annotations change.

In general, the use of `id` and `xref` attributes allows a full correspondence between sub-expressions to be given in text that is at most a constant factor larger than the original. The direction of the references should not be taken to imply that sub-expression selection is intended to be permitted only on one child of the `semantics` element. It is equally feasible to select a subtree in any branch and to recover the corresponding subtrees of the other branches.

Parallel markup with cross-references may be used in any XML-encoded branch of the semantic annotations, as shown by the following example where the Boolean expression of the previous section is annotated with OpenMath markup that includes cross-references:

```
<semantics>
  <mrow id="E">
    <mrow id="E.1">
      <mo id="E.1.1">(</mo>
      <mi id="E.1.2">a</mi>
      <mo id="E.1.3">+</mo>
      <mi id="E.1.4">b</mi>
      <mo id="E.1.5">)</mo>
    </mrow>
```

```
    <mo id="E.2">&InvisibleTimes;</mo>
    <mrow id="E.3">
      <mo id="E.3.1">(</mo>
      <mi id="E.3.2">c</mi>
      <mo id="E.3.3">+</mo>
      <mi id="E.3.4">d</mi>
      <mo id="E.3.5">)</mo>
    </mrow>
  </mrow>

  <annotation-xml encoding="MathML Content">
    <apply xref="E">
      <and xref="E.2"/>
      <apply xref="E.1">
        <xor xref="E.1.3"/><ci xref="E.1.2">a</ci><ci xref="E.1.4">b</ci>
      </apply>
      <apply xref="E.3">
        <xor xref="E.3.3"/><ci xref="E.3.2">c</ci><ci xref="E.3.4">d</ci>
      </apply>
    </apply>
  </annotation-xml>

  <annotation-xml encoding="OpenMath"
                  xmlns:om="http://www.openmath.org/OpenMath">

    <om:OMA href="E">
      <om:OMS name="and" cd="logic1" href="E.2"/>

      <om:OMA href="E.1">
        <om:OMS name="xor" cd="logic1" href="E.1.3"/>
        <om:OMV name="a" href="E.1.2"/>
        <om:OMV name="b" href="E.1.4"/>
      </om:OMA>

      <om:OMA href="E.3">
        <om:OMS name="xor" cd="logic1" href="E.3.3"/>
        <om:OMV name="c" href="E.3.2"/>
        <om:OMV name="d" href="E.3.4"/>
      </om:OMA>
    </om:OMA>
  </annotation-xml>
</semantics>
```

Here `OMA`, `OMS` and `OMV` are elements defined in the OpenMath standard for representing application, symbol, and variable, respectively. The references from the OpenMath annotation are given by the `href` attributes.

Chapter 6

Interactions of MathML with the Host Environment

6.1 Introduction

To be effective, MathML must work well with a wide variety of renderers, processors, translators and editors. This chapter raises some of the interface issues involved in generating and rendering MathML. Since MathML exists primarily to encode mathematics in Web documents, perhaps the most important interface issues are related to embedding MathML in [HTML4] and [XHTML], and in any newer HTML when it appears.

There are three kinds of interface issues that arise in embedding MathML in other XML documents. First, MathML must be semantically integrated. MathML markup must be recognized as valid embedded XML content, and not as an error. This could be seen as primarily a question of managing namespaces in XML [Namespaces]. However, the implementation of XML namespaces and their management has not been well supported by recent commercial software. So there have grown up other ways of dealing with 'foreign content' in an XML document which is viewed as of a particular type.

Second, in the case of HTML/XHTML, MathML rendering must be integrated with browser software. Some browsers already implement MathML rendering natively, and one can expect more browsers will do so in the future. At the same time, other browsers have developed infrastructure to facilitate the rendering of MathML and other embedded XML content by third-party software or other built-in technology. Examples of this built-in technology are the sophisticated CSS rendering engines now available, and the powerful implementations of ECMAscript (or JavaScript) that are becoming common. Using these browser-specific mechanisms generally requires additional interface markup of some sort to activate them. In the case of CSS, there is a special restricted form of MathML3 tailored for use with present-day CSS, up to CSS2.1, which is specified in "A MathML for CSS profile" [MathMLforCSS]. This does not offer the full expressiveness afforded by MathML3 but provides a portable simpler form that can be rendered acceptably on the screen by modern CSS engines.

Third, other tools for generating and processing MathML must be able to communicate. A number of MathML tools have been or are being developed, including editors, translators, computer algebra systems, and other scientific software. However, since MathML expressions tend to be lengthy, and prone to error when entered by hand, special emphasis must be given to ensuring that MathML can be easily generated by user-friendly conversion and authoring tools, and that these tools work together in a dependable, platform and vendor independent way.

Issue (names-without-dashes): So as to conclude the thread atClipboard section implementation?, we need to correct all occurrences of annotation and annotation-xml elements to replace MathML-Content by MathML Content.This is now achieved with the space being unbreakable instead of the plain space.

This chapter applies to both content and presentation MathML and indicates a particular processing model to the `semantics`, `annotation` and `annotation-xml` elements defined in Section 5.1.

6.2 Invoking MathML Processors: namespace, extensions, and mime-types

6.2.1 Recognizing MathML in an XML Model

Within an XML document supporting namespaces [XML], [Namespaces], the preferred method to recognize MathML markup is by the identification of the `math` element in the appropriate namespace, i.e. that of URI `http://www.w3.org/1998/Math/MathML`.

This is the recommended method to embed MathML within [XHTML] documents. Some user-agents' setup may require supplementary information to be available.

Markup-language specifications that wish to embed MathML may provide special conditions or recognizing MathML independently of this recommendation. The conditions should be similar to those expressed in this recommendation and the elements' local-names should remain the same.

6.2.2 Resource Types for MathML Documents

Although rendering MathML expressions often occurs in place in a Web browser, other MathML processing functions take place more naturally in other applications. Particularly common tasks include opening a MathML expression in an equation editor or computer algebra system. It is important therefore to specify the encoding names with which MathML fragments should be called.

Outside of the conditions of an XML environment where namespaces are recognized, media types [RFC2045], [RFC2046] should be used if possible to to ensure the invocation of a MathML processor. In the cases where media types are not usable, for example in some clipboard environments, the names described in the next section should be used.

6.2.3 Names of MathML Encodings

Issue (types-for-p-and-c): Different media-types for generic, presentation, and content MathML are wished by some group members but not others. The current section describes the alternative where a media-type for generic, presentation, and content MathML exist.

MathML contains two distinct vocabularies: one for encoding mathematical semantics defined in Chapter 4 and one for encoding visual presentation defined in Chapter 3. Some MathML-aware applications import and export only one of these vocabularies, while other may be capable of producing and consuming both. The following three encoding names should be used to distinguish between content and presentation MathML when needed.

- *MathML Presentation*: Instance contains presentation MathML markup only
  - Windows Flavor Name: `MathML Presentation`
  - Media Type: `application/mathml-presentation+xml`
  - Universal Type Identifier: `public.mathml.presentation`
- *MathML Content*: Instance contains content MathML markup only
  - Windows Flavor Name: `MathML Content`
  - Media Type:`application/mathml-content+xml`
  - Universal Type Identifier: `public.mathml.content`
- *MathML (generic)*: Any well-formed MathML instance presentation markup, content markup, or a mixture of the two is allowed
  - Windows Flavor Name:`MathML`
  - Media Type: `application/mathml+xml`
  - Filename extension: `.mml`
  - Universal Type Identifier: `public.mathml`

See Appendix B for more details about each of the encoding names.

Supplementary to these names, the clipboard flavor names `MathML-Content`, and `MathML-Presentation` have been used and are now deprecated; moreover, in MathML 1.0, the media-type `text/mathml` was suggested, this has been superceded by [RFC3023].

6.3 Transferring MathML

MathML expressions are often exchanged between applications using the familiar copy-and-paste or drag-and-drop paradigms and sometimes are stored in files or exchanged over the HTTP protocol. This section provides recommended ways to process MathML while applying these transfers.

The transfers of MathML fragments are between the contexts of two applications by making them available in several flavors, often called *media types*, *clipboard formats* or *data flavors*. These flavors are ordered by preference. The copy-and-paste paradigm lets applications *place* content in a central *clipboard*, one data-stream per *clipboard format*; consuming applications negotiate by choosing to read the data of the format they elect. The drag-and-drop paradigm lets application offer content by declaring the available formats and potential recipients accept or reject a drop based on this list; the drop action then lets the receiving application request the delivery of the format in the indicated format. HTTP GET transfers, as in [HTTP11], let clients submit a list of accepted media-types and the server deliver the appropriate data indicating the chosen media-type. HTTP POST transfers, as in [HTTP11], let client submit the data with the accepted media-type.

Current desktop platforms offer copy-and-paste and drag-and-drop using similar transfer architectures with varying naming depending on the platform. HTTP transfers are all based on media-types. In this section we specify what applications should provide as transfer types, how they should be named, and how they should handle the special `semantics`, `annotation`, and `annotation-xml` elements.

To summarize the three negotiation mechanisms, we shall, here, be talking of *flavors*, each having a *name* (a character string) and a *content* (a stream of binary data), which are *offered*, *accepted*, *exported*.

6.3.1 Basic Transfer Flavors' Names and Contents

Note that the names indicated in Section 6.2.3 are the exact strings that should be used to describe the flavors corresponding to the encodings. On operating systems that allow such, applications should register such names (e.g. a call to Windows' RegisterClipboardFormat on launch or, on the Macintosh platform, the declaration of the support for the universal type in the application descriptor).

When transferring MathML, for example when placing it within a clipboard, an application MUST ensure the content is a well-formed XML instance of a MathML schema. Specifically:

1. The instance MUST begin with a XML declaration, e.g. <?xml version="1.0">
2. The instance MUST contain exactly one root `math` element.
3. Since MathML is frequently embedded within other XML document types, the instance MUST declare the MathML namespace on the root `math` element. In addition, the instance MAY use a `schemaLocation` attribute on the `math` element to indicate the location of MathML schema documents against which the instance is valid. Note that the presence of the `schemaLocation` attribute does not require a consumer of the MathML instance to obtain or use the cited schema documents.
4. The instance MUST use numeric character references (e.g. α) rather than character entity names (e.g. α) for greater interoperability.

5. The character encoding for the instance MUST be either specified in the XML header, UTF-16, or UTF-8. UTF-16-encoded data MUST begin with a byte-order mark (BOM). If no BOM or encoding is given, the character encoding will be assumed to be UTF-8.

6.3.2 Recommended Behaviors when Transferring

Applications that transfer MathML SHOULD adhere to the following conventions:

1. Applications that have pure presentation markup and/or pure content markup versions of an expression SHOULD offer as many of these two flavors as are available.
2. Applications that only export one MathML flavor should name it `MathML` if they don't know its nature. If they do, they should offer both the generic and specific transfer types but should only deliver the specific types if they know the recipient will support it. For the HTTP GET transfer, this means that the specific transfer types (presentation, content) should only be returned if the `Accept` HTTP header includes it. Applications that export the two specific transfer types should export the content and presentation transfer types as well as the generic flavor which combines the two others using a top-level MathML's `semantics` element (see Section 5.4.1).
3. When an application exports a MathML fragment whose only child of the root element is a `semantics` element, it SHOULD offer, after the flavors above in the preference order, a flavor for each `annotation` or `annotation-xml` element that has a `clipboardflavor` attribute: the flavor name should be given by the `clipboardflavor` attribute value of the `annotation` or `annotation-xml` element, and the content should be the child text in the surrounding encoding (if the `annotation` element contains only textual data), a valid XML fragment (if the `annotation-xml` element contains children), or the data resulting of requesting the URL given by the `href` attribute.User-agents implementors should be aware that the fragments contained in the clipboard could be *used* in a insecure way: indeed, opening a fragment in a receiving application will execute the programmes it contains, for example embedded scripts, and this without the traditional security restrictions of web-content. User-agent implementors should, thus, only allow transfer of *safe* content flavors, maybe resorting to a *sanitization* process.
4. As a final fallback applications MAY export a version of the data in plain-text flavor (such as `text/plain`, `CF_UNICODETEXT`, `UnicodeText`, `NSStringPboardType`, ...). When an application has multiple versions of an expression available, it may choose the version to export as text at its discretion. Since some older MathML-aware programs expect MathML instances transferred as text to begin with a `math` element, the text version should generally omit the XML declaration, DOCTYPE declaration and other XML prolog material before the `math` element. Similarly, the BOM should be omitted for Unicode text encoded as UTF-16. Note, the Unicode text version of the data should always be the last flavor exported, following the principle that exported flavors should be ordered with the most specific flavor first and the least specific flavor last.

6.3.3 Discussion

For purposes of determining whether a MathML instance is pure content markup or pure presentation markup, the `math` element and the `semantics`, `annotation` and `annotation-xml` elements should be regarded as belonging to both the presentation and content markup vocabularies. This is obvious for the root `math` element which is required for all MathML expressions. However, the `semantics` element and its child annotation elements comprise an arbitrary annotation mechanism within MathML, and are not tied to either presentation or content markup. Consequently, applications consuming MathML

should always process these four elements even if the application only implements one of the two vocabularies.

It is worth noting that the above recommendations allow agents producing MathML to provide binary data for the clipboard, for example as an image or an application-specific format. The sole method to do so is to reference the binary data by the `href` attribute since XML character data does not allow arbitrary byte-streams.

While the above recommendations are intended to improve interoperability between MathML-aware applications utilizing the transfer features, it should be noted that they do not guarantee interoperablility. For example, references to external resources (e.g. stylesheets, etc.) in MathML data can also cause interoperability problems if the consumer of the data is unable to locate them, just as can happen when cutting and pasting HTML or many other data types. Applications that make use of references to external resources are encouraged to make users aware of potential problems and provide alternate ways for obtaining the referenced resources. In general, consumers of MathML data containing references they cannot resolve or do not understand should ignore them.

6.3.4 Examples

6.3.4.1 Example 1

An e-Learning application has a database of quiz questions, some of which contain MathML. The MathML comes from multiple sources, and the e-Learning application merely passes the data on for display, but does not have sophisticated MathML analysis capabilities. Consequently, the application is not aware whether a given MathML instance is pure presentation or pure content markup, nor does it know whether the instance is valid with respect to a particular version of the MathML schema. It therefore places the following data formats on the clipboard:

Flavor Name	Flavor Content
`MathML`	`<?xml version="1.0"?>` `<math xmlns="http://www.w3.org/1998/Math/MathML">...</math>`
`Unicode Text`	`<math xmlns="http://www.w3.org/1998/Math/MathML">...</math>`

6.3.4.2 Example 2

An equation editor is able to generate pure presentation markup, valid with respect to MathML 3. Consequently, it exports the following flavors:

Flavor Name	Flavor Content
`MathML Presentation`	`<?xml version="1.0"?>` `<math xmlns="http://www.w3.org/1998/Math/MathML">...</math>`
`Tiff`	(a rendering sample)
`Unicode Text`	`<math xmlns="http://www.w3.org/1998/Math/MathML">...</math>`

6.3.4.3 Example 3

A schema-based content management system contains multiple MathML representations of a collection of mathematical expressions, including mixed markup from authors, pure content markup for interfacing to symbolic computation engines, and pure presentation markup for print publication. Due to the system's use of schemas, markup is stored with a namespace prefix. The system therefore can transfer the following data:

Flavor Name	Flavor Content

MathML Presentation	`<?xml version="1.0"?>`
	`<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd">`
	`<mml:mrow>`
	`...`
	`<mml:mrow>`
	`</mml:math>`

MathML Content	`<?xml version="1.0"?>`
	`<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd">`
	`<mml:apply>`
	`...`
	`<mml:apply>`
	`</mml:math>`

MathML	`<?xml version="1.0"?>`
	`<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd">`
	`<mml:mrow>`
	`<mml:apply> ... content markup within presentation markup ... </mml:apply>`
	`...`
	`</mml:mrow>`
	`</mml:math>`

TeX	`x \over x-1`

Unicode Text	`<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd">`
	`<mml:mrow>`
	`...`
	`<mml:mrow>`
	`</mml:math>`

6.3.4.4 *Example 4*

A similar content management system is web-based and delivers MathML representations of mathematical expressions. The system is able to produce presentation MathML, content MathML, TeX and pictures in PNG format. In web-pages being browsed, it could produce a MathML fragment such as the following:

```
<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
  <mml:semantics>
    <mml:mrow>...</mml:mrow>
    <mml:annotation-xml encoding="MathML Content">...</mml:annotation-xml>
    <mml:annotation clipboardflavor="TeX">{1 \over x}</mml:annotation>
    <mml:annotation clipboardflavor="image/png" href="formula3848.png"/>
  </mml:semantics>
</mml:math>
```

A web-browser that receives such a fragment and tries to export it as part of a drag-and-drop action, can offer the following flavors:

Flavor Name	Flavor Content

MathML Presentation	`<?xml version="1.0"?>`
	`<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd">`
	`<mml:mrow>`
	`...`
	`<mml:mrow>`
	`</mml:math>`

MathML Content	`<?xml version="1.0"?>`
	`<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd">`
	`<mml:apply>`
	`...`
	`<mml:apply>`
	`</mml:math>`

MathML	`<?xml version="1.0"?>`
	`<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd">`
	`<mml:mrow>`
	`<mml:apply> ... content markup within presentation markup ... </mml:apply>`
	`...`
	`</mml:mrow>`
	`</mml:math>`

TeX	`x \over x-1`
image/png	(the content of the picture file, requested from formula3848.png

Unicode Text	`<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd">`
	`<mml:mrow>`
	`...`
	`<mml:mrow>`
	`</mml:math>`

6.4 Combining MathML and Other Formats

MathML is usually used in combination with other markup languages. The most typical case is perhaps the use of MathML within a document-level markup language, such as XHTML or DocBook. It is also common that other object-level markup languages are also included in a compound document format, such as the W3C XHTML+MathML+SVG profile. Other common use cases include mixing other markup within MathML. For example, an authoring tool might insert an element representing a cursor position or other state information within MathML markup, so that an author can pick up editing where he or she left off.

Most document markup languages have some concept of an inline equation, (or graphic, object, etc.) so there is a typically a natural and well-defined way to incorporate MathML instances into the content model. However, in the other direction, embedding of markup within MathML is not so clear cut, since in many MathML elements, the role of child elements is defined by positionally. For example, the first child of an `apply` must be an operator, and the second child of an `mfrac` is the denominator. The proper behavior when foreign markup appears in such contexts is problematic. Even when such behavior can be defined in a particular context, it presents an implementation challenge for generic MathML processors.

For this reason, by default, the MathML schema does not allow foreign markup elements to be included within MathML instances. However, a lax schema is also provided for use in specific compound document formats where the behavior of mixed markup can be well-defined, and where the implemen-

tation of specialized user agents can be justified. Note that this is not uncommon, particularly when the allowed mixing is minimal.

In the default schema profile, no element from other namespaces are allowed, but attributes from other namespaces are allowed. MathML-processors that encounter unknown XML markup should behave as follows:

1. Attributes from non-MmathML namespaces should be silently ignored.
2. An element from a non-MathML namespaces should be treated as an error. If the element is a child of a presentation element, should be handled as described in Section 3.3.5. If the element is a child of a content element should be handled as described in Section 4.2.9.

For example, if the second child of an `mfrac` is an unknown element, the fraction should be rendered with a denominator indicating an error.

Under the lax schema profile, elements non-MathML namespaces are allowed in token elements, but not in other elements. This restriction on the location of elements from other namespaces largely eliminates the issues of positionally defined roles for elements in MathML, while still providing some flexibility for compound documents. Attributes from other namespaces are also allowed. MathML-processors that encounter unknown markup should behave as follows:

1. Unrecognized XML attributes should be silently ignored.
2. Unrecognized elements in token XML elements should be silently ignored.
3. An element from a non-MathML namespaces should be treated as an error. If the element is a child of a presentation element, should be handled as described in Section 3.3.5. If the element is a child of a content element should be handled as described in Section 4.2.9.

Considerations about mixing markup vocabularies in compound documents arise when the format is first designed. But once the format is fixed, it is not generally practical for specific software tools to further modify the content model to suit their needs. However, it is still frequently the case that such tools may need to persist additional information within a MathML instance. Since MathML is most often generated by authoring tools, a particularly common and important case is an authoring tool needing persist some information about its internal state along with a MathML expression, so that an author can pick up editing where he or she left off. For example, placeholders are often used to indicate incomplete parts of an expression, and a cursor position within an expression is sometimes persisted.

Applications needing to persist private data within MathML expressions should generally attempt to do so without altering the underlying content model, even in situations where it is feasible to do so. To support this, regardless of what may be allowed by the content model of a particular compound document format, MathML always permits the persistance of private data via the following strategies:

1. For small amounts of data, attributes from other namespaces are allows on all MathML elements.
2. For larger amounts of data, applications may use the `semantics` element presented in Section 5.1.
3. For authoring tools and other applications needing to associate particular actions with presentation MathML subtrees, e.g. to mark an incomplete expression to be filled in by an author, the `maction` element may be used. See Section 3.7.1.

Issue (ended-schema): The schema should exist in strict versions, prohibiting foreign markup and in lax or parametrized version to open support for external formats. (type parametrization in XML-schema, entity redifinition in DTD, something in RelaxNG)

6.4.1 Mixing MathML and HTML

In order to fully integrate MathML into XHTML, it should be possible not only to embed MathML in XHTML, as described in Section 6.2.1, but also to embed XHTML in MathML. However, the prob-

lem of supporting XHTML in MathML presents many difficulties. Therefore, at present, the MathML specification does not permit any XHTML elements within a MathML expression, although this may be subject to change in a future revision of MathML.

In most cases, XHTML elements (headings, paragraphs, lists, etc.) either do not apply in mathematical contexts, or MathML already provides equivalent or better functionality specifically tailored to mathematical content (tables, mathematics style changes, etc.). However, there are two notable exceptions, the XHTML anchor and image elements. For this functionality, MathML relies on the general XML linking and graphics mechanisms being developed by other W3C Activities.

6.4.1.1 Compatibility Suggestions

While the use of namespaces to embed MathML in other XML applications is completely described by the relevant W3C Recommendations, a certain degree of pragmatism is still called for at present. Support for XML, namespaces and rendering behaviors in popular user agents is not always fully in alignment with W3C Recommendations. When using namespace prefixes with MathML markup, use `m:` as a conventional prefix for the MathML namespace. Using an explicit prefix is probably safer for compatibility in current user agents. For example:

```
<body>
...
<m:math xmlns:m="http://www.w3.org/1998/Math/MathML">
<m:mrow>...<m:mrow>
</m:math>
...
</body>
```

Consult the W3C Math Working Group home page for compatibility and implementation suggestions for current browsers and other MathML-aware tools.

6.4.2 Linking

Issue (and-marking-ids): We wish to stop using xlink for links since it seems unimplemented and add the necessary attributes at presentation elements.

In MathML 3, an element is designated as a link by the presence of the `href` attribute. MathML has no element that corresponds to the HTML/XHTML anchor element *a*.

MathML allows the `href` attribute on all elements. However, links on elements that do not correspond to any part of a typical visual rendering should be avoided, since they will have no effect in typical user agents.

The list of presentation markup elements that do not ordinarily have a visual rendering, and thus should not be used as linking elements, is given in the table below.

MathML elements that should not be linking elements

`mprescripts`	`none`
`malignmark`	`maligngroup`

Note that MathML 2 had no direct support for linking, and followed the W3C Recommendation "XML Linking Language" [XLink] in defining links using an `xlink:href` attribute. This has changed, and MathML 3 now uses the `href` attribute for linking. However, particular compound document formats may specify the use of XML Linking with MathML elements. So, user agents that implement XML Linking in compound documents containing MathML should continue to support the use of the `xlink:href` attribute with MathML 3 as well.

For compound document formats that use XML Linking or other format-specific linking mechanisms, the `id` attribute should be used to specify locations for links into a MathML expressions. The `id` attribute is allowed on all elements, and values must be unique within the document, making it ideal for the this purpose.

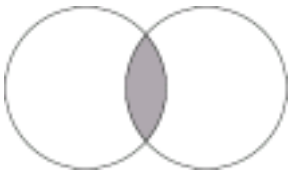
6.4.3 MathML and Graphical Markup

Apart from the introduction of new glyphs, many of the situations where one might be inclined to use an image amount to displaying labeled diagrams. For example, knot diagrams, Venn diagrams, Dynkin diagrams, Feynman diagrams and commutative diagrams all fall into this category. As such, their content would be better encoded via some combination of structured graphics and MathML markup. However, at the time of this writing, it is beyond the scope of the W3C Math Activity to define a markup language to encode such a general concept as 'labeled diagrams.' (See http://www.w3.org/Math for current W3C activity in mathematics and http://www.w3.org/Graphics for the W3C graphics activity.)

One mechanism for embedding additional graphical content is via the `semantics` element, as in the following example:

```
<semantics>
  <apply>
    <intersect/>
    <ci>A</ci>
    <ci>B</ci>
  </apply>
  <annotation-xml encoding="SVG1.1">
    <svg xmlns="http://www.w3.org/2000/svg"  viewBox="0 0 290 180">
      <clipPath id="a">
      <circle cy="90" cx="100" r="60"/>
      </clipPath>
      <circle fill="#AAAAAA" cy="90" cx="190"
              r="60" style="clip-path:url(#a)"/>
      <circle stroke="black" fill="none" cy="90" cx="100" r="60"/>
      <circle stroke="black" fill="none" cy="90" cx="190" r="60"/>
    </svg>
  </annotation-xml>
  <annotation-xml encoding="application/xhtml+xml">
    <img xmlns="http://www.w3.org/1999/xhtml" src="intersect.gif" alt="A intersect B"/>
  </annotation-xml>
</semantics>
```

Here, the `annotation-xml` elements are used to indicate alternative representations of the Content MathML depiction of the intersection of two sets. The first one is in the 'Scalable Vector Graphics' format [SVG1.1] (see [XHTML-MathML-SVG] for the definition of an XHTML profile integrating MathML and SVG), the second one uses the XHTML `img` element embedded as an XHTML fragment. In this situation, a MathML processor can use any of these representations for display, perhaps producing a graphical format such as the image below.



Note that the semantics representation of this example is given in Content MathML markup, as the first child of the `semantics` element. In this regard, it is the representation most analogous to the `alt` attribute of the `img` element in XHTML, and would likely be the best choice for non-visual rendering.

6.5 Using CSS with MathML

Issue (): The current section needs updating further in later drafts.

When MathML is rendered in an environment that supports [CSS2], controlling mathematics style properties with a CSS stylesheet is obviously desirable. MathML significantly redesigned the way presentation element style properties are organized to facilitate better interaction between MathML renderers and CSS style mechanisms. It introduced four *mathematics style* attributes. Roughly speaking, these attributes can be viewed as the proper selectors for CSS rules that affect MathML.

Controlling mathematics styling is not as simple as it might first appear because mathematics styling and text styling are quite different in character. In text, meaning is primarily carried by the relative positioning of characters next to one another to form words. Thus, although the font used to render text may impart nuances to the meaning, transforming the typographic properties of the individual characters leaves the meaning of text basically intact. By contrast, in mathematical expressions, individual characters in specific typefaces tend to function as atomic symbols. Thus, in the same equation, a bold italic 'x' and a normal italic 'x' are almost always intended to be two distinct symbols that mean different things. In traditional usage, there are eight basic typographical categories of symbols. These categories are described by mathematics style attributes, primarily the `mathvariant` attribute.

Text and mathematics layout also obviously differ in that mathematics uses 2-dimensional layout. As a result, many of the style parameters that affect mathematics layout have no textual analogs. Even in cases where there are analogous properties, the sensible values for these properties may not correspond. For example, traditional mathematical typography usually uses italic fonts for single character identifiers, and upright fonts for multicharacter identifier. In text, italicization does not usually depend on the number of letters in a word. Thus although a font-slant property makes sense for both mathematics and text, the natural default values are quite different.

Because of the difference between text and mathematics styling, only the styling aspects that do not affect layout are good candidates for CSS control. MathML 3.0 captures the most important properties with the mathematics style attributes, and users should try to use them whenever possible over more direct, but less robust, approaches. Further discussion and a sample CSS stylesheet may be found in [MathMLforCSS].

Generally speaking, the model for CSS interaction with the math style attributes runs as follows. A CSS style sheet might provide a style rule such as:

```
math *.[mathsize="small"] {
  font-size: 80%
}
```

This rule sets the CSS font-size properties for all children of the `math` element that have the `mathsize` attribute set to small. A MathML renderer would then query the style engine for the CSS environment, and use the values returned as input to its own layout algorithms. MathML does not specify the mechanism by which style information is inherited from the environment. However, some suggested rendering rules for the interaction between properties of the ambient style environment and MathML-specific rendering rules are discussed in Section 3.2.2, and more generally throughout Chapter 3.

It should be stressed, however, that some caution is required in writing CSS stylesheets for MathML. Because changing typographic properties of mathematics symbols can change the meaning of an equation, stylesheet should be written in a way such that changes to document-wide typographic styles

do not affect embedded MathML expressions. By using the MathML mathematics style attributes as selectors for CSS rules, this danger is minimized.

Another pitfall to be avoided is using CSS to provide typographic style information necessary to the proper understanding of an expression. Expressions dependent on CSS for meaning will not be portable to non-CSS environments such as computer algebra systems. By using the logical values of the new MathML 3.0 mathematics style attributes as selectors for CSS rules, it can be assured that style information necessary to the sense of an expression is encoded directly in the MathML.

MathML 3.0 does not specify how a user agent should process style information, because there are many non-CSS MathML environments, and because different users agents and renderers have widely varying degrees of access to CSS information. In general, however, developers are urged to provide as much CSS support for MathML as possible.

6.5.1 Order of processing attributes versus style sheets

CSS or analogous style sheets can specify changes to rendering properties of selected MathML elements. Since rendering properties can also be changed by attributes on an element, or be changed automatically by the renderer, it is necessary to specify the order in which changes requested by various sources should occur. An example of automatic adjustment is what happens for `fontsize`, as explained in the discussion on `scriptlevel` in Section 3.3.4. In the case of 'absolute' changes, i.e., setting a new property value independent of the old value (as opposed to 'relative' changes, such as increments or multiplications by a factor), the absolute change performed last will be the only absolute change which is effective, so the sources of changes which should have the highest priority must be processed last.

In the case of CSS, the order of processing of changes from various sources which affect one MathML element's rendering properties should be as follows:

(first changes; lowest priority)

- Automatic changes to properties or attributes based on the type of the parent element, and this element's position in the parent, as for the changes to `fontsize` in relation to `scriptlevel` mentioned above; such changes will usually be implemented by the parent element itself before it passes a set of rendering properties to this element
- From a style sheet from the reader: styles which are *not* declared 'important'
- Explicit attribute settings on this MathML element
- From a style sheet from the author: styles which are *not* declared 'important'
- From a style sheet from the author: styles which *are* declared 'important'
- From a style sheet from the reader: styles which *are* declared 'important'

(last changes; highest priority)

Note that the order of the changes derived from CSS style sheets is specified by CSS itself (this is the order specified by CSS2). The following rationale is related only to the issue of where in this pre-existing order the changes caused by explicit MathML attribute settings should be inserted.

Rationale: MathML rendering attributes are analogous to HTML rendering attributes such as `align`, which the CSS section on cascading order specifies should be processed with the same priority. Furthermore, this choice of priority permits readers, by declaring certain CSS styles as 'important', to decide which of their style preferences should override explicit attribute settings in MathML. Since MathML expressions, whether composed of 'presentation' or 'content' elements, are primarily intended to convey meaning, with their 'graphic design' (if any) intended mainly to aid in that purpose but not to be essential in it, it is likely that readers will often want their own style preferences to have priority; the main exception will be when a rendering attribute is intended to alter the meaning conveyed by an expression, which is generally discouraged in the presentation attributes of MathML.

Chapter 7

Characters, Entities and Fonts

7.1 Introduction

Issue (): Many of the tables in chapter 6 need to be updated and regenerated. In this draft references to tables in chapter 6 link to the published MathML2 Recommendation, and are marked [MathML2]

Resolution: Separate xml-entity-names WD

Notation and symbols have proved very important for mathematics. Mathematics has grown in part because its notation continually changes toward being succinct and suggestive. There have been many new signs developed for use in mathematical notation, and mathematicians have not held back from making use of many symbols originally introduced elsewhere. The result is that mathematics makes use of a very large collection of symbols. It is difficult to write mathematics fluently if these characters are not available for use. It is difficult to read mathematics if corresponding glyphs are not available for presentation on specific display devices.

The W3C Math Working Group therefore took on directly the task of specifying part of the full mechanism needed to proceed from notation to final presentation, and has collaborated with the STIX Fonts Project and Unicode Technical Committee (UTC) in undertaking specification of the rest.

This chapter of the MathML specification contains a listing of character names for use with MathML, recommendations for their use, and warnings to pay attention to the correct form of the corresponding code points given in the UCS (Universal Character Set) as codified in Unicode and ISO 10646 [Unicode] and the Unicode Web site. For simplicity we refer to this character set by the short name Unicode. Though Unicode changes from time to time so that it is specified exactly by using version numbers, unless this brings clarity on some point we do not use them. MathML 2.0 (Second Edition) is based on Unicode 4.0, and MathML 3.0 on Unicode 5.1.)

While a long process of review and adoption by UTC and ISO/IEC of the characters of special interest to mathematics and MathML is now complete, more characters may be added in the future. To ensure any possible corrections to relevant standards are taken into account, and for the latest character tables and font information, see the W3C Entities page and the Unicode site, notably Unicode Work in Progress and Unicode Technical Report #25 "Unicode Support for Mathematics".

A MathML token element (see Section 3.2, Section 4.2.1, Section 4.2.2, Section 4.2.3) takes as content a sequence of *MathML Characters*. MathML Characters are defined to be either Unicode characters legal in XML documents or `mglyph` elements. The latter are used to represent characters that do not have a Unicode encoding, as described in Section 3.2.9. Because the Unicode UCS provided approximately one thousand special alphabetic characters for the use of mathematics with Unicode 3.1, and over 900 further special symbols in Unicode 3.2, the need for `mglyph` should be rare.

7.2 Unicode Character Data

Any character allowed by XML may be used in MathML in an XML document. The legal characters have the hexadecimal code numbers 09 (tab = U+0009), 0A (line feed = U+000A), 0D (carriage return = U+000D), 20-D7FF (U+0020..U+D7FF), E000-FFFD (U+E000..U+FFFD), and 10000-10FFFF (U+010000..U+10FFFF). The notation, just introduced in parentheses, beginning with U+ is that recommended by Unicode for referring to Unicode characters [see [Unicode], page xxviii]. The exclusions above code number D7FF are of the blocks used in surrogate pairs, and the two characters guaranteed not to be Unicode characters at all. U+FFFE is excluded to allow determination of byte order in certain encodings.

There are essentially three different ways of encoding character data.

- Using characters directly: For example, an A may be entered as 'A' from a keyboard (character U+0041). This option is only available if the character encoding specified for the XML document includes the character. Most commonly used encodings will have 'A' in the ASCII position. In many encodings, characters may need more than one byte. Note that if the document is, for example, encoded in Latin-1 (ISO-8859-1) then *only* the characters in that encoding are available directly. Using UTF-8 or UTF-16, the only two encodings that all XML processors are required to accept, mathematical symbols can be encoded as character data.
- Using numeric XML character references: Using this notation, 'A' may be represented as A (decimal) or A (hex). Note that the numbers always refer to the Unicode encoding (and not to the character encoding used in the XML file). By using character references it is always possible to access the entire Unicode range. For a general XML vocabulary, there is a disadvantage to this approach: character references may not be used in XML element or attribute names. However, this is not an issue for MathML, as all element names in MathML are restricted to ASCII characters.
- Using entity references: The MathML DTD defines internal entities that expand to character data. Thus for example the entity reference é may be used rather than the character reference "é or, if, for example, the document is encoded in ISO-8859-1, the character é. An XML fragment that uses an entity reference which is not defined in a DTD is not well-formed; therefore it will be rejected by an XML parser. For this reason *every* fragment using entity references *must* use a DOCTYPE declaration which specifies the MathML DTD, or a DTD that at least declares any entity reference used in the MathML instance. The need to use a DOCTYPE complicates inclusion of MathML in some documents. However, entity references are very useful for small illustrative examples, and are used in most examples in this document.

7.3 Entity Declarations

Earlier versions of this MathML specification included detailed listings of the entity definitions to be used with the MathML DTD. These entity definitions are of more general use, and have now been separated into a separate document, [Entities]. That document describes several entity sets, not all of them are used in the MathML DTD, although an XML document that includes MathML may reference any entity definitions. The standard MathML DTD references the following entity sets:

- isobox Box and Line Drawing
- isocyr1 Russian Cyrillic
- isocyr2 Non-Russian Cyrillic
- isodia Diacritical Marks

- isolat1 Added Latin 1
- isolat2 Added Latin 2
- isonum Numeric and Special Graphic
- isopub Publishing
- isoamsa Added Math Symbols: Arrow Relations
- isoamsb Added Math Symbols: Binary Operators
- isoamsc Added Math Symbols: Delimiters
- isoamsn Added Math Symbols: Negated Relations
- isoamso Added Math Symbols: Ordinary
- isoamsr Added Math Symbols: Relations
- isogrk3 Greek Symbols
- isomfrk Math Alphabets: Fraktur
- isomopf Math Alphabets: Open Face
- isomscr Math Alphabets: Script
- isotech General Technical
- mmlextra Additional MathML Symbols
- mmlalias MathML Aliases

7.4 Special Characters Not in Unicode

For special purposes, one may need to use a character which is not in Unicode. In these cases one may use the `mglyph` element for direct access to a glyph as an image, or (in some systems) from a font that uses a non-uniocde encoding. All MathML token elements that accept character data also accept an `mglyph` in their content. Beware, however, that use of `mglyph` to access a font is deprecated and the mechanism may not work in all systems. The `mglyph` element should always supply an alternatve representation in its `alt` attribute.

7.5 Mathematical Alphanumeric Symbols

A noticeable feature of mathematical and scientific writing is the use of single letters to denote variables and constants in a given context. The increasing complexity of science has led to the use of certain common alphabet and font variations to provide enough special symbols of this letter-like type. These denotations are in fact *not* letters that may be used to make up words with recognized meanings, but individual carriers of semantics themselves. Writing a string of such symbols is usually interpreted in terms of some composition law, for instance, multiplication. Many letter-like symbols may be quickly interpreted by specialists in a given area as of a certain mathematical type: for instance, bold symbols, whether based on Latin or Greek letters, as vectors in physics or engineering, or fraktur symbols as Lie algebras in part of pure mathematics. To this end the STIX Fonts Project defined a set of mathematical characters all of which are included in Unicode 5.0.

The additional Mathematical Alphanumeric Symbols provided in Unicode 3.1 have code points in the range U+1D400 to U+1D7FF in *Plane 1*, that is, in the first plane with Unicode values higher than 2^{16}. This plane of characters is also known as the Secondary Multilingual Plane (SMP), in contrast to the Basic Multilingual Plane (BMP) which was originally the entire extent of Unicode. Support for Plane 1 characters in currently deployed software is not always reliable, but it should be possible in multilingual operating systems, since Plane 2 has many Chinese characters that must be displayable in East Asian locales.

As discussed in Section 3.2.2, MathML offers an alternative mechanism to specify mathematical alphabetic characters. This alternative spans the gap between the specification of Unicode 3.1 and its associated deployment in software and fonts. Namely, one uses the `mathvariant` attribute on the surrounding token element, which will most commonly be `mi`. In this section we detail the correspondence that a MathML processor should apply between certain characters in *Plane 0* (BMP) of Unicode, modified by the `mathvariant` attribute, and the Plane 1 Mathematical Alphanumeric Symbol characters.

The basic idea of the correspondence is fairly simple. For example, a Mathematical Fraktur alphabet is in Plane 1, and the code point for Mathematical Fraktur A is U+1D504. Thus using these characters, a typical example might be

```
<mi>&#x1D504;</mi>
```

However, an alternative, equivalent markup would be to use the standard A and modify the identifier using the `mathvariant` attribute, as follows:

```
<mi mathvariant="fraktur">A</mi>
```

The exact correspondence between a mathematical alphabetic character and an unstyled character is complicated by the fact that certain characters that were already present in Unicode are not in the 'expected' sequence.

Mathematical Alphanumeric Symbol characters should not be used for styled text. For example, Mathematical Fraktur A must not be used to just select a blackletter font for an uppercase A. Doing this sort of thing would create problems for searching, restyling (e.g. for accessibility), and many other kinds of processing.

7.6 Non-Marking Characters

Some characters, although important for the quality of print or alternative rendering, do not have glyph marks that correspond directly to them. They are called here non-marking characters. Their roles are discussed in Chapter 3 and Chapter 4.

In MathML 2 control of page composition, such as line-breaking, is effected by the use of the proper attributes on the `mspace` element.

The characters below are not simple spacers. They are especially important new additions to the UCS because they provide textual clues which can increase the quality of print rendering, permit correct audio rendering, and allow the unique recovery of mathematical semantics from text which is visually ambiguous.

Unicode codepoint	Unicode name	Description
02061	FUNCTION APPLICATION	character showing function application in presentation tagging (Section 3.2.5
02062	INVISIBLE TIMES	marks multiplication when it is understood without a mark (Section 3.2.5
02063	INVISIBLE SEPARATOR	used as a separator, e.g., in indices (Section 3.2.5
02064*	INVISIBLE PLUS	marks addition, especialy in constructs such a $1\frac{1}{2}$ (Section 3.2.5

*Character U+2064 has been accepted by the UTC and ISO for inclusion into the next revision of Unicode, 5.1

Appendix A

Parsing MathML

A.1 Use of MathML as Well-Formed XML

Issue ():DTD and W3C XML Schema need updating to MathML3

Issue (): Should we add a (normative?) Relax NG schema.

Resolution: We make it normative

A MathML document must be a well-formed XML document using elements in the MathML namespace as defined by this specification, however it is not required that the document refer to any specific Document Type Definition (DTD) or schema that specifies MathML. It is sometimes advantagous *not* to specify such a language definition as these files are large, often much larger than the MathML expression and unless they have been previously cached by the MathML application, the time taken to fetch the DTD or schema may have an appreciable effect on the processing of the MathML document.

Note also that if no DTD is specified with a DOCTYPE declaration, that entity references (for example to refer to MathML characters by name) may not be used. The document should be encoded in an encoding (for example UTF-8) in which all needed characters may be encoded as character data, or characters may be referenced using numeric character references, for example ∫ rather than ∫

If a MathML fragment is parsed without a DTD, in other words as a well-formed XML fragment, it is the responsibility of the processing application to treat the white space characters occurring outside of token elements as not significant.

However, in many circumstances, especially while producing or editing MathML, it is useful to use a language definition, to constrain the editing process or to check the correctness of generated files. The following section, Section A.2, discusses the RelaxNG Schema for MathML3 [RelaxNG], which forms a normative part of the specification. Following that, Section A.4, and Section A.3 discuss alternative languages definition using the document type definitions (DTD) and the W3C XML schema language, [XMLSchemas], both of which are derived from the normative RelaxNG schema automatically. One should note that the schema definitions of the language is currently stricter than the DTD version. That is, a schema validating processor will declare invalid documents that are declared valid by a (DTD) validating XML parser. This is partly due to the fact that the XML schema language may express additional constraints not expressable in the DTD, and partly due to the fact that for reasons of compatibility with earlier releases, the DTD is intentionally forgiving in some places and does not enforce constraints that are specified in the text of this specification.

A.2 Using the RelaxNG Schema for MathML3

MathML documents should be validated using the RelaxNG Schema for MathML, either in the XML encoding (`http://www.w3.org/Math/RelaxNG/mathml3/mathml3.rng`) or in compact notation

(http://www.w3.org/Math/RelaxNG/mathml3/mathml3.rnc) which is also shown below.

In contrast to DTDs there is no in-document method to associate a RelaxNG schema with a document.

We provide five RelaxNG schemata for sub-languages of MathML3:

- The grammar for full MathML
- The grammar for Presentation MathML without content elements mixed in
- The grammar for strict Content MathML3
- The grammar for pragmatic Content MathML3 without presentation MathML in token elements
- The grammar for the deprecated parts of MathML

Editor's note: MiKoI think that this is no longer correct

we will present them in detail in the next sections below. As the compact notation for RelaxNG grammars is more readable, we will use this format here.

Note that the RelaxNG grammars here are considerably more strict than the MathML2 DTDs (even in strict mode).

A.2.1 Full MathML

The RelaxNG schema for full MathML builds on the schema describing the various arts of teh language which are given in the following sections. It can be found at http://www.w3.org/Math/RelaxNG/mathml3/mathml3.rnc.

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2008 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
#
#
#     Revision:   $Id: mathml3.rnc,v 1.7 2008/11/09 00:24:40 dcarlis Exp $
#

default namespace m = "http://www.w3.org/1998/Math/MathML"

## Content  MathML
include "mathml3-content.rnc"

## Presentation MathML
include "mathml3-presentation.rnc"

## math and semantics common to both Content and Presentation
include "mathml3-common.rnc"

# Derived from MathML2 DTD
#
```

```
# Revision History:
#
#         Initial draft (syntax = XML) 1997-05-09
#            Stephen Buswell
#         Revised 1997-05-14
#            Robert Miner
#         Revised 1997-06-29 and 1997-07-02
#            Stephen Buswell
#         Revised 1997-12-15
#            Stephen Buswell
#         Revised 1998-02-08
#            Stephen Buswell
#         Revised 1998-04-04
#            Stephen Buswell
#         Entities and small revisions 1999-02-21
#            David Carlisle
#         Added attribute definitionURL to ci and cn 1999-10-11
#            Nico Poppelier
#         Additions for MathML 2  1999-12-16
#            David Carlisle
#         Namespace support 2000-01-14
#            David Carlisle
#         XHTML Compatibility 2000-02-23
#            Murray Altheim
#         New content elements 2000-03-26
#            David Carlisle
#         Further revisions for MathML2 CR draft 2000-07-11
#            David Carlisle
#         Further revisions for MathML2 CR draft 2000-10-31
#            David Carlisle
#         Revisions for Unicode 3.2  2002-05-21
#            David Carlisle
#         Add width and side attributes to mtable (to align with the specification) 2002-06-
#            David Carlisle
#         Use %XLINK.prefix rather than hardwired xlink:, add xlink:type 2002-06-12
#            David Carlisle
#         Add missing numalign and denomalign attributes for mfrac 2002-07-05
#            David Carlisle
#         Add MathMLstrict entity and related extra constraints 2002-12-05
#            David Carlisle
#         Add support for xi:schemaLocation 2003-04-05
#            David Carlisle
#         Removed actiontype from mstyle (to match spec) 2003-04-07
#            David Carlisle
#         Additional constraints for MathMLstrict code (From Simon
#            Pepping on www-math list) 2003-05-22
#            David Carlisle
#         Add missing minlabelspacing attribute (From Simon
#            Pepping on www-math list) 2003-05-22
#            David Carlisle
```

```
#        Removed restricted menclose notation checking from MathMLstrict 2003-09-08
#           David Carlisle
#
#      MathML3: Switch to Relax NG as development language for MathML Schema
#           David Carlisle and Michael Kohlhase

#
#
```

A.2.2 The Grammar for Presentation MathML

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2008 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
#
#
#     Revision:   $Id: mathml3-presentation.rnc,v 1.8 2008/11/09 11:15:50 mkohlhas2 Exp $
#
#    Update to MathML3 and Relax NG: David Carlisle and Michael Kohlhase

default namespace m = "http://www.w3.org/1998/Math/MathML"


math.content |= ContInPres*


Browser-interface.attrib = attribute baseline {text}?,


                           attribute alttext {text}?,
           attribute type {text}?,
   attribute name {text}?,
   attribute height {text}?,
   attribute width {text}?

math.attlist |= Browser-interface.attrib,attribute display {"block" | "inline"}?,
                attribute dir {"ltr" | "rtl"}?,
                linebreak.attrib

simple-size = "small" | "normal" | "big"

centering.values = "left" | "center" | "right"
```

```
named-space = "veryverythinmathspace" | "verythinmathspace" | "thinmathspace" |
              "mediummathspace" |
              "thickmathspace" | "verythickmathspace" | "veryverythickmathspace"
thickness = "thin" | "medium" | "thick"

# number with units used to specified lengths

length-with-unit =
    text #{pattern="(-?([0-9]+|[0-9]*\.[0-9]+)(em|ex|px|in|cm|mm|pt|pc|%))|0"}
length-with-optional-unit =
   text #{pattern="-?([0-9]+|[0-9]*\.[0-9]+)(em|ex|px|in|cm|mm|pt|pc|%)?"}


# colors defined as RGB
RGB-color = text # {pattern="#(([0-9]|[a-f]){3}|([0-9]|[a-f]){6})"}

# The mathematics style attributes. These attributes are valid on all
#     presentation token elements except "mspace" and "mglyph", and on no
#     other elements except "mstyle".

Token-style.attrib = attribute mathvariant
       {"normal" | "bold" | "italic" | "bold-italic" | "double-struck" |
                          "bold-fraktur" | "script" | "bold-script" | "fraktur" |
  "sans-serif" | "bold-sans-serif" | "sans-serif-italic" |
"sans-serif-bold-italic" | "monospace" |
                          "initial" | "tailed" | "looped" | "stretched"}?,
                      attribute mathsize {simple-size | length-with-unit}?,

                      attribute mathcolor {text}?,
         attribute mathbackground {text}?

truefalse = "true" | "false"

Operator.attrib =
# this attribute value is normally inferred from the position of
# the operator in its "<mrow">
   attribute form {"prefix" | "infix" | "postfix"}?,
   # set by dictionary, else it is "thickmathspace"
   attribute lspace {length-with-unit | named-space}?,
   # set by dictionary, else it is "thickmathspace"
   attribute rspace {length-with-unit | named-space}?,
   # set by dictionnary, else it is "false"
   attribute fence {truefalse}?,
   # set by dictionnary, else it is "false"
   attribute separator {truefalse}?,
   # set by dictionnary, else it is "false"
   attribute stretchy {truefalse}?,
   # set by dictionnary, else it is "true"
   attribute symmetric {truefalse}?,
   # set by dictionnary, else it is "false"
```

```
   attribute movablelimits {truefalse}?,
   # set by dictionnary, else it is "false"
   attribute accent {truefalse}?,
   # set by dictionnary, else it is "false"
   attribute largeop {truefalse}?,
   attribute minsize {length-with-unit | named-space}?,
   attribute maxsize {length-with-unit | named-space | "infinity" | xsd:float}?


mglyph = element {mglyph} {MathML.Common.attrib,
                     attribute alt {text}?,
                     (attribute src {xsd:anyURI}| attribute fontfamily {text}),
     attribute width {text}?,
     attribute height {text}?,
     attribute baseline {text}?,
     attribute index {xsd:positiveInteger}?}


linethickness.attrib = attribute linethickness {length-with-optional-unit|thickness}
mline = element {mline} {MathML.Common.attrib,
      linethickness.attrib?,
      attribute spacing {text}?,
      attribute length {length-with-unit | named-space}?}

Glyph-alignmark = malignmark|mglyph

mi = element {mi} {MathML.Common.attrib,Token-style.attrib,(Glyph-alignmark|text)*}

mo = element {mo} {MathML.Common.attrib,Operator.attrib,Token-style.attrib,
                   linebreak.attrib,
                  (text|Glyph-alignmark)*}

mn = element {mn} {MathML.Common.attrib,Token-style.attrib,(text|Glyph-alignmark)*}

mtext = element {mtext} {MathML.Common.attrib,Token-style.attrib,(text|Glyph-alignmark)*}

ms = element {ms} {MathML.Common.attrib,Token-style.attrib,
                  attribute lquote {text}?,
 attribute rquote {text}?,
 (text|Glyph-alignmark)*}

# And the group of any token
Pres-token = mi | mo | mn | mtext | ms

msub = element {msub} {MathML.Common.attrib,
                   attribute subscriptshift {length-with-unit}?,
                   ContInPres,ContInPres}

msup = element {msup} {MathML.Common.attrib,
                   attribute supscriptshift {length-with-unit}?,
```

```
                  ContInPres,ContInPres}

msubsup = element {msubsup} {MathML.Common.attrib,
                     attribute subscriptshift {length-with-unit}?,
                     attribute supscriptshift {length-with-unit}?,
                     ContInPres,ContInPres,ContInPres}

munder = element {munder} {MathML.Common.attrib,
                         attribute accentunder {truefalse}?,
                         ContInPres,ContInPres}

mover = element {mover} {MathML.Common.attrib,
                       attribute accent {truefalse}?,
                       ContInPres,ContInPres}

munderover = element {munderover} {MathML.Common.attrib,
                                   attribute accentunder {truefalse}?,
                                   attribute accent {truefalse}?,
                                   ContInPres,ContInPres,ContInPres}

PresExp-or-none = ContInPres | none
mmultiscripts = element {mmultiscripts}{MathML.Common.attrib,
                                ContInPres,
      (PresExp-or-none,PresExp-or-none)*,
      (mprescripts,(PresExp-or-none,PresExp-or-none)*)?}
none = element {none} {empty}
mprescripts = element {mprescripts} {empty}

Pres-script = msub|msup|msubsup|munder|mover|munderover|mmultiscripts
linebreak-values =
   "auto" | "newline" | "indentingnewline" | "nobreak" |
   "goodbreak" | "badbreak"
mspace = element {mspace} {MathML.Common.attrib,
                         attribute width {length-with-unit | named-space}?,
             attribute height {length-with-unit}?,
             attribute depth {length-with-unit}?,
attribute spacing {text}?,
                     linebreak.attrib}

mrow = element {mrow} {MathML.Common.attrib,ContInPres*}

mfrac = element {mfrac} {MathML.Common.attrib,
                       attribute bevelled {truefalse}?,
                       attribute denomalign {centering.values}?,
       attribute numalign {centering.values}?,
       linethickness.attrib?,
       ContInPres,ContInPres}
msqrt = element {msqrt} {MathML.Common.attrib,ContInPres*}

mroot = element {mroot} {MathML.Common.attrib,ContInPres,ContInPres}
```

```
mpadded-space = text



mpadded-width-space = text

mpadded = element {mpadded} {MathML.Common.attrib,
                   attribute width {mpadded-width-space}?,
      attribute lspace {mpadded-space}?,
      attribute height {mpadded-space}?,
      attribute depth {mpadded-space}?,
      ContInPres*}

mphantom = element {mphantom} {MathML.Common.attrib,ContInPres*}

mfenced = element {mfenced} {MathML.Common.attrib,
                          attribute open {text}?,
                      attribute close {text}?,
      attribute separators {text}?,
   ContInPres*}

notation-values = "actuarial"|"longdiv"|"radical"|
                              "box"|"roundedbox"|"circle"|
                              "left"|"right"|"top"|"bottom"|
                              "updiagonalstrike"|"downdiagonalstrike"|
                              "verticalstrike"|"horizontalstrike" | "madruwb"
menclose = element {menclose} {MathML.Common.attrib,
                           attribute notation {list{notation-values*}}?,
  ContInPres*}

# And the group of everything
Pres-layout = mrow|mfrac|msqrt|mroot|mpadded|mphantom|mfenced|menclose

Table-alignment.attrib = attribute rowalign  {text}?,
        attribute columnalign {text }?,
        attribute groupalign {text}?

mtr.content = mtd
mtr = element {mtr} {Table-alignment.attrib, MathML.Common.attrib,(mtr.content)+}

mlabeledtr = element {mlabeledtr} {
  Table-alignment.attrib,MathML.Common.attrib,(mtr.content)*}

mtd = element {mtd} {MathML.Common.attrib,
                   Table-alignment.attrib,
                   attribute columnspan {xsd:positiveInteger}?,
      attribute rowspan {xsd:positiveInteger}?,
   ContInPres*}
```

```
mtable.content = mtr|mlabeledtr
mtable = element {mtable} {Table-alignment.attrib,
                           attribute align {text}?,
 attribute alignmentscope {text}?,
 attribute columnwidth {text}?,
    attribute width {text}?,
    attribute rowspacing {text}?,
    attribute columnspacing {text}?,
    attribute rowlines {text}?,
    attribute columnlines {text}?,
    attribute frame {"none" | "solid" | "dashed"}?,
    attribute framespacing {text}?,
    attribute equalrows {truefalse}?,
    attribute equalcolumns {truefalse}?,
    attribute displaystyle {truefalse}?,
 attribute side {"left"|"right"|"leftoverlap"|"rightoverlap"}?,
    attribute minlabelspacing {length-with-unit}?,
    MathML.Common.attrib,
 (mtable.content)*}

maligngroup = element {maligngroup} {MathML.Common.attrib,
     attribute groupalign {"left" | "center" | "right" | "decimalpoint"}?}

malignmark = element {malignmark} {MathML.Common.attrib,attribute edge {"left" | "right"}?}

Pres-table = mtable|maligngroup|malignmark

mcolumn = element {mcolumn} {MathML.Common.attrib,
     attribute align {"left" | "right"}?,ContInPres*}

mstyle = element {mstyle} {MathML.Common.attrib,
                           linebreak.attrib,
                           attribute scriptlevel {xsd:integer}?,
                           attribute displaystyle {truefalse}?,
 attribute scriptsizemultiplier {xsd:decimal}?,
    attribute scriptminsize {length-with-unit}?,
    attribute background {text}?,
    attribute veryverythinmathspace {length-with-unit}?,
    attribute verythinmathspace {length-with-unit}?,
 attribute thinmathspace {length-with-unit}?,
                           attribute mediummathspace {length-with-unit}?,
                           attribute thickmathspace {length-with-unit}?,
                           attribute verythickmathspace {length-with-unit}?,
                           attribute veryverythickmathspace {length-with-unit}?,
                           linethickness.attrib?,
    Operator.attrib,Token-style.attrib,
 ContInPres*}

merror = element {merror} {MathML.Common.attrib,ContInPres*}
```

```
maction = element {maction} {MathML.Common.attrib,
   attribute actiontype {text}?,
                       attribute selection {xsd:positiveInteger}?,
      ContInPres*}

semantics-pmml = element {semantics} {semantics.attribs,PresExp, semantics-annotation*}

PresExp = Pres-token | Pres-layout | Pres-script | Pres-table
      |  mspace | mline | mcolumn |  maction | merror | mstyle
      | semantics-pmml

ContInPres |= PresExp
```

Issue (): David wrote in an e-mail: `length-with-unit` doesn't allow white space (anywhere) which (if any) of the following do we want to allow " 2em ", "2 em", "- 2 em". Also it insists on starting with a digit or -, but do we want to allow ".5em" "-.5em"However we do claim css compatibility here which may suggest some answers to the above `http://www.w3.org/TR/CSS21/syndata.html#length-units`.css allows an optional leading `+` as well `+2em` css requires number to "immediately" follow any sign and the unit to "immediately" follow the number, which I think means no intervening white space. css <number> are allowed to start with a `.` so `.5em` is allowed. css insists on a digit following a `.` so `5.em` is not allowed.Once we have firm answers to the above it should be easy to drop the regexp back in, and make the text match.I think we should not allow white space except at beginning and end but allow a leading `+` (a change from mathml2) and allow no digits before the `.`, but insist on digits after a `.` which would be `[\-\+]?([0-9]+(\.[0-9]+)?|\.[0-9]+)(em|ex|px|in|cm|mm|pt|pc|%)|0` as written this doesn't allow " `2em` " but I think we can set white space trim properties to apply before the regex is checked (I'll check)

A.2.3 The Grammar for Strict Content MathML3

The grammar for Strict Content MathML3 can be found at `http://www.w3.org/Math/RelaxNG/mathml3/mathml3-strict.rnc`.

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2008 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
#
#
#     Revision:   $Id: mathml3-strict.rnc,v 1.8 2008/11/09 11:15:50 mkohlhas2 Exp $
#
#    Update to MathML3 and Relax NG: David Carlisle and Michael Kohlhase
#
#  This is the RelaxNG schema module for the strict content part of MathML.
```

```
default namespace m = "http://www.w3.org/1998/Math/MathML"

include "mathml3-common.rnc"

math.content |= ContExp


opel.content = text

# we want to extend this in pragmatic CMathML, so we introduce abbrevs here.

cn.content = text |(cn,cn)
cn.type.vals  = "integer"|"real"|"double"

cn = element {cn} {attribute base {text}?,
                  attribute type {cn.type.vals}?,
    Definition.attrib,
    MathML.Common.attrib,
 (cn.content)*}

ci = element {ci} {attribute type {xsd:string}?,
                  attribute nargs {xsd:string}?,
 attribute occurrence {xsd:string}?,
                  Definition.attrib,
    MathML.Common.attrib,
 opel.content,
 name.attrib?}

cdname.attrib = attribute cd {xsd:NCName}

csymbol       = element {csymbol} {MathML.Common.attrib,
                          Definition.attrib,cdname.attrib?,cdbase.attrib?,
 opel.content}

# the content of the apply element, leave it empty and extend it later
apply = element {apply} {MathML.Common.attrib,cdbase.attrib?,apply.content}
apply-head = apply|bind|ci|csymbol|semantics-apply
apply.content = apply-head,ContExp*
semantics-apply = element {semantics} {semantics.attribs,apply-head, semantics-annotation*}

qualifier = notAllowed

# the content of the bind element, leave it empty and extend it later
bind = element {bind} {MathML.Common.attrib,cdbase.attrib?,bind.content}
bind-head = apply|csymbol|semantics-bind
bind.content = bind-head,bvar*,qualifier?,ContExp
semantics-bind   = element {semantics} {semantics.attribs,bind-head, semantics-annotation*}

bvar = element {bvar} {MathML.Common.attrib,cdbase.attrib?,bvar-head}
```

```
bvar-head = ci|semantics-bvar
semantics-bvar   = element {semantics} {semantics.attribs,bvar-head, semantics-annotation*}

share = element {share} {MathML.Common.attrib,attribute href {xsd:anyURI}}

# the content of the cerror element, leave it empty and extend it later
cerror = element {cerror} {MathML.Common.attrib,cdbase.attrib?,cerror.content}
cerror-head = csymbol|apply|semantics-cerror
cerror.content = cerror-head,ContExp*
semantics-cerror = element {semantics} {semantics.attribs,cerror-head, semantics-annotation

semantics-cmml = element {semantics} {semantics.attribs,ContExp, semantics-annotation*}

ContExp = cn| ci | csymbol | apply | bind | share | cerror | semantics-cmml
```

A.2.4 The Grammar for Pragmatic MathML

The grammar for pragmatic MathML3 can be found at http://www.w3.org/Math/RelaxNG/mathml3/mathml3-pragmatic.rnc.

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2008 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
#
#
#     Revision:   $Id: mathml3-pragmatic.rnc,v 1.10 2008/11/09 17:55:28 dcarlis Exp $
#
#    Update to MathML3 and Relax NG: David Carlisle and Michael Kohlhase
#
#     This is the RelaxNG schema module for the pragmatic content part of
#     MathML (but without the presentation in token elements).

default namespace m = "http://www.w3.org/1998/Math/MathML"


## the content of "cn" may have <sep> elements in it
sep = element {sep} {empty}
cn.content |= (sep|text|Glyph-alignmark)*
cn.type.vals |= "e-notation"|"rational"|"complex-cartesian"|"complex-polar"|"constant"

## allow degree in bvar
degree = element {degree} {MathML.Common.attrib,ContExp}
```

```
logbase = element {logbase} {MathML.Common.attrib,ContExp}
momentabout = element {momentabout} {MathML.Common.attrib,ContExp}
bvar-head |= (degree?,ci)|(ci,degree?)

## allow degree to modify <root/>
apply.content |= root_arith1_elt,degree,ContExp*
apply.content |= moment_s_data1_elt,(degree? & momentabout?),ContInPres*
apply.content |= log_transc1_elt,logbase,ContExp*

##allow apply to act as a binder
apply.content |= bind.content

domainofapplication = element {domainofapplication} {Definition.attrib,MathML.Common.attrib

lowlimit = element {lowlimit} {Definition.attrib,MathML.Common.attrib,cdbase.attrib?,ContEx
uplimit = element {uplimit} {Definition.attrib,MathML.Common.attrib,cdbase.attrib?,ContExp+

condition = element {condition} {Definition.attrib,cdbase.attrib?,ContExp}

## allow the non-strict qualifiers
qualifier |= domainofapplication|(uplimit,lowlimit?)|(lowlimit,uplimit?)|degree|condition

## we collect the operator elements by role
opel.constant = notAllowed
opel.binder = notAllowed
opel.application = notAllowed
opel.semantic-attribution = notAllowed
opel.attribution = notAllowed
opel.error = notAllowed

opels = opel.constant | opel.binder | opel.application |
        opel.semantic-attribution | opel.attribution |
opel.error
container = notAllowed

## the values of the MathML type attributes;
MathMLType |= "real" | "complex" | "function" | "algebraic" | "integer"


## we instantiate the strict content model by structure checking
apply-binder-head = semantics-apply-binder|opel.binder
apply.content |= apply-binder-head,bvar*,qualifier?,ContExp*
semantics-apply-binder = element {semantics} {semantics.attribs,apply-binder-head, semantic

apply-head |= opel.application
bind-head |= opel.binder
cerror-head |= opel.error

## allow all functions, constants, and containers to be content expressions on their own
ContExp |= opel.constant|opel.application|container
```

```
# allow no body
bind.content |= bind-head,bvar*,qualifier?

# not sure what a sequence of things is supposed to map to in strict/OM
# but is definitely allowed in pragmatic
# see Content/SequencesAndSeries/product/rec-product3
math.content |= ContExp*

opel.content |= PresExp|Glyph-alignmark

#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2008 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
#
#
#     Revision:   $Id: mathml3-pragmatic.rnc,v 1.10 2008/11/09 17:55:28 dcarlis Exp $
#
#    Update to MathML3 and Relax NG: David Carlisle and Michael Kohlhase
#
#     This is the RelaxNG schema module for the pragmatic content part of
#     MathML (but without the presentation in token elements).

default namespace m = "http://www.w3.org/1998/Math/MathML"


## the content of "cn" may have <sep> elements in it
sep = element {sep} {empty}
cn.content |= (sep|text|Glyph-alignmark)*
cn.type.vals |= "e-notation"|"rational"|"complex-cartesian"|"complex-polar"|"constant"

## allow degree in bvar
degree = element {degree} {MathML.Common.attrib,ContExp}
logbase = element {logbase} {MathML.Common.attrib,ContExp}
momentabout = element {momentabout} {MathML.Common.attrib,ContExp}
bvar-head |= (degree?,ci)|(ci,degree?)

## allow degree to modify <root/>
apply.content |= root_arith1_elt,degree,ContExp*
apply.content |= moment_s_data1_elt,(degree? & momentabout?),ContInPres*
apply.content |= log_transc1_elt,logbase,ContExp*

##allow apply to act as a binder
```

```
apply.content |= bind.content

domainofapplication = element {domainofapplication} {Definition.attrib,MathML.Common.attrib

lowlimit = element {lowlimit} {Definition.attrib,MathML.Common.attrib,cdbase.attrib?,ContEx
uplimit = element {uplimit} {Definition.attrib,MathML.Common.attrib,cdbase.attrib?,ContExp+

condition = element {condition} {Definition.attrib,cdbase.attrib?,ContExp}

## allow the non-strict qualifiers
qualifier |= domainofapplication|(uplimit,lowlimit?)|(lowlimit,uplimit?)|degree|condition

## we collect the operator elements by role
opel.constant = notAllowed
opel.binder = notAllowed
opel.application = notAllowed
opel.semantic-attribution = notAllowed
opel.attribution = notAllowed
opel.error = notAllowed

opels = opel.constant | opel.binder | opel.application |
        opel.semantic-attribution | opel.attribution |
opel.error
container = notAllowed

## the values of the MathML type attributes;
MathMLType |= "real" | "complex" | "function" | "algebraic" | "integer"


## we instantiate the strict content model by structure checking
apply-binder-head = semantics-apply-binder|opel.binder
apply.content |= apply-binder-head,bvar*,qualifier?,ContExp*
semantics-apply-binder = element {semantics} {semantics.attribs,apply-binder-head, semantic

apply-head |= opel.application
bind-head |= opel.binder
cerror-head |= opel.error

## allow all functions, constants, and containers to be content expressions on their own
ContExp |= opel.constant|opel.application|container


# allow no body
bind.content |= bind-head,bvar*,qualifier?

# not sure what a sequence of things is supposed to map to in strict/OM
# but is definitely allowed in pragmatic
# see Content/SequencesAndSeries/product/rec-product3
math.content |= ContExp*
```

```
opel.content |= PresExp|Glyph-alignmark
```

This grammar focuses on the pragmatic extensions in , , , and .

Editor's note: MiKocheck this again

The pragmatic extensions in , , and rely on information that is specified in the MathML content dictionaries. This is handled in the schema http://www.w3.org/Math/RelaxNG/mathml3/mathml3-cds-pragmatic.rnc.

Editor's note: MiKoThe generated grammar allows `type` attributes for the operator elements, this is incorrect

Finally, the pragmatic extensions given in are not covered in this schema, but will be left for full MathML in the next section.

A.2.5 Deprecated Features

The grammar for the deprecated features in MathML3 can be found at http://www.w3.org/Math/RelaxNG/mathml3/mathml3-deprecated.rnc.

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2008 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
#
#
#     Revision:   $Id: mathml3-deprecated.rnc,v 1.9 2008/12/17 09:10:34 mkohlhas2 Exp $
#
#    Update to MathML3 and Relax NG: David Carlisle and Michael Kohlhase

default namespace m = "http://www.w3.org/1998/Math/MathML"


Token-style.attrib &=
  attribute fontsize {xsd:string}? ,
  attribute fontstyle {xsd:string}? ,
  attribute fontweight {xsd:string}? ,
  attribute color {xsd:string}? ,
  attribute fontfamily {xsd:string}?

#Deprecated Content Elements
dep-content =
  element {reln} {ContExp*}|
  element {fn} {ContExp}
```

```
ContExp |= dep-content

apply-head |= dep-content

declare = element {declare} {attribute type {xsd:string}?,
                             attribute scope {xsd:string}?,
                             attribute nargs {xsd:nonNegativeInteger}?,
                             attribute occurrence {"prefix"|"infix"|"function-model"}?,
                             Definition.attrib,cdbase.attrib?,
                             ContExp+}
ContExp |= declare

mtr.content |= ContInPres
```

A.2.6 MathML as a module in a RelaxNG Schema

Normally, a MathML expression does not constitute an entire XML document. MathML is designed to be used as the mathematics fragment of larger markup languages. In particular it is designed to be used as a *module* in documents marked up with the XHTML family of markup languages. As RelaxNG directly supports modular development, this is usually very easy: an XHTML+MathML schema can be specified as simply as

```
# A RelaxNG Schema for  XHTML+MathML
include "xhtml.rnc"
math = external "mathml3.rnc"
Inline.class |= math
Block.class |= math
```

assuming that we have access to a modular RelaxNG schema for xhtml that uses `Inline.class` and `Block.class` to collect the the content models for inline and block-level elements.

Editor's note: Mikocheck this and reference an external schema

Specilizing the MathML3 schema so that we can check the content of `annotation-xml` elements is similarly simple:

```
# A RelaxNG Schema for MathML with OpenMath3 annotations
omobj = external "openmath3.rnc"
include "mathml3.rnc" {anotation-xml.model = omobj}
```

For details about RelaxNG grammars and modularization see [RelaxNG] or [RelaxNGBook].

Editor's note: Mikocheck this and reference an external schema; I think we can even tie the OpenMath model to the value `OpenMath` in the `encoding` attribute.

A.3 Using the MathML DTD

Editor's note: DavidDTD to be generated from Relax NG

Editor's note: BruceI've moved DTD related material from Chapter 2 to here. It most likely needs to be pruned somewhat

A.3.1 Document Validation Issues

The use of namespace prefixes creates an issue for DTD validation of documents embedding MathML. DTD validation requires knowing the literal (possibly prefixed) element names used in the document. However, the Namespaces in XML Recommendation [Namespaces] allows the prefix to be changed at arbitrary points in the document, since namespace prefixes may be declared on any element.

The 'historical' method of bridging this gap was to write a DTD with a fixed prefix, or in the case of XHTML and MathML, with no prefix, and mandate that the specified form must be used throughout the document. However, this is somewhat restricting for a modular DTD that is intended for use in conjunction with another DTD, which is exactly the situation with MathML in XHTML. In essence, the MathML DTD would have to allocate a prefix for itself and hope no other module uses the same prefix to avoid name clashes, thus losing one of the main benefits of XML namespaces.

One strategy for addressing this problem is to make every element name in the DTD be accessed by an entity reference. This means that by declaring a couple of entities to specify the prefix before the DTD is loaded, the prefix can be chosen by a document author, and compound DTDs that include several modules can, without changing the module DTDs, specify unique prefixes for each module to avoid clashes. The MathML DTD has been designed in this fashion. See Section A.3 and [Modularization] for details.

An extra issue arises in the case where explicit prefixes are used on the top-level `math` element, but a default namespace is used for other MathML elements. In this case, one wants the MathML module to be included into XHTML with the prefix set to empty. However, the 'driver' DTD file that sets up the inclusion of the MathML module would then need to define a new element called m:math. This would allow the top-level `math` element to use an explicit prefix, for attaching rendering behaviors in current browsers, while the contents would not need an explicit prefix, for ease of interoperability between authoring tools, etc.

A.3.2 Attribute values in the MathML DTD

In an XML DTD, allowed attribute values can be declared as general strings, or they can be constrained in various ways, either by enumerating the possible values, or by declaring them to be certain special data types. The choice of an XML attribute type affects the extent to which validity checks can be performed using a DTD.

The MathML DTD specifies formal XML attribute types for all MathML attributes, including enumerations of legitimate values in some cases. In general, however, the MathML DTD is relatively permissive, frequently declaring attribute values as strings; this is done to provide for interoperability with SGML parsers while allowing multiple attributes on one MathML element to accept the same values (such as `"true"` and `"false"`), and also to allow extension to the lists of predefined values.

At the same time, even though an attribute value may be declared as a string in the DTD, only certain values are legitimate in MathML, as described above and in the rest of this specification. For example, many attributes expect numerical values. In the sections which follow, the allowed attribute values are described for each element. To determine when these constraints are actually enforced in the MathML DTD, consult Appendix A. However, lack of enforcement of a requirement in the DTD does *not* imply that the requirement is not part of the MathML language itself, or that it will not be enforced by a particular MathML renderer. (See Section 2.3.2 for a description of how MathML renderers should respond to MathML errors.)

Furthermore, the MathML DTD is provided for convenience; although it is intended to be fully compatible with the text of the specification, the text should be taken as definitive if there is a contradiction.

(Any contradictions which may exist between various chapters of the text should be resolved by favoring Chapter 7 first, then Chapter 3, Chapter 4, then Section 2.1, and then other parts of the text.) For the MathML schema the situation will be the same: the published Recommendation text takes precedence. Though this is what is intended to happen, there is a practical difficulty. If the system processing the MathML uses a validating parser, whether it be based on a DTD or on a schema, the process will probably simply stop when it hits something held to be incorrect syntax, whether or not further MathML processing in full harmony with the specification would have processed the piece correctly.

A.4 Using the MathML XML Schema

Editor's note: DavidXSD schema to be generated from Relax NG

Appendix B

Media Types Registrations

The registrations of this normative appendix will be submitted to the IESG for review, approval, and registration with IANA. This normative appendix specifies the detailed information about the media-types according to [RFC4288] and their interoperability requirements. A more textual explanation is in Section 6.2.3

B.1 Media type for Generic MathML

This registration is for community review and will be submitted to the IESG for review, approval, and registration with IANA.

Type name `application`
Subtype name `mathml+xml`
Required parameters none
Optional parameters `charset` as per [RFC3023]
Encoding considerations same as XML as specified in [RFC3023].
Security considerations none
Interoperability considerations entities with this media type also have the media type `application/xml` and may also have the media types `application/presentation-mathml+xml`or `application/content-mathml+xml`.
Published specification this specification, MathML-3
Applications that use this media type (todo)
Additional information Magic number(s): see [RFC3023]
File extension(s): `.mml`
Windows Clipboard Name: `MathML`
Macintosh file type code(s) MML
Macintosh Universal Type Identifier code `public.mathml` conforming to `public.xml`
Person & email address to contact for further information World Wide Web Consortium web-human@w3.org
Intended usage COMMON
Restrictions on usage None
Author and Change controller The MathML specification is the product of the World Wide Web Consortium's Math Working Group. The W3C has change control over this specification.

B.2 Media type for Presentation MathML

This registration is for community review and will be submitted to the IESG for review, approval, and registration with IANA.

Type name `application`
Subtype name `mathml-presentation+xml`
Required parameters none
Optional parameters `charset` as per [RFC3023]
Encoding considerations same as XML as specified in [RFC3023].
Security considerations none
Interoperability considerations This media type is a more specific type than `application/mathml+xml`. Hence any entity of this type is also of the type `application/mathml+xml` as well as `application/xml`. An agent offering this specific media type MUST also offer the `application/mathml+xml` media type. An agent delivering an entity of this media type SHOULD only do so if it knows the receiving party supports it, for example thanks to the `Accept` http header, and should otherwise deliver the more generic type.
Published specification this specification, MathML-3
Applications that use this media type (todo)
Additional information Magic number(s): see [RFC3023]
File extension(s): none
Windows Clipboard Name: `MathML Presentation`
Macintosh file type code(s) MMLp
Macintosh Universal Type Identifier code `public.mathml.presentation` conforming to `public.mathml` (described above) conforming to `public.xml`
Person & email address to contact for further information World Wide Web Consortium web-human@w3.org
Intended usage COMMON
Restrictions on usage None
Author and Change controller The MathML specification is the product of the World Wide Web Consortium's Math Working Group. The W3C has change control over this specification.

B.3 Media type for Content MathML

This registration is for community review and will be submitted to the IESG for review, approval, and registration with IANA.

Type name `application`
Subtype name `mathml-content+xml`
Required parameters none
Optional parameters `charset` as per [RFC3023]
Encoding considerations same as XML as specified in [RFC3023].
Security considerations none
Interoperability considerations This media type is a more specific type than `application/mathml+xml`. Hence any entity of this type is also of the type `application/mathml+xml` as well as `application/xml`. An agent offering this specific media type MUST also offer the `application/mathml+xml` media type. An agent delivering an entity of this media type SHOULD only do so if it knows the receiving party supports it, for example thanks to the `Accept` http header, and should otherwise deliver the more generic type.
Published specification this specification, MathML-3
Applications that use this media type (todo)
Additional information Magic number(s): see [RFC3023]
File extension(s): none

Windows Clipboard Name: `MathML Content`
Macintosh file type code(s) MMLc
Macintosh Universal Type Identifier code `public.mathml.content` conforming to `public.mathml` (described above) conforming to `public.xml`
Person & email address to contact for further information World Wide Web Consortium web-human@w3.org
Intended usage COMMON
Restrictions on usage None
Author and Change controller The MathML specification is the product of the World Wide Web Consortium's Math Working Group. The W3C has change control over this specification.

Appendix C

Operator Dictionary (Non-Normative)

The following table gives the suggested dictionary of rendering properties for operators, fences, separators, and accents in MathML, all of which are represented by `mo` elements. For brevity, all such elements will be called simply 'operators' in this Appendix.

C.1 Indexing of the operator dictionary

Note that the dictionary is indexed not just by the element content, but by the element content and `form` attribute value, together. Operators with more than one possible form have more than one entry. The MathML specification describes how the renderer chooses ('infers') which form to use when no `form` attribute is given; see Section 3.2.5.7.

Having made that choice, or with the `form` attribute explicitly specified in the `<mo>` element's start tag, the MathML renderer uses the remaining attributes from the dictionary entry for the appropriate single form of that operator, ignoring the entries for the other possible forms.

In the table below, all non-ASCII characters are represented by XML-style hexadecimal numeric character references. The choice of markup (character data, numeric character reference or named entity reference) for a given character in MathML has no effect on its rendering.

C.2 Format of operator dictionary entries

Each row of the table is indexed as described above by the both the character (given by codeoint and Unicode Name) and the value of the form attribute. The fourth column gives the `priority` which as described in Section 3.3.1), is significant for the proper grouping of sub-expressions using `<mrow>`. The rule described there is especially relevant to the automatic generation of MathML by conversion from other formats for displayed mathematics, such as TEX, which do not always specify how sub-expressions nest.

The length valued attribues such as `lspace` are given explictly in the following columns. Boolean valued attributes such as `stretch` are specified together in the final `Properties` column by listing the attribute name if its value should be set to `true` by the dictionary.

Any attribute not listed for some entry has its default value, which is given in parentheses in the table of attributes in Section 3.2.5.

((left parenthesis prefix 1 0 0 fence stretchy

could be expressed as an `mo` element by:

`<mo form="prefix" fence="true" stretchy="true" lspace="0em" rspace="0em"> ( </mo>`

(note the whitespace added around the content for readability; whitespace is optional in MathML).

This entry means that, for MathML renderers which use this suggested operator dictionary, giving the element `<mo form="prefix"> ( </mo>` alone, or simply `<mo> ( </mo>` in a position for which `form="prefix"` would be inferred (see below), is equivalent to giving the element with all attributes as shown above.

In some versions of this specification, the rows of the table may be reordered by clicking on a heading in the top row, to cause the table to be ordered by that column.

C.3 Notes on `lspace` and `rspace` attributes

The values for `lspace` and `rspace` given here range from 0 to 5, they are given numerically in order to save space in the table, the values should be taken as refering to the named mathematocal spaces, as follows.

Table Entry	Named Space	Default Length
0		0em
1	verythinmathspace	0.111111em
2	thinmathspace	0.166667em
3	mediummathspace	0.222222em
4	thickmathspace	0.277778em
5	verythickmathspace	0.333333em

For the invisible operators whose content is `&InvisibleTimes;` or `&ApplyFunction;`, it is suggested that MathML renderers choose spacing in a context-sensitive way (which is an exception to the static values given in the following table). For `<mo>&ApplyFunction;</mo>`, the total spacing (`"lspace"+ "rspace"`) in expressions such as 'sin x' (where the right operand doesn't start with a fence) should be greater than zero; for `<mo>&InvisibleTimes;</mo>`, the total spacing should be greater than zero when both operands (or the nearest tokens on either side, if on the baseline) are identifiers displayed in a non-slanted font (i.e. under the suggested rules, when both operands are multi-character identifiers).

Some renderers may wish to use no spacing for most operators appearing in scripts (i.e. when `scriptlevel` is greater than 0; see Section 3.3.4), as is the case in TEX.

C.4 Operator dictionary entries

Character	form	priority	lspace	rspace	minsize	Properties
⨿ amalgamation or coproduct	infix	1	4	4		
⟨ mathematical left angle bracket	prefix	2	0	0		fence stretchy
─ box drawings light horizontal	infix	4	0	0	0	stretchy
❘ light vertical bar	infix	7	4	4		stretchy
-> multiple character operator: ->	infix	8	4	4		
// multiple character operator: //	infix	8	4	4		
; semicolon	infix	10	0	3		separator

Character	form	priority	lspace	rspace	minsize	Properties
‖ multiple character operator: ‖	infix	10	3	3		
&& multiple character operator: &&	infix	12	4	4		
** multiple character operator: **	infix	12	1	1		
<= multiple character operator: <=	infix	12	4	4		
<> multiple character operator: <>	infix	12	1	1		
‘ left single quotation mark	prefix	12	0	0		fence
“ left double quotation mark	prefix	12	0	0		fence
⊏̸ square image of with slash	infix	15	4	4		
⊐̸ square original of with slash	infix	15	4	4		
≂̸ minus tilde with slash	infix	16	4	4		
≎̸ geometrically equivalent to with slash	infix	16	4	4		
≏̸ difference between with slash	infix	16	4	4		
≦̸ less-than over equal to with slash	infix	16	4	4		
≪̸ much less than with slash	infix	16	4	4		
≫̸ much greater than with slash	infix	16	4	4		
≿̸ succeeds or equivalent to with slash	infix	16	4	4		
⊂⃒ subset of with vertical line	infix	16	4	4		
⊃⃒ superset of with vertical line	infix	16	4	4		
⟦ mathematical left white square bracket	prefix	16	0	0		fence stretchy
⟵ long leftwards arrow	infix	16	1	1		stretchy
⟶ long rightwards arrow	infix	16	1	1		stretchy
⟷ long left right arrow	infix	16	1	1		stretchy
⟸ long leftwards double arrow	infix	16	1	1		stretchy
⟹ long rightwards double arrow	infix	16	1	1		stretchy
⟺ long left right double arrow	infix	16	1	1		stretchy
⧏̸ left triangle beside vertical bar with slash	infix	16	4	4		
⧐̸ vertical bar beside right triangle with slash	infix	16	4	4		
⩽̸ less-than or slanted equal to with slash	infix	16	4	4		
⩾̸ greater-than or slanted equal to with slash	infix	16	4	4		
⪡̸ double nested less-than with slash	infix	16	4	4		

Character	form	priority	lspace	rspace	minsize	Properties
⪢̸ double nested greater-than with slash	infix	16	4	4		
⪯̸ precedes above single-line equals sign with slash	infix	16	4	4		
⪰̸ succeeds above single-line equals sign with slash	infix	16	4	4		
>= multiple character operator: >=	infix	17	4	4		
@ commercial at	infix	25	1	1		
? question mark	infix	28	1	1		
_ low line	infix	29	1	1		
, comma	infix	30	0	3		separator
⁣ invisible separator	infix	30	0	0		separator
∴ therefore	infix	70	5	5		
∵ because	infix	70	5	5		
.. multiple character operator: ..	postfix	100	0	0		
... multiple character operator: ...	postfix	100	0	0		
: colon	infix	100	1	2		
… horizontal ellipsis	infix	150	0	0		
⋮ vertical ellipsis	infix	150	5	5		
⋯ midline horizontal ellipsis	infix	150	0	0		
⋱ down right diagonal ellipsis	infix	150	5	5		
∋ contains as member	infix	160	5	5		
⊢ right tack	infix	170	5	5		
⊣ left tack	infix	170	5	5		
⊤ down tack	infix	170	5	5		
⊨ true	infix	170	5	5		
⊩ forces	infix	170	5	5		
⊬ does not prove	infix	170	5	5		
⊭ not true	infix	170	5	5		
⊮ does not force	infix	170	5	5		
⊯ negated double vertical bar double right turnstile	infix	170	5	5		
∨ logical or	infix	190	4	4		
’ right single quotation mark	postfix	200	0	0		fence

Character	form	priority	lspace	rspace	minsize	Properties
” right double quotation mark	postfix	200	0	0		fence
∧ logical and	infix	200	4	4		
∀ for all	prefix	230	2	1		
∃ there exists	prefix	230	2	1		
∄ there does not exist	prefix	230	2	1		
∁ complement	infix	240	1	2		
∈ element of	infix	240	5	5		
∉ not an element of	infix	240	5	5		
∌ does not contain as member	infix	240	5	5		
⊂ subset of	infix	240	5	5		
⊃ superset of	infix	240	5	5		
⊄ not a subset of	infix	240	5	5		
⊅ not a superset of	infix	240	5	5		
⊆ subset of or equal to	infix	240	5	5		
⊇ superset of or equal to	infix	240	5	5		
⊈ neither a subset of nor equal to	infix	240	5	5		
⊉ neither a superset of nor equal to	infix	240	5	5		
⊊ subset of with not equal to	infix	240	5	5		
⊋ superset of with not equal to	infix	240	5	5		
≤ less-than or equal to	infix	241	5	5		
≥ greater-than or equal to	infix	242	5	5		
>; greater-than sign	infix	243	5	5		
≯ not greater-than	infix	244	5	5		
< less-than sign	infix	245	5	5		
≮ not less-than	infix	246	5	5		
≈ almost equal to	infix	247	5	5		
∼ tilde operator	infix	250	5	5		
≉ not almost equal to	infix	250	5	5		
≢ not identical to	infix	252	5	5		
≠ not equal to	infix	255	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
!= multiple character operator: !=	infix	260	4	4		
*= multiple character operator: *=	infix	260	4	4		
+= multiple character operator: +=	infix	260	4	4		
-= multiple character operator: -=	infix	260	4	4		
/= multiple character operator: /=	infix	260	4	4		
:= multiple character operator: :=	infix	260	4	4		
= equals sign	infix	260	5	5		
== multiple character operator: ==	infix	260	4	4		
∝ proportional to	infix	260	5	5		
∤ does not divide	infix	260	5	5		
∥ parallel to	infix	260	5	5		
∦ not parallel to	infix	260	5	5		
≁ not tilde	infix	260	5	5		
≃ asymptotically equal to	infix	260	5	5		
≄ not asymptotically equal to	infix	260	5	5		
≅ approximately equal to	infix	260	5	5		
≆ approximately but not actually equal to	infix	260	5	5		
≇ neither approximately nor actually equal to	infix	260	5	5		
≍ equivalent to	infix	260	5	5		
≔ colon equals	infix	260	5	5		
≗ ring equal to	infix	260	5	5		
≙ estimates	infix	260	5	5		
≚ equiangular to	infix	260	5	5		
≜ delta equal to	infix	260	5	5		
≟ questioned equal to	infix	260	5	5		
≡ identical to	infix	260	5	5		
≨ less-than but not equal to	infix	260	5	5		
≩ greater-than but not equal to	infix	260	5	5		
≪ much less-than	infix	260	5	5		
≫ much greater-than	infix	260	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
≭ not equivalent to	infix	260	5	5		
≰ neither less-than nor equal to	infix	260	5	5		
≱ neither greater-than nor equal to	infix	260	5	5		
≺ precedes	infix	260	5	5		
≻ succeeds	infix	260	5	5		
≼ precedes or equal to	infix	260	5	5		
≽ succeeds or equal to	infix	260	5	5		
⊀ does not precede	infix	260	5	5		
⊁ does not succeed	infix	260	5	5		
⊥ up tack	infix	260	5	5		
⊴ normal subgroup of or equal to	infix	260	5	5		
⊵ contains as normal subgroup or equal to	infix	260	5	5		
⋉ left normal factor semidirect product	infix	260	4	4		
⋊ right normal factor semidirect product	infix	260	4	4		
⋋ left semidirect product	infix	260	4	4		
⋌ right semidirect product	infix	260	4	4		
⋔ pitchfork	infix	260	5	5		
⋖ less-than with dot	infix	260	5	5		
⋗ greater-than with dot	infix	260	5	5		
⋘ very much less-than	infix	260	5	5		
⋙ very much greater-than	infix	260	5	5		
⋪ not normal subgroup of	infix	260	5	5		
⋫ does not contain as normal subgroup	infix	260	5	5		
⋬ not normal subgroup of or equal to	infix	260	5	5		
⋭ does not contain as normal subgroup or equal	infix	260	5	5		
■ black square	infix	260	3	3		
□ white square	infix	260	3	3		
▪ black small square	infix	260	3	3		
▫ white small square	infix	260	3	3		
▭ white rectangle	infix	260	3	3		

Character	form	priority	lspace	rspace	minsize	Properties
▮ black vertical rectangle	infix	260	3	3		
▯ white vertical rectangle	infix	260	3	3		
▰ black parallelogram	infix	260	3	3		
▱ white parallelogram	infix	260	3	3		
△ white up-pointing triangle	infix	260	4	4		
▴ black up-pointing small triangle	infix	260	4	4		
▵ white up-pointing small triangle	infix	260	4	4		
▶ black right-pointing triangle	infix	260	4	4		
▷ white right-pointing triangle	infix	260	4	4		
▸ black right-pointing small triangle	infix	260	4	4		
▹ white right-pointing small triangle	infix	260	4	4		
▼ black down-pointing triangle	infix	260	4	4		
▽ white down-pointing triangle	infix	260	4	4		
▾ black down-pointing small triangle	infix	260	4	4		
▿ white down-pointing small triangle	infix	260	4	4		
◀ black left-pointing triangle	infix	260	4	4		
◁ white left-pointing triangle	infix	260	4	4		
◂ black left-pointing small triangle	infix	260	4	4		
◃ white left-pointing small triangle	infix	260	4	4		
◄ black left-pointing pointer	infix	260	4	4		
◅ white left-pointing pointer	infix	260	4	4		
◆ black diamond	infix	260	4	4		
◇ white diamond	infix	260	4	4		
◈ white diamond containing black small diamond	infix	260	4	4		
◉ fisheye	infix	260	4	4		
◌ dotted circle	infix	260	4	4		
◍ circle with vertical fill	infix	260	4	4		
◎ bullseye	infix	260	4	4		
● black circle	infix	260	4	4		
◖ left half black circle	infix	260	4	4		

Character	form	priority	lspace	rspace	minsize	Properties
◗ right half black circle	infix	260	4	4		
◦ white bullet	infix	260	4	4		
⧀ circled less-than	infix	260	5	5		
⧁ circled greater-than	infix	260	5	5		
⧣ equals sign and slanted parallel	infix	260	5	5		
⧤ equals sign and slanted parallel with tilde above	infix	260	5	5		
⧥ identical to and slanted parallel	infix	260	5	5		
⧦ gleich stark	infix	260	5	5		
⧳ error-barred black circle	infix	260	3	3		
⪇ less-than and single-line not equal to	infix	260	5	5		
⪈ greater-than and single-line not equal to	infix	260	5	5		
⪯ precedes above single-line equals sign	infix	260	5	5		
⪰ succeeds above single-line equals sign	infix	260	5	5		
∆ increment	infix	265	3	3		
∊ small element of	infix	265	5	5		
∍ small contains as member	infix	265	5	5		
∎ end of proof	infix	265	3	3		
∕ division slash	infix	265	4	4		
∗ asterisk operator	infix	265	4	4		
∙ bullet operator	infix	265	4	4		
∟ right angle	infix	265	5	5		
∣ divides	infix	265	5	5		
∶ ratio	infix	265	5	5		
∷ proportion	infix	265	5	5		
∸ dot minus	infix	265	4	4		
∹ excess	infix	265	5	5		
∺ geometric proportion	infix	265	4	4		
∻ homothetic	infix	265	5	5		
∽ reversed tilde	infix	265	5	5		
∾ inverted lazy s	infix	265	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
∿ sine wave	infix	265	3	3		
≂ minus tilde	infix	265	5	5		
≊ almost equal or equal to	infix	265	5	5		
≋ triple tilde	infix	265	5	5		
≌ all equal to	infix	265	5	5		
≎ geometrically equivalent to	infix	265	5	5		
≏ difference between	infix	265	5	5		
≐ approaches the limit	infix	265	5	5		
≑ geometrically equal to	infix	265	5	5		
≒ approximately equal to or the image of	infix	265	5	5		
≓ image of or approximately equal to	infix	265	5	5		
≕ equals colon	infix	265	5	5		
≖ ring in equal to	infix	265	5	5		
≘ corresponds to	infix	265	5	5		
≝ equal to by definition	infix	265	5	5		
≞ measured by	infix	265	5	5		
≣ strictly equivalent to	infix	265	5	5		
≦ less-than over equal to	infix	265	5	5		
≧ greater-than over equal to	infix	265	5	5		
≬ between	infix	265	5	5		
≲ less-than or equivalent to	infix	265	5	5		
≳ greater-than or equivalent to	infix	265	5	5		
≴ neither less-than nor equivalent to	infix	265	5	5		
≵ neither greater-than nor equivalent to	infix	265	5	5		
≶ less-than or greater-than	infix	265	5	5		
≷ greater-than or less-than	infix	265	5	5		
≸ neither less-than nor greater-than	infix	265	5	5		
≹ neither greater-than nor less-than	infix	265	5	5		
≾ precedes or equivalent to	infix	265	5	5		
≿ succeeds or equivalent to	infix	265	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⊌ multiset	infix	265	4	4		
⊍ multiset multiplication	infix	265	4	4		
⊎ multiset union	infix	265	4	4		stretchy
⊏ square image of	infix	265	5	5		
⊐ square original of	infix	265	5	5		
⊑ square image of or equal to	infix	265	5	5		
⊒ square original of or equal to	infix	265	5	5		
⊓ square cap	infix	265	4	4		stretchy
⊔ square cup	infix	265	4	4		stretchy
⊚ circled ring operator	infix	265	4	4		
⊛ circled asterisk operator	infix	265	4	4		
⊜ circled equals	infix	265	4	4		
⊝ circled dash	infix	265	4	4		
⊦ assertion	infix	265	5	5		
⊧ models	infix	265	5	5		
⊪ triple vertical bar right turnstile	infix	265	5	5		
⊫ double vertical bar double right turnstile	infix	265	5	5		
⊰ precedes under relation	infix	265	5	5		
⊱ succeeds under relation	infix	265	5	5		
⊲ normal subgroup of	infix	265	5	5		
⊳ contains as normal subgroup	infix	265	5	5		
⊶ original of	infix	265	5	5		
⊷ image of	infix	265	5	5		
⊹ hermitian conjugate matrix	infix	265	5	5		
⊺ intercalate	infix	265	4	4		
⊻ xor	infix	265	4	4		
⊼ nand	infix	265	4	4		
⊽ nor	infix	265	4	4		
⊾ right angle with arc	infix	265	3	3		
⊿ right triangle	infix	265	3	3		

Character	form	priority	lspace	rspace	minsize	Properties
⋄ diamond operator	infix	265	4	4		
⋆ star operator	infix	265	4	4		
⋇ division times	infix	265	4	4		
⋈ bowtie	infix	265	5	5		
⋍ reversed tilde equals	infix	265	5	5		
⋎ curly logical or	infix	265	4	4		
⋏ curly logical and	infix	265	4	4		
⋐ double subset	infix	265	5	5		
⋑ double superset	infix	265	5	5		
⋒ double intersection	infix	265	4	4		
⋓ double union	infix	265	4	4		
⋕ equal and parallel to	infix	265	5	5		
⋚ less-than equal to or greater-than	infix	265	5	5		
⋛ greater-than equal to or less-than	infix	265	5	5		
⋜ equal to or less-than	infix	265	5	5		
⋝ equal to or greater-than	infix	265	5	5		
⋞ equal to or precedes	infix	265	5	5		
⋟ equal to or succeeds	infix	265	5	5		
⋠ does not precede or equal	infix	265	5	5		
⋡ does not succeed or equal	infix	265	5	5		
⋢ not square image of or equal to	infix	265	5	5		
⋣ not square original of or equal to	infix	265	5	5		
⋤ square image of or not equal to	infix	265	5	5		
⋥ square original of or not equal to	infix	265	5	5		
⋦ less-than but not equivalent to	infix	265	5	5		
⋧ greater-than but not equivalent to	infix	265	5	5		
⋨ precedes but not equivalent to	infix	265	5	5		
⋩ succeeds but not equivalent to	infix	265	5	5		
⋰ up right diagonal ellipsis	infix	265	5	5		
⋲ element of with long horizontal stroke	infix	265	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⋳ element of with vertical bar at end of horizontal stroke	infix	265	5	5		
⋴ small element of with vertical bar at end of horizontal stroke	infix	265	5	5		
⋵ element of with dot above	infix	265	5	5		
⋶ element of with overbar	infix	265	5	5		
⋷ small element of with overbar	infix	265	5	5		
⋸ element of with underbar	infix	265	5	5		
⋹ element of with two horizontal strokes	infix	265	5	5		
⋺ contains with long horizontal stroke	infix	265	5	5		
⋻ contains with vertical bar at end of horizontal stroke	infix	265	5	5		
⋼ small contains with vertical bar at end of horizontal stroke	infix	265	5	5		
⋽ contains with overbar	infix	265	5	5		
⋾ small contains with overbar	infix	265	5	5		
⋿ z notation bag membership	infix	265	5	5		
▲ black up-pointing triangle	infix	265	4	4		
⦁ z notation spot	infix	265	3	3		
⦂ z notation type colon	infix	265	3	3		
⦑ left angle bracket with dot	prefix	265	0	0		fence stretchy
⦒ right angle bracket with dot	postfix	265	0	0		fence stretchy
⦓ left arc less-than bracket	prefix	265	0	0		fence stretchy
⦔ right arc greater-than bracket	postfix	265	0	0		fence stretchy
⦕ double left arc greater-than bracket	prefix	265	0	0		fence stretchy
⦖ double right arc less-than bracket	postfix	265	0	0		fence stretchy
⦠ spherical angle opening left	infix	265	3	3		
⦡ spherical angle opening up	infix	265	3	3		
⦢ turned angle	infix	265	3	3		
⦣ reversed angle	infix	265	3	3		
⦤ angle with underbar	infix	265	3	3		
⦥ reversed angle with underbar	infix	265	3	3		
⦦ oblique angle opening up	infix	265	3	3		
⦧ oblique angle opening down	infix	265	3	3		

Character	form	priority	lspace	rspace	minsize	Properties
⦨ measured angle with open arm ending in arrow pointing up and right	infix	265	3	3		
⦩ measured angle with open arm ending in arrow pointing up and left	infix	265	3	3		
⦪ measured angle with open arm ending in arrow pointing down and right	infix	265	3	3		
⦫ measured angle with open arm ending in arrow pointing down and left	infix	265	3	3		
⦬ measured angle with open arm ending in arrow pointing right and up	infix	265	3	3		
⦭ measured angle with open arm ending in arrow pointing left and up	infix	265	3	3		
⦮ measured angle with open arm ending in arrow pointing right and down	infix	265	3	3		
⦯ measured angle with open arm ending in arrow pointing left and down	infix	265	3	3		
⦰ reversed empty set	infix	265	3	3		
⦱ empty set with overbar	infix	265	3	3		
⦲ empty set with small circle above	infix	265	3	3		
⦳ empty set with right arrow above	infix	265	3	3		
⦴ empty set with left arrow above	infix	265	3	3		
⦵ circle with horizontal bar	infix	265	3	3		
⦶ circled vertical bar	infix	265	4	4		
⦷ circled parallel	infix	265	4	4		
⦸ circled reverse solidus	infix	265	4	4		
⦹ circled perpendicular	infix	265	4	4		
⦺ circle divided by horizontal bar and top half divided by vertical bar	infix	265	4	4		
⦻ circle with superimposed x	infix	265	4	4		
⦼ circled anticlockwise-rotated division sign	infix	265	4	4		
⦽ up arrow through circle	infix	265	4	4		
⦾ circled white bullet	infix	265	4	4		
⦿ circled bullet	infix	265	4	4		
⧂ circle with small circle to the right	infix	265	3	3		
⧃ circle with two horizontal strokes to the right	infix	265	3	3		
⧄ squared rising diagonal slash	infix	265	4	4		
⧅ squared falling diagonal slash	infix	265	4	4		
⧆ squared asterisk	infix	265	4	4		
⧇ squared small circle	infix	265	4	4		

Character	form	priority	lspace	rspace	minsize	Properties
⧈ squared square	infix	265	4	4		
⧉ two joined squares	infix	265	3	3		
⧊ triangle with dot above	infix	265	3	3		
⧋ triangle with underbar	infix	265	3	3		
⧌ s in triangle	infix	265	3	3		
⧍ triangle with serifs at bottom	infix	265	3	3		
⧎ right triangle above left triangle	infix	265	5	5		
⧏ left triangle beside vertical bar	infix	265	5	5		
⧐ vertical bar beside right triangle	infix	265	5	5		
⧑ bowtie with left half black	infix	265	5	5		
⧒ bowtie with right half black	infix	265	5	5		
⧓ black bowtie	infix	265	5	5		
⧔ times with left half black	infix	265	5	5		
⧕ times with right half black	infix	265	5	5		
⧖ white hourglass	infix	265	4	4		
⧗ black hourglass	infix	265	4	4		
⧘ left wiggly fence	infix	265	3	3		
⧙ right wiggly fence	infix	265	3	3		
⧚ left double wiggly fence	infix	265	3	3		
⧛ right double wiggly fence	infix	265	3	3		
⧜ incomplete infinity	infix	265	3	3		
⧝ tie over infinity	infix	265	3	3		
⧞ infinity negated with vertical bar	infix	265	5	5		
⧠ square with contoured outline	infix	265	3	3		
⧡ increases as	infix	265	5	5		
⧢ shuffle product	infix	265	4	4		
⧧ thermodynamic	infix	265	3	3		
⧨ down-pointing triangle with left half black	infix	265	3	3		
⧩ down-pointing triangle with right half black	infix	265	3	3		
⧪ black diamond with down arrow	infix	265	3	3		

Character	form	priority	lspace	rspace	minsize	Properties
⧫ black lozenge	infix	265	3	3		
⧬ white circle with down arrow	infix	265	3	3		
⧭ black circle with down arrow	infix	265	3	3		
⧮ error-barred white square	infix	265	3	3		
⧰ error-barred white diamond	infix	265	3	3		
⧱ error-barred black diamond	infix	265	3	3		
⧲ error-barred white circle	infix	265	3	3		
⧵ reverse solidus operator	infix	265	4	4		
⧶ solidus with overbar	infix	265	4	4		
⧷ reverse solidus with horizontal stroke	infix	265	4	4		
⧸ big solidus	infix	265	3	3		
⧹ big reverse solidus	infix	265	3	3		
⧺ double plus	infix	265	3	3		
⧻ triple plus	infix	265	3	3		
⧾ tiny	infix	265	4	4		
⧿ miny	infix	265	4	4		
⨝ join	infix	265	3	3		
⨞ large left triangle operator	infix	265	3	3		
⨟ z notation schema composition	infix	265	3	3		
⨠ z notation schema piping	infix	265	3	3		
⨡ z notation schema projection	infix	265	3	3		
⨢ plus sign with small circle above	infix	265	4	4		
⨣ plus sign with circumflex accent above	infix	265	4	4		
⨤ plus sign with tilde above	infix	265	4	4		
⨥ plus sign with dot below	infix	265	4	4		
⨦ plus sign with tilde below	infix	265	4	4		
⨧ plus sign with subscript two	infix	265	4	4		
⨨ plus sign with black triangle	infix	265	4	4		
⨩ minus sign with comma above	infix	265	4	4		
⨪ minus sign with dot below	infix	265	4	4		

Character	form	priority	lspace	rspace	minsize	Properties
⨫ minus sign with falling dots	infix	265	4	4		
⨬ minus sign with rising dots	infix	265	4	4		
⨭ plus sign in left half circle	infix	265	4	4		
⨮ plus sign in right half circle	infix	265	4	4		
⨰ multiplication sign with dot above	infix	265	4	4		
⨱ multiplication sign with underbar	infix	265	4	4		
⨲ semidirect product with bottom closed	infix	265	4	4		
⨳ smash product	infix	265	4	4		
⨴ multiplication sign in left half circle	infix	265	4	4		
⨵ multiplication sign in right half circle	infix	265	4	4		
⨶ circled multiplication sign with circumflex accent	infix	265	4	4		
⨷ multiplication sign in double circle	infix	265	4	4		
⨸ circled division sign	infix	265	4	4		
⨹ plus sign in triangle	infix	265	4	4		
⨺ minus sign in triangle	infix	265	4	4		
⨻ multiplication sign in triangle	infix	265	4	4		
⨼ interior product	infix	265	4	4		
⨽ righthand interior product	infix	265	4	4		
⨾ z notation relational composition	infix	265	4	4		
⩀ intersection with dot	infix	265	4	4		
⩁ union with minus sign	infix	265	4	4		
⩂ union with overbar	infix	265	4	4		
⩃ intersection with overbar	infix	265	4	4		
⩄ intersection with logical and	infix	265	4	4		
⩅ union with logical or	infix	265	4	4		
⩆ union above intersection	infix	265	4	4		
⩇ intersection above union	infix	265	4	4		
⩈ union above bar above intersection	infix	265	4	4		
⩉ intersection above bar above union	infix	265	4	4		
⩊ union beside and joined with union	infix	265	4	4		

Character	form	priority	lspace	rspace	minsize	Properties
⩋ intersection beside and joined with intersection	infix	265	4	4		
⩌ closed union with serifs	infix	265	4	4		
⩍ closed intersection with serifs	infix	265	4	4		
⩎ double square intersection	infix	265	4	4		
⩏ double square union	infix	265	4	4		
⩐ closed union with serifs and smash product	infix	265	4	4		
⩑ logical and with dot above	infix	265	4	4		
⩒ logical or with dot above	infix	265	4	4		
⩓ double logical and	infix	265	4	4		stretchy
⩔ double logical or	infix	265	4	4		stretchy
⩕ two intersecting logical and	infix	265	4	4		
⩖ two intersecting logical or	infix	265	4	4		
⩗ sloping large or	infix	265	4	4		
⩘ sloping large and	infix	265	4	4		
⩙ logical or overlapping logical and	infix	265	5	5		
⩚ logical and with middle stem	infix	265	4	4		
⩛ logical or with middle stem	infix	265	4	4		
⩜ logical and with horizontal dash	infix	265	4	4		
⩝ logical or with horizontal dash	infix	265	4	4		
⩞ logical and with double overbar	infix	265	4	4		
⩟ logical and with underbar	infix	265	4	4		
⩠ logical and with double underbar	infix	265	4	4		
⩡ small vee with underbar	infix	265	4	4		
⩢ logical or with double overbar	infix	265	4	4		
⩣ logical or with double underbar	infix	265	4	4		
⩤ z notation domain antirestriction	infix	265	4	4		
⩥ z notation range antirestriction	infix	265	4	4		
⩦ equals sign with dot below	infix	265	5	5		
⩧ identical with dot above	infix	265	5	5		
⩨ triple horizontal bar with double vertical stroke	infix	265	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⩩ triple horizontal bar with triple vertical stroke	infix	265	5	5		
⩪ tilde operator with dot above	infix	265	5	5		
⩫ tilde operator with rising dots	infix	265	5	5		
⩬ similar minus similar	infix	265	5	5		
⩭ congruent with dot above	infix	265	5	5		
⩮ equals with asterisk	infix	265	5	5		
⩯ almost equal to with circumflex accent	infix	265	5	5		
⩰ approximately equal or equal to	infix	265	5	5		
⩱ equals sign above plus sign	infix	265	4	4		
⩲ plus sign above equals sign	infix	265	4	4		
⩳ equals sign above tilde operator	infix	265	5	5		
⩴ double colon equal	infix	265	5	5		
⩵ two consecutive equals signs	infix	265	5	5		
⩶ three consecutive equals signs	infix	265	5	5		
⩷ equals sign with two dots above and two dots below	infix	265	5	5		
⩸ equivalent with four dots above	infix	265	5	5		
⩹ less-than with circle inside	infix	265	5	5		
⩺ greater-than with circle inside	infix	265	5	5		
⩻ less-than with question mark above	infix	265	5	5		
⩼ greater-than with question mark above	infix	265	5	5		
⩽ less-than or slanted equal to	infix	265	5	5		
⩾ greater-than or slanted equal to	infix	265	5	5		
⩿ less-than or slanted equal to with dot inside	infix	265	5	5		
⪀ greater-than or slanted equal to with dot inside	infix	265	5	5		
⪁ less-than or slanted equal to with dot above	infix	265	5	5		
⪂ greater-than or slanted equal to with dot above	infix	265	5	5		
⪃ less-than or slanted equal to with dot above right	infix	265	5	5		
⪄ greater-than or slanted equal to with dot above left	infix	265	5	5		
⪅ less-than or approximate	infix	265	5	5		
⪆ greater-than or approximate	infix	265	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⪉ less-than and not approximate	infix	265	5	5		
⪊ greater-than and not approximate	infix	265	5	5		
⪋ less-than above double-line equal above greater-than	infix	265	5	5		
⪌ greater-than above double-line equal above less-than	infix	265	5	5		
⪍ less-than above similar or equal	infix	265	5	5		
⪎ greater-than above similar or equal	infix	265	5	5		
⪏ less-than above similar above greater-than	infix	265	5	5		
⪐ greater-than above similar above less-than	infix	265	5	5		
⪑ less-than above greater-than above double-line equal	infix	265	5	5		
⪒ greater-than above less-than above double-line equal	infix	265	5	5		
⪓ less-than above slanted equal above greater-than above slanted equal	infix	265	5	5		
⪔ greater-than above slanted equal above less-than above slanted equal	infix	265	5	5		
⪕ slanted equal to or less-than	infix	265	5	5		
⪖ slanted equal to or greater-than	infix	265	5	5		
⪗ slanted equal to or less-than with dot inside	infix	265	5	5		
⪘ slanted equal to or greater-than with dot inside	infix	265	5	5		
⪙ double-line equal to or less-than	infix	265	5	5		
⪚ double-line equal to or greater-than	infix	265	5	5		
⪛ double-line slanted equal to or less-than	infix	265	5	5		
⪜ double-line slanted equal to or greater-than	infix	265	5	5		
⪝ similar or less-than	infix	265	5	5		
⪞ similar or greater-than	infix	265	5	5		
⪟ similar above less-than above equals sign	infix	265	5	5		
⪠ similar above greater-than above equals sign	infix	265	5	5		
⪡ double nested less-than	infix	265	5	5		
⪢ double nested greater-than	infix	265	5	5		
⪣ double nested less-than with underbar	infix	265	5	5		
⪤ greater-than overlapping less-than	infix	265	5	5		
⪥ greater-than beside less-than	infix	265	5	5		
⪦ less-than closed by curve	infix	265	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⪧ greater-than closed by curve	infix	265	5	5		
⪨ less-than closed by curve above slanted equal	infix	265	5	5		
⪩ greater-than closed by curve above slanted equal	infix	265	5	5		
⪪ smaller than	infix	265	5	5		
⪫ larger than	infix	265	5	5		
⪬ smaller than or equal to	infix	265	5	5		
⪭ larger than or equal to	infix	265	5	5		
⪮ equals sign with bumpy above	infix	265	5	5		
⪱ precedes above single-line not equal to	infix	265	5	5		
⪲ succeeds above single-line not equal to	infix	265	5	5		
⪳ precedes above equals sign	infix	265	5	5		
⪴ succeeds above equals sign	infix	265	5	5		
⪵ precedes above not equal to	infix	265	5	5		
⪶ succeeds above not equal to	infix	265	5	5		
⪷ precedes above almost equal to	infix	265	5	5		
⪸ succeeds above almost equal to	infix	265	5	5		
⪹ precedes above not almost equal to	infix	265	5	5		
⪺ succeeds above not almost equal to	infix	265	5	5		
⪻ double precedes	infix	265	5	5		
⪼ double succeeds	infix	265	5	5		
⪽ subset with dot	infix	265	5	5		
⪾ superset with dot	infix	265	5	5		
⪿ subset with plus sign below	infix	265	5	5		
⫀ superset with plus sign below	infix	265	5	5		
⫁ subset with multiplication sign below	infix	265	5	5		
⫂ superset with multiplication sign below	infix	265	5	5		
⫃ subset of or equal to with dot above	infix	265	5	5		
⫄ superset of or equal to with dot above	infix	265	5	5		
⫅ subset of above equals sign	infix	265	5	5		
⫆ superset of above equals sign	infix	265	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⫇ subset of above tilde operator	infix	265	5	5		
⫈ superset of above tilde operator	infix	265	5	5		
⫉ subset of above almost equal to	infix	265	5	5		
⫊ superset of above almost equal to	infix	265	5	5		
⫋ subset of above not equal to	infix	265	5	5		
⫌ superset of above not equal to	infix	265	5	5		
⫍ square left open box operator	infix	265	5	5		
⫎ square right open box operator	infix	265	5	5		
⫏ closed subset	infix	265	5	5		
⫐ closed superset	infix	265	5	5		
⫑ closed subset or equal to	infix	265	5	5		
⫒ closed superset or equal to	infix	265	5	5		
⫓ subset above superset	infix	265	5	5		
⫔ superset above subset	infix	265	5	5		
⫕ subset above subset	infix	265	5	5		
⫖ superset above superset	infix	265	5	5		
⫗ superset beside subset	infix	265	5	5		
⫘ superset beside and joined by dash with subset	infix	265	5	5		
⫙ element of opening downwards	infix	265	5	5		
⫚ pitchfork with tee top	infix	265	5	5		
⫛ transversal intersection	infix	265	5	5		
⫝̸ forking	infix	265	5	5		
⫝ nonforking	infix	265	5	5		
⫞ short left tack	infix	265	5	5		
⫟ short down tack	infix	265	5	5		
⫠ short up tack	infix	265	5	5		
⫡ perpendicular with s	infix	265	5	5		
⫢ vertical bar triple right turnstile	infix	265	5	5		
⫣ double vertical bar left turnstile	infix	265	5	5		
⫤ vertical bar double left turnstile	infix	265	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⫥ double vertical bar double left turnstile	infix	265	5	5		
⫦ long dash from left member of double vertical	infix	265	5	5		
⫧ short down tack with overbar	infix	265	5	5		
⫨ short up tack with underbar	infix	265	5	5		
⫩ short up tack above short down tack	infix	265	5	5		
⫪ double down tack	infix	265	5	5		
⫫ double up tack	infix	265	5	5		
⫬ double stroke not sign	infix	265	5	5		
⫭ reversed double stroke not sign	infix	265	5	5		
⫮ does not divide with reversed negation slash	infix	265	5	5		
⫯ vertical line with circle above	infix	265	5	5		
⫰ vertical line with circle below	infix	265	5	5		
⫱ down tack with circle below	infix	265	5	5		
⫲ parallel with horizontal stroke	infix	265	5	5		
⫳ parallel with tilde operator	infix	265	5	5		
⫴ triple vertical bar binary relation	infix	265	4	4		
⫵ triple vertical bar with horizontal stroke	infix	265	4	4		
⫶ triple colon operator	infix	265	4	4		
⫷ triple nested less-than	infix	265	5	5		
⫸ triple nested greater-than	infix	265	5	5		
⫹ double-line slanted less-than or equal to	infix	265	5	5		
⫺ double-line slanted greater-than or equal to	infix	265	5	5		
⫻ triple solidus binary relation	infix	265	4	4		
⫽ double solidus operator	infix	265	4	4		
⫾ white vertical bar	infix	265	3	3		
(left parenthesis	prefix	270	0	0		fence stretchy
) right parenthesis	postfix	270	0	0		fence stretchy
[left square bracket	prefix	270	0	0		fence stretchy
] right square bracket	postfix	270	0	0		fence stretchy
{ left curly bracket	prefix	270	0	0		fence stretchy

Character	form	priority	lspace	rspace	minsize	Properties
| vertical line	infix	270	2	2		fence stretchy
} right curly bracket	postfix	270	0	0		fence stretchy
→ rightwards arrow	infix	270	5	5		stretchy accent
↔ left right arrow	infix	270	5	5		stretchy accent
↖ north west arrow	infix	270	5	5		stretchy
↙ south west arrow	infix	270	5	5		stretchy
↚ leftwards arrow with stroke	infix	270	5	5		
↛ rightwards arrow with stroke	infix	270	5	5		
↜ leftwards wave arrow	infix	270	5	5		
↝ rightwards wave arrow	infix	270	5	5		
↞ leftwards two headed arrow	infix	270	5	5		
↟ upwards two headed arrow	infix	270	5	5		
↠ rightwards two headed arrow	infix	270	5	5		
↡ downwards two headed arrow	infix	270	5	5		
↢ leftwards arrow with tail	infix	270	5	5		
↣ rightwards arrow with tail	infix	270	5	5		
↤ leftwards arrow from bar	infix	270	5	5		stretchy
↥ upwards arrow from bar	infix	270	5	5		stretchy
↦ rightwards arrow from bar	infix	270	5	5		stretchy
↧ downwards arrow from bar	infix	270	5	5		stretchy
↨ up down arrow with base	infix	270	5	5		
↩ leftwards arrow with hook	infix	270	5	5		
↪ rightwards arrow with hook	infix	270	5	5		
↫ leftwards arrow with loop	infix	270	5	5		
↬ rightwards arrow with loop	infix	270	5	5		
↭ left right wave arrow	infix	270	5	5		
↮ left right arrow with stroke	infix	270	5	5		
↯ downwards zigzag arrow	infix	270	5	5		
↰ upwards arrow with tip leftwards	infix	270	5	5		
↱ upwards arrow with tip rightwards	infix	270	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
↲ downwards arrow with tip leftwards	infix	270	5	5		
↳ downwards arrow with tip rightwards	infix	270	5	5		
↴ rightwards arrow with corner downwards	infix	270	5	5		
↵ downwards arrow with corner leftwards	infix	270	5	5		
↶ anticlockwise top semicircle arrow	infix	270	5	5		
↷ clockwise top semicircle arrow	infix	270	5	5		
↸ north west arrow to long bar	infix	270	5	5		
↹ leftwards arrow to bar over rightwards arrow to bar	infix	270	5	5		
↺ anticlockwise open circle arrow	infix	270	5	5		
↻ clockwise open circle arrow	infix	270	5	5		
↼ leftwards harpoon with barb upwards	infix	270	5	5		stretchy accent
↽ leftwards harpoon with barb downwards	infix	270	5	5		stretchy
↾ upwards harpoon with barb rightwards	infix	270	5	5		stretchy
↿ upwards harpoon with barb leftwards	infix	270	5	5		stretchy
⇀ rightwards harpoon with barb upwards	infix	270	5	5		stretchy accent
⇁ rightwards harpoon with barb downwards	infix	270	5	5		stretchy
⇂ downwards harpoon with barb rightwards	infix	270	5	5		stretchy
⇃ downwards harpoon with barb leftwards	infix	270	5	5		stretchy
⇄ rightwards arrow over leftwards arrow	infix	270	5	5		stretchy
⇅ upwards arrow leftwards of downwards arrow	infix	270	5	5		stretchy
⇆ leftwards arrow over rightwards arrow	infix	270	5	5		stretchy
⇇ leftwards paired arrows	infix	270	5	5		
⇈ upwards paired arrows	infix	270	5	5		
⇉ rightwards paired arrows	infix	270	5	5		
⇊ downwards paired arrows	infix	270	5	5		
⇋ leftwards harpoon over rightwards harpoon	infix	270	5	5		stretchy
⇌ rightwards harpoon over leftwards harpoon	infix	270	5	5		stretchy
⇍ leftwards double arrow with stroke	infix	270	5	5		
⇎ left right double arrow with stroke	infix	270	5	5		
⇏ rightwards double arrow with stroke	infix	270	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⇐ leftwards double arrow	infix	270	5	5		stretchy
⇒ rightwards double arrow	infix	270	5	5		stretchy
⇔ left right double arrow	infix	270	5	5		stretchy
⇖ north west double arrow	infix	270	5	5		
⇗ north east double arrow	infix	270	5	5		
⇘ south east double arrow	infix	270	5	5		
⇙ south west double arrow	infix	270	5	5		
⇚ leftwards triple arrow	infix	270	5	5		
⇛ rightwards triple arrow	infix	270	5	5		
⇜ leftwards squiggle arrow	infix	270	5	5		
⇝ rightwards squiggle arrow	infix	270	5	5		
⇞ upwards arrow with double stroke	infix	270	5	5		
⇟ downwards arrow with double stroke	infix	270	5	5		
⇠ leftwards dashed arrow	infix	270	5	5		
⇡ upwards dashed arrow	infix	270	5	5		
⇢ rightwards dashed arrow	infix	270	5	5		
⇣ downwards dashed arrow	infix	270	5	5		
⇤ leftwards arrow to bar	infix	270	5	5		stretchy
⇥ rightwards arrow to bar	infix	270	5	5		stretchy
⇦ leftwards white arrow	infix	270	5	5		
⇧ upwards white arrow	infix	270	5	5		
⇨ rightwards white arrow	infix	270	5	5		
⇩ downwards white arrow	infix	270	5	5		
⇪ upwards white arrow from bar	infix	270	5	5		
⇫ upwards white arrow on pedestal	infix	270	5	5		
⇬ upwards white arrow on pedestal with horizontal bar	infix	270	5	5		
⇭ upwards white arrow on pedestal with vertical bar	infix	270	5	5		
⇮ upwards white double arrow	infix	270	5	5		
⇯ upwards white double arrow on pedestal	infix	270	5	5		
⇰ rightwards white arrow from wall	infix	270	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⇱ north west arrow to corner	infix	270	5	5		
⇲ south east arrow to corner	infix	270	5	5		
⇳ up down white arrow	infix	270	5	5		
⇴ right arrow with small circle	infix	270	5	5		
⇵ downwards arrow leftwards of upwards arrow	infix	270	5	5		stretchy
⇶ three rightwards arrows	infix	270	5	5		
⇷ leftwards arrow with vertical stroke	infix	270	5	5		
⇸ rightwards arrow with vertical stroke	infix	270	5	5		
⇹ left right arrow with vertical stroke	infix	270	5	5		
⇺ leftwards arrow with double vertical stroke	infix	270	5	5		
⇻ rightwards arrow with double vertical stroke	infix	270	5	5		
⇼ left right arrow with double vertical stroke	infix	270	5	5		
⇽ leftwards open-headed arrow	infix	270	5	5		
⇾ rightwards open-headed arrow	infix	270	5	5		
⇿ left right open-headed arrow	infix	270	5	5		
⊸ multimap	infix	270	5	5		
⌈ left ceiling	prefix	270	0	0		fence stretchy
⌉ right ceiling	postfix	270	0	0		fence stretchy
⌊ left floor	prefix	270	0	0		fence stretchy
⌋ right floor	postfix	270	0	0		fence stretchy
⟧ mathematical right white square bracket	postfix	270	0	0		fence stretchy
⟩ mathematical right angle bracket	postfix	270	0	0		fence stretchy
⤀ rightwards two-headed arrow with vertical stroke	infix	270	5	5		
⤁ rightwards two-headed arrow with double vertical stroke	infix	270	5	5		
⤂ leftwards double arrow with vertical stroke	infix	270	5	5		
⤃ rightwards double arrow with vertical stroke	infix	270	5	5		
⤄ left right double arrow with vertical stroke	infix	270	5	5		
⤅ rightwards two-headed arrow from bar	infix	270	5	5		
⤆ leftwards double arrow from bar	infix	270	5	5		
⤇ rightwards double arrow from bar	infix	270	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⤈ downwards arrow with horizontal stroke	infix	270	5	5		
⤉ upwards arrow with horizontal stroke	infix	270	5	5		
⤊ upwards triple arrow	infix	270	5	5		
⤋ downwards triple arrow	infix	270	5	5		
⤌ leftwards double dash arrow	infix	270	5	5		
⤍ rightwards double dash arrow	infix	270	5	5		
⤎ leftwards triple dash arrow	infix	270	5	5		
⤏ rightwards triple dash arrow	infix	270	5	5		
⤐ rightwards two-headed triple dash arrow	infix	270	5	5		
⤑ rightwards arrow with dotted stem	infix	270	5	5		
⤒ upwards arrow to bar	infix	270	5	5		stretchy
⤓ downwards arrow to bar	infix	270	5	5		stretchy
⤔ rightwards arrow with tail with vertical stroke	infix	270	5	5		
⤕ rightwards arrow with tail with double vertical stroke	infix	270	5	5		
⤖ rightwards two-headed arrow with tail	infix	270	5	5		
⤗ rightwards two-headed arrow with tail with vertical stroke	infix	270	5	5		
⤘ rightwards two-headed arrow with tail with double vertical stroke	infix	270	5	5		
⤙ leftwards arrow-tail	infix	270	5	5		
⤚ rightwards arrow-tail	infix	270	5	5		
⤛ leftwards double arrow-tail	infix	270	5	5		
⤜ rightwards double arrow-tail	infix	270	5	5		
⤝ leftwards arrow to black diamond	infix	270	5	5		
⤞ rightwards arrow to black diamond	infix	270	5	5		
⤟ leftwards arrow from bar to black diamond	infix	270	5	5		
⤠ rightwards arrow from bar to black diamond	infix	270	5	5		
⤡ north west and south east arrow	infix	270	5	5		
⤢ north east and south west arrow	infix	270	5	5		
⤣ north west arrow with hook	infix	270	5	5		
⤤ north east arrow with hook	infix	270	5	5		
⤥ south east arrow with hook	infix	270	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⤦ south west arrow with hook	infix	270	5	5		
⤧ north west arrow and north east arrow	infix	270	5	5		
⤨ north east arrow and south east arrow	infix	270	5	5		
⤩ south east arrow and south west arrow	infix	270	5	5		
⤪ south west arrow and north west arrow	infix	270	5	5		
⤫ rising diagonal crossing falling diagonal	infix	270	5	5		
⤬ falling diagonal crossing rising diagonal	infix	270	5	5		
⤭ south east arrow crossing north east arrow	infix	270	5	5		
⤮ north east arrow crossing south east arrow	infix	270	5	5		
⤯ falling diagonal crossing north east arrow	infix	270	5	5		
⤰ rising diagonal crossing south east arrow	infix	270	5	5		
⤱ north east arrow crossing north west arrow	infix	270	5	5		
⤲ north west arrow crossing north east arrow	infix	270	5	5		
⤳ wave arrow pointing directly right	infix	270	5	5		
⤴ arrow pointing rightwards then curving upwards	infix	270	5	5		
⤵ arrow pointing rightwards then curving downwards	infix	270	5	5		
⤶ arrow pointing downwards then curving leftwards	infix	270	5	5		
⤷ arrow pointing downwards then curving rightwards	infix	270	5	5		
⤸ right-side arc clockwise arrow	infix	270	5	5		
⤹ left-side arc anticlockwise arrow	infix	270	5	5		
⤺ top arc anticlockwise arrow	infix	270	5	5		
⤻ bottom arc anticlockwise arrow	infix	270	5	5		
⤼ top arc clockwise arrow with minus	infix	270	5	5		
⤽ top arc anticlockwise arrow with plus	infix	270	5	5		
⤾ lower right semicircular clockwise arrow	infix	270	5	5		
⤿ lower left semicircular anticlockwise arrow	infix	270	5	5		
⥀ anticlockwise closed circle arrow	infix	270	5	5		
⥁ clockwise closed circle arrow	infix	270	5	5		
⥂ rightwards arrow above short leftwards arrow	infix	270	5	5		
⥃ leftwards arrow above short rightwards arrow	infix	270	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⥄ short rightwards arrow above leftwards arrow	infix	270	5	5		
⥅ rightwards arrow with plus below	infix	270	5	5		
⥆ leftwards arrow with plus below	infix	270	5	5		
⥇ rightwards arrow through x	infix	270	5	5		
⥈ left right arrow through small circle	infix	270	5	5		
⥉ upwards two-headed arrow from small circle	infix	270	5	5		
⥊ left barb up right barb down harpoon	infix	270	5	5		
⥋ left barb down right barb up harpoon	infix	270	5	5		
⥌ up barb right down barb left harpoon	infix	270	5	5		
⥍ up barb left down barb right harpoon	infix	270	5	5		
⥎ left barb up right barb up harpoon	infix	270	5	5		stretchy accent
⥏ up barb right down barb right harpoon	infix	270	5	5		stretchy
⥐ left barb down right barb down harpoon	infix	270	5	5		stretchy
⥑ up barb left down barb left harpoon	infix	270	5	5		stretchy
⥒ leftwards harpoon with barb up to bar	infix	270	5	5		stretchy
⥓ rightwards harpoon with barb up to bar	infix	270	5	5		stretchy
⥔ upwards harpoon with barb right to bar	infix	270	5	5		stretchy
⥕ downwards harpoon with barb right to bar	infix	270	5	5		stretchy
⥖ leftwards harpoon with barb down to bar	infix	270	5	5		stretchy
⥗ rightwards harpoon with barb down to bar	infix	270	5	5		stretchy
⥘ upwards harpoon with barb left to bar	infix	270	5	5		stretchy
⥙ downwards harpoon with barb left to bar	infix	270	5	5		stretchy
⥚ leftwards harpoon with barb up from bar	infix	270	5	5		stretchy
⥛ rightwards harpoon with barb up from bar	infix	270	5	5		stretchy
⥜ upwards harpoon with barb right from bar	infix	270	5	5		stretchy
⥝ downwards harpoon with barb right from bar	infix	270	5	5		stretchy
⥞ leftwards harpoon with barb down from bar	infix	270	5	5		stretchy
⥟ rightwards harpoon with barb down from bar	infix	270	5	5		stretchy
⥠ upwards harpoon with barb left from bar	infix	270	5	5		stretchy
⥡ downwards harpoon with barb left from bar	infix	270	5	5		stretchy

Character	form	priority	lspace	rspace	minsize	Properties
⥢ leftwards harpoon with barb up above leftwards harpoon with barb down	infix	270	5	5		
⥣ upwards harpoon with barb left beside upwards harpoon with barb right	infix	270	5	5		
⥤ rightwards harpoon with barb up above rightwards harpoon with barb down	infix	270	5	5		
⥥ downwards harpoon with barb left beside downwards harpoon with barb right	infix	270	5	5		
⥦ leftwards harpoon with barb up above rightwards harpoon with barb up	infix	270	5	5		
⥧ leftwards harpoon with barb down above rightwards harpoon with barb down	infix	270	5	5		
⥨ rightwards harpoon with barb up above leftwards harpoon with barb up	infix	270	5	5		
⥩ rightwards harpoon with barb down above leftwards harpoon with barb down	infix	270	5	5		
⥪ leftwards harpoon with barb up above long dash	infix	270	5	5		
⥫ leftwards harpoon with barb down below long dash	infix	270	5	5		
⥬ rightwards harpoon with barb up above long dash	infix	270	5	5		
⥭ rightwards harpoon with barb down below long dash	infix	270	5	5		
⥮ upwards harpoon with barb left beside downwards harpoon with barb right	infix	270	5	5		stretchy
⥯ downwards harpoon with barb left beside upwards harpoon with barb right	infix	270	5	5		stretchy
⥰ right double arrow with rounded head	infix	270	5	5		
⥱ equals sign above rightwards arrow	infix	270	5	5		
⥲ tilde operator above rightwards arrow	infix	270	5	5		
⥳ leftwards arrow above tilde operator	infix	270	5	5		
⥴ rightwards arrow above tilde operator	infix	270	5	5		
⥵ rightwards arrow above almost equal to	infix	270	5	5		
⥶ less-than above leftwards arrow	infix	270	5	5		
⥷ leftwards arrow through less-than	infix	270	5	5		
⥸ greater-than above rightwards arrow	infix	270	5	5		
⥹ subset above rightwards arrow	infix	270	5	5		
⥺ leftwards arrow through subset	infix	270	5	5		
⥻ superset above leftwards arrow	infix	270	5	5		
⥼ left fish tail	infix	270	5	5		
⥽ right fish tail	infix	270	5	5		
⥾ up fish tail	infix	270	5	5		
⥿ down fish tail	infix	270	5	5		

Character	form	priority	lspace	rspace	minsize	Properties
⦀ triple vertical bar delimiter	infix	270	0	0		
⦃ left white curly bracket	prefix	270	0	0		fence stretchy
⦄ right white curly bracket	postfix	270	0	0		fence stretchy
⦅ left white parenthesis	prefix	270	0	0		fence stretchy
⦆ right white parenthesis	postfix	270	0	0		fence stretchy
⦇ z notation left image bracket	prefix	270	0	0		fence stretchy
⦈ z notation right image bracket	postfix	270	0	0		fence stretchy
⦉ z notation left binding bracket	prefix	270	0	0		fence stretchy
⦊ z notation right binding bracket	postfix	270	0	0		fence stretchy
⦋ left square bracket with underbar	prefix	270	0	0		fence stretchy
⦌ right square bracket with underbar	postfix	270	0	0		fence stretchy
⦍ left square bracket with tick in top corner	prefix	270	0	0		fence stretchy
⦎ right square bracket with tick in bottom corner	postfix	270	0	0		fence stretchy
⦏ left square bracket with tick in bottom corner	prefix	270	0	0		fence stretchy
⦐ right square bracket with tick in top corner	postfix	270	0	0		fence stretchy
⦗ left black tortoise shell bracket	prefix	270	0	0		fence stretchy
⦘ right black tortoise shell bracket	postfix	270	0	0		fence stretchy
⦙ dotted fence	infix	270	3	3		
⦚ vertical zigzag line	infix	270	3	3		
⦛ measured angle opening left	infix	270	3	3		
⦜ right angle variant with square	infix	270	3	3		
⦝ measured right angle with dot	infix	270	3	3		
⦞ angle with s inside	infix	270	3	3		
⦟ acute angle	infix	270	3	3		
⧟ double-ended multimap	infix	270	3	3		
⧯ error-barred black square	infix	270	3	3		
⧴ rule-delayed	infix	270	5	5		
⧼ left-pointing curved angle bracket	prefix	270	0	0		fence stretchy
⧽ right-pointing curved angle bracket	postfix	270	0	0		fence stretchy
← leftwards arrow	infix	271	5	5		stretchy accent

Character	form	priority	lspace	rspace	minsize	Properties
↗ north east arrow	infix	271	5	5		stretchy
↘ south east arrow	infix	271	5	5		stretchy
+ plus sign	infix	275	4	4		
- hyphen-minus	infix	275	4	4		
± plus-minus sign	infix	275	4	4		
− minus sign	infix	275	4	4		
∓ minus-or-plus sign	infix	275	4	4		
∔ dot plus	infix	275	4	4		
⊞ squared plus	infix	275	4	4		
⊟ squared minus	infix	275	4	4		
∑ n-ary summation	prefix	290	1	2		stretchy largeop movablelimits
⨊ modulo two sum	prefix	290	1	2		largeop movablelimits
⨋ summation with integral	prefix	290	1	2		largeop movablelimits
∬ double integral	prefix	300	0	1		largeop
∭ triple integral	prefix	300	0	1		largeop
⊕ circled plus	infix	300	4	4		largeop movablelimits
⊖ circled minus	infix	300	4	4		largeop movablelimits
⊘ circled division slash	infix	300	4	4		largeop movablelimits
⨁ n-ary circled plus operator	prefix	300	1	2		largeop movablelimits
∫ integral	prefix	310	0	1		stretchy largeop
∮ contour integral	prefix	310	0	1		stretchy largeop movablelimits
∯ surface integral	prefix	310	0	1		stretchy largeop movablelimits
∰ volume integral	prefix	310	0	1		largeop movablelimits
∱ clockwise integral	prefix	310	0	1		largeop movablelimits
∲ clockwise contour integral	prefix	310	0	1		stretchy largeop movablelimits
∳ anticlockwise contour integral	prefix	310	0	1		stretchy largeop movablelimits
⨌ quadruple integral operator	prefix	310	0	1		largeop movablelimits
⨍ finite part integral	prefix	310	1	2		largeop movablelimits

Character	form	priority	lspace	rspace	minsize	Properties
⨎ integral with double stroke	prefix	310	1	2		largeop movablelimits
⨏ integral average with slash	prefix	310	1	2		largeop movablelimits
⨐ circulation function	prefix	310	1	2		largeop movablelimits
⨑ anticlockwise integration	prefix	310	1	2		largeop movablelimits
⨒ line integration with rectangular path around pole	prefix	310	1	2		largeop movablelimits
⨓ line integration with semicircular path around pole	prefix	310	1	2		largeop movablelimits
⨔ line integration not including the pole	prefix	310	1	2		largeop movablelimits
⨕ integral around a point operator	prefix	310	1	2		largeop movablelimits
⨖ quaternion integral operator	prefix	310	1	2		largeop movablelimits
⨗ integral with leftwards arrow with hook	prefix	310	1	2		largeop movablelimits
⨘ integral with times sign	prefix	310	1	2		largeop movablelimits
⨙ integral with intersection	prefix	310	1	2		largeop movablelimits
⨚ integral with union	prefix	310	1	2		largeop movablelimits
⨛ integral with overbar	prefix	310	1	2		largeop movablelimits
⨜ integral with underbar	prefix	310	1	2		largeop movablelimits
⋃ n-ary union	prefix	320	1	2		stretchy largeop movablelimits
⨃ n-ary union operator with dot	prefix	320	1	2		largeop movablelimits
⨄ n-ary union operator with plus	prefix	320	1	2		largeop movablelimits
⋂ n-ary intersection	prefix	330	1	2		stretchy largeop movablelimits
⨅ n-ary square intersection operator	prefix	330	1	2		largeop movablelimits
⨆ n-ary square union operator	prefix	330	1	2		largeop movablelimits
⫼ large triple vertical bar operator	prefix	330	1	2		largeop movablelimits
⫿ n-ary white vertical bar	prefix	330	1	2		largeop movablelimits
≀ wreath product	infix	340	4	4		
∏ n-ary product	prefix	350	1	2		stretchy largeop movablelimits
∐ n-ary coproduct	prefix	350	1	2		stretchy largeop movablelimits
∩ intersection	infix	350	4	4		
∪ union	infix	350	4	4		

Character	form	priority	lspace	rspace	minsize	Properties
* asterisk	infix	390	3	3		
. full stop	infix	390	3	3		
× multiplication sign	infix	390	4	4		
• bullet	infix	390	4	4		
⁢ invisible times	infix	390	0	0		
⊠ squared times	infix	390	4	4		
⊡ squared dot operator	infix	390	4	4		
⨯ vector or cross product	infix	390	4	4		
· middle dot	infix	400	4	4		
⋅ dot operator	infix	400	4	4		
⊗ circled times	infix	410	4	4		largeop movablelimits
⨂ n-ary circled times operator	prefix	410	1	2		largeop movablelimits
⨉ n-ary times operator	prefix	410	1	2		largeop movablelimits
⋁ n-ary logical or	prefix	420	1	2		stretchy largeop movablelimits
⨈ two logical or operator	prefix	420	1	2		largeop movablelimits
⋀ n-ary logical and	prefix	430	1	2		stretchy largeop movablelimits
⨇ two logical and operator	prefix	430	1	2		largeop movablelimits
% percent sign	infix	440	3	3		
\ reverse solidus	infix	450	0	0		
∖ set minus	infix	450	4	4		stretchy
/ solidus	infix	460	1	1		stretchy
÷ division sign	infix	460	4	4		
∠ angle	prefix	470	0	0		
∡ measured angle	prefix	470	0	0		
∢ spherical angle	prefix	470	0	0		
¬ not sign	prefix	480	2	1		
⊙ circled dot operator	infix	510	4	4		largeop movablelimits
⨀ n-ary circled dot operator	prefix	510	1	2		largeop movablelimits
∂ partial differential	prefix	540	2	1		

Character	form	priority	lspace	rspace	minsize	Properties
∇ nabla	prefix	540	2	1		
↑ upwards arrow	infix	570	5	5		stretchy
↓ downwards arrow	infix	570	5	5		stretchy
↕ up down arrow	infix	570	5	5		stretchy
⇑ upwards double arrow	infix	570	5	5		stretchy
⇓ downwards double arrow	infix	570	5	5		stretchy
⇕ up down double arrow	infix	570	5	5		stretchy
′ prime	postfix	600	0	2		
♭ music flat sign	postfix	600	0	2		
♮ music natural sign	postfix	600	0	2		
♯ music sharp sign	postfix	600	0	2		
! exclamation mark	postfix	610	1	0		
!! multiple character operator: !!	postfix	610	1	0		
min multiple character operator: "min"	prefix	645	1	3		
σ greek small letter sigma	prefix	645	1	1		
ℑ black-letter capital i	prefix	645	1	1		
ℓ script small l	prefix	645	1	1		
ℜ black-letter capital r	prefix	645	1	1		
ⅅ double-struck italic capital d	prefix	645	2	1		
ⅆ double-struck italic small d	prefix	645	2	0		
ⅇ double-struck italic small e	prefix	645	0	0		
ⅈ double-struck italic small i	prefix	645	0	0		
ⅉ double-struck italic small j	prefix	645	0	0		
√ square root	prefix	645	1	1		stretchy
∛ cube root	prefix	645	1	1		
∜ fourth root	prefix	645	1	1		
⁡ function application	infix	650	0	0		
& ampersand	postfix	680	0	0		
’ apostrophe	postfix	680	0	0		accent
++ multiple character operator: ++	postfix	680	0	0		

Character	form	priority	lspace	rspace	minsize	Properties
– multiple character operator: –	postfix	680	0	0		
^ circumflex accent	postfix	680	0	0		stretchy accent
‘ grave accent	postfix	680	0	0		accent
 tilde	postfix	680	0	0		stretchy accent
¨ diaeresis	postfix	680	0	0		accent
¯ macron	postfix	680	0	0		stretchy accent
° degree sign	postfix	680	0	0		
´ acute accent	postfix	680	0	0		accent
¸ cedilla	postfix	680	0	0		accent
ˆ modifier letter circumflex accent	postfix	680	0	0		stretchy accent
ˇ caron	postfix	680	0	0		stretchy accent
˘ breve	postfix	680	0	0		accent
˙ dot above	postfix	680	0	0		accent
˚ ring above	postfix	680	0	0		accent
˜ small tilde	postfix	680	0	0		stretchy accent
˝ double acute accent	postfix	680	0	0		accent
̂ combining circumflex accent	postfix	680	0	0		accent
̑ combining inverted breve	postfix	680	0	0		accent
̲ combining low line	postfix	680	0	0		stretchy accent
⁤ invisible plus	infix	680	0	0		
⃛ combining three dots above	postfix	680	0	0		accent
⎴ top square bracket	postfix	680	0	0		stretchy accent
⎵ bottom square bracket	postfix	680	0	0		stretchy accent
⏜ top parenthesis	postfix	680	0	0		stretchy accent
⏝ bottom parenthesis	postfix	680	0	0		stretchy accent
⏞ top curly bracket	postfix	680	0	0		stretchy accent
⏟ bottom curly bracket	postfix	680	0	0		stretchy accent
✓ check mark	infix	680	0	0		stretchy accent
∘ ring operator	infix	700	4	4		

Appendix D

Glossary (Non-Normative)

Several of the following definitions of terms have been borrowed or modified from similar definitions in documents originating from W3C or standards organizations. See the individual definitions for more information.

Argument A child of a presentation layout schema. That is, 'A is an argument of B' means 'A is a child of B and B is a presentation layout schema'. Thus, token elements have no arguments, even if they have children (which can only be `malignmark`).

Attribute A parameter used to specify some property of an SGML or XML element type. It is defined in terms of an attribute name, attribute type, and a default value. A value may be specified for it on a start-tag for that element type.

Axis The axis is an imaginary alignment line upon which a fraction line is centered. Often, operators as well as characters that can stretch, such as parentheses, brackets, braces, summation signs etc., are centered on the axis, and are symmetric with respect to it.

Baseline The baseline is an imaginary alignment line upon which a glyph without a descender rests. The baseline is an intrinsic property of the glyph (namely a horizontal line). Often baselines are aligned (joined) during typesetting.

Black box The bounding box of the actual size taken up by the viewable portion (ink) of a glyph or expression.

Bounding box The rectangular box of smallest size, taking into account the constraints on boxes allowed in a particular context, which contains some specific part of a rendered display.

Box A rectangular plane area considered to contain a character or further sub-boxes, used in discussions of rendering for display. It is usually considered to have a baseline, height, depth and width.

Cascading Style Sheets (CSS) A language that allows authors and readers to attach style (e.g. fonts, colors and spacing) to HTML and XML documents.

Character A member of a set of identifiers used for the organization, control or representation of text. ISO/IEC Standard 10646-1:1993 uses the word 'data' here instead of 'text'.

Character data (`CDATA`) A data type in SGML and XML for raw data that does not include markup or entity references. Attributes of type `CDATA` may contain entity references. These are expanded by an XML processor before the attribute value is processed as `CDATA`.

Character or expression depth Distance between the baseline and bottom edge of the character glyph or expression. Also known as the descent.

Character or expression height Distance between the baseline and top edge of the character glyph or expression. Also known as the ascent.

Character or expression width Horizontal distance taken by the character glyph as indicated in the font metrics, or the total width of an expression.

Condition A MathML content element used to place a mathematical condition on one or more variables.

Contained (element A is contained in element B) A is part of B's content.

Container (Constructor) A non-empty MathML Content element that is used to construct a mathematical object such as a number, set, or list.

Content elements MathML elements that explicitly specify the mathematical meaning of a portion of a MathML expression (defined in Chapter 4).

Content token element Content element having only `PCDATA`, `sep` and presentation expressions as content. Represents either an identifier (`ci`) or a number (`cn`).

Context (of a given MathML expression) Information provided during the rendering of some MathML data to the rendering process for the given MathML expression; especially information about the MathML markup surrounding the expression.

Declaration An instance of the declare element.

Depth (of a box) The distance from the baseline of the box to the bottom edge of the box.

Direct sub-expression (of a MathML expression 'E') A sub-expression directly contained in E.

Directly contained (element A in element B) A is a child of B (as defined in XML), in other words A is contained in B, but not in any element that is itself contained in B.

Document Object Model A model in which the document or Web page is treated as an object repository. This model is developed by the DOM Working Group (DOM) of the W3C.

Document Style Semantics and Specification Language (DSSSL) A method of specifying the formatting and transformation of SGML documents. ISO International Standard 10179:1996.

Document Type Definition (DTD) In SGML or XML, a DTD is a formal definition of the elements and the relationship among the data elements (the structure) for a particular type of document.

Em A font-relative measure encoded by the font. Before electronic typesetting, an `"em"` was the width of an 'M' in the font. In modern usage, an `"em"` is either specified by the designer of the font or is taken to be the height (point size) of the font. Em's are typically used for font-relative horizontal sizes.

Ex A font-relative measure that is the height of an 'x' in the font. `"ex"`s are typically used for font-relative vertical sizes.

Height (of a box) The distance from the baseline of the box to the top edge of the box.

Inferred `mrow` An `mrow` element that is 'inferred' around the contents of certain layout schemata when they have other than exactly one argument. Defined precisely in Section 3.1.9

Embedded object Embedded objects such as Java applets, Microsoft Component Object Model (COM) objects (e.g. ActiveX Controls and ActiveX Document embeddings), and plug-ins that reside in an HTML document.

Embellished operator An operator, including any 'embellishment' it may have, such as superscripts or style information. The 'embellishment' is represented by a layout schema that contains the operator itself. Defined precisely in Section 3.2.5.

Entity reference A sequence of ASCII characters of the form `&name;` representing some other data, typically a non-ASCII character, a sequence of characters, or an external source of data, e.g. a file containing a set of standard entity definitions such as ISO Latin 1.

Extensible Markup Language (XML) A simple dialect of SGML intended to enable generic SGML to be served, received, and processed on the Web.

Fences In typesetting, bracketing tokens like parentheses, braces, and brackets, which usually appear in matched pairs.

Font A particular collection of glyphs of a typeface of a given size, weight and style, for example 'Times Roman Bold 12 point'.

Glyph The actual shape (bit pattern, outline) of a character. ISO/IEC Standard 9541-1:1991 defines a glyph as a recognizable abstract graphic symbol that is independent of any specific design.

Indirectly contained A is contained in B, but not directly contained in B.

Instance of MathML A single instance of the top level element of MathML, and/or a single instance of embedded MathML in some other data format.

Inverse function A mathematical function that, when composed with the original function acts like an identity function.

Lambda expression A mathematical expression used to define a function in terms of variables and an expression in those variables.

Layout schema (plural: schemata) A presentation element defined in chapter 3, other than the token elements and empty elements defined there (i.e. not the elements defined in Section 3.2 and Section 3.5.5, or the empty elements `none` and `mprescripts` defined in Section 3.4.7). The layout schemata are never empty elements (though their content may contain nothing in some cases), are always expressions, and all allow any MathML expressions as arguments (except for requirements on argument count, and the requirement for a certain empty element in `mmultiscripts`).

Mathematical Markup Language (MathML) The markup language specified in this document for describing the structure of mathematical expressions, together with a mathematical context.

MathML element An XML element that forms part of the logical structure of a MathML document.

MathML expression (within some valid MathML data) A single instance of a presentation element, except for the empty elements `none` or `mprescripts`, or an instance of `malignmark` within a token element (defined below); or a single instance of certain of the content elements (see Chapter 4 for a precise definition of which ones).

Multi-purpose Internet Mail Extensions (MIME) A set of specifications that offers a way to interchange text in languages with different character sets, and multimedia content among many different computer systems that use Internet mail standards.

Operator, content element A mathematical object that is applied to arguments using the `apply` element.

Operator, an `mo` element Used to represent ordinary operators, fences, separators in MathML presentation. (The token element `mo` is defined in Section 3.2.5).

OpenMath A general representation language for communicating mathematical objects between application programs.

Parsed character data (`PCDATA`) An SGML/XML data type for raw data occurring in a context where text is parsed and markup (for instance entity references and element start/end tags) is recognized.

Point Point is often abbreviated 'pt'. The value of 1 pt is approximately 1/72 inch. Points are typically used to specify absolute sizes for font-related objects.

Pre-defined function One of the empty elements defined in [mathml3cds] and used with the `apply` construct to build function applications.

Presentation elements MathML tags and entities intended to express the syntactic structure of mathematical notation (defined in Chapter 3).

Presentation layout schema A presentation element that can have other MathML elements as content.

Presentation token element A presentation element that can contain only parsed character data or the `malignmark` element.

Qualifier A MathML content element that is used to specify the value of a specific named parameter in the application of selected pre-defined functions.

Relation A MathML content element used to construct expressions such as $a < b$.

Render Faithfully translate into application-specific form allowing native application operations to be performed.

Schema Schema (plural: schemata or schemas). See 'presentation layout schema'.

Scope of a declaration The portion of a MathML document in which a particular definition is active.

Selected sub-expression (of an `maction` element) The argument of an `maction` element (a layout schema defined in Section 3.7) that is (at any given time) 'selected' within the viewing

state of a MathML renderer, or by the `selection` attribute when the element exists only in MathML data. Defined precisely in the abovementioned section.

Space-like (MathML expression) A MathML expression that is ignored by the suggested rendering rules for MathML presentation elements when they determine operator forms and effective operator rendering attributes based on operator positions in `mrow` elements. Defined precisely in Section 3.2.7.

Standard Generalized Markup Language (SGML) An ISO standard (ISO 8879:1986) that provides a formal mechanism for the definition of document structure via DTDs (Document Type Definitions), and a notation for the markup of document instances conforming to a DTD.

Sub-expression (of a MathML expression 'E') A MathML expression contained (directly or indirectly) in the content of E.

Suggested rendering rules for MathML presentation elements Defined throughout Chapter 3; the ones that use other terms defined here occur mainly in Section 3.2.5 and in Section 3.7.

TEX A software system developed by Professor Donald Knuth for typesetting documents.

Token element Presentation token element or a Content token element. (See above.)

Top-level element (of MathML) `math` (defined in Section 2.2).

Typeface A typeface is a specific design of a set of letters, numbers and symbols, such as 'Times Roman' or 'Chicago'.

Valid MathML data MathML data that (1) conforms to the MathML DTD, (2) obeys the additional rules defined in the MathML standard for the legal contents and attribute values of each MathML element, and (3) satisfies the EBNF grammar for content elements.

Width (of a box) The distance from the left edge of the box to the right edge of the box.

Extensible Style Language (XSL) A style language for XML developed by W3C. See XSL FO and XSLT.

XSL Formatting Objects (XSL FO) An XML vocabulary to express formatting, which is a part of XSL.

XSL Transformation (XSLT) A language to express the transformation of XML documents into other XML documents.

Appendix E

Working Group Membership and Acknowledgments (Non-Normative)

E.1 The Math Working Group Membership

The present W3C Math Working Group (2006–2009) is co-chaired by Patrick Ion of the AMS and Robert Miner of Design Science. Contact the co-chairs about membership in the Working Group. For the current membership see the W3C Math home page.

Participants in the Working Group responsible for MathML 3.0 have been:

- Ron Ausbrooks, Mackichan Software, Las Cruces NM, USA
- Laurent Bernardin, Waterloo Maple, Inc., Waterloo ON, CAN
- Pierre-Yves Bertholet, MITRE Corporation, McLean VA, USA
- Bert Bos, W3C, Sophia-Antipolis, FRA
- Mike Brenner, MITRE Corporation, Bedford MA, USA
- Olga Caprotti, University of Helsinki, Helsinki, FI
- David Carlisle, NAG Ltd., Oxford, UK
- Giorgi Chavchanidze, Opera Software, Oslo, NO
- Ananth Coorg, The Boeing Company, Seattle WA, USA
- Stéphane Dalmas, INRIA, Sophia Antipolis, FRA
- Stan Devitt, Agfa-Gevaert N. V., Trier, GER
- Sam Dooley, Integretech, USA
- Margaret Hinchcliffe, Waterloo Maple, Inc., Waterloo ON, CAN
- Patrick Ion, W3C Invited Experts:Mathematical Reviews (American Mathematical Society), Ann Arbor MI, USA
- Michael Kohlhase, German Research Center for Artificial Intelligence (DFKI) Gmbh, GER
- Azzeddine Lazrek, W3C Invited Experts: University of Marrakesh, Morocco
- Dennis Leas, DAISY Consortium
- Paul Libbrecht, German Research Center for Artificial Intelligence (DFKI) Gmbh, GER
- Manolis Mavrikis, University of Edinburgh, Edinburg, UK
- Bruce Miller, National Institute of Standards and Technology (NIST), Gaithersburg MD, USA
- Robert Miner, Design Science Inc., Long Beach CA, USA
- Christopher Rowley, The Open University, UK
- Murray Sargent III, Microsoft, Redmond WA, USA
- Kyle Siegrist, Mathematical Association of America, Washington DC, USA
- Andrew Smith, Maplesoft Inc., Canada
- Neil Soiffer, Design Science Inc., Long Beach CA, USA
- Stephen Watt, University of Western Ontario, London ON, CAN
- Mohamed Zergaoui, Innovimax, Paris, FRA

All the above persons have been members of the currently chartered Math Working Group, but some not for the whole life of the Working Group. The 22 authors listed for MathML3 at the start of this specification are those who contributed reworkings and reformulations used in the actual text of the specification. Thus the list includes the principal authors of MathML2 much of whose text was re-purposed here. They were, of course, supported and encouraged by the activity and discussions of the whole Math Working Group, and by helpful commentary from outside it, both within the W3C and further afield.

For 2003 to 2006 W3C Math Activity comprised a Math Interest Group, chaired by David Carlisle of NAG and Robert Miner of Design Science.

The W3C Math Working Group (2001–2003) was co-chaired by Patrick Ion of the AMS, and Angel Diaz of IBM from June 2001 to May 2002; afterwards Patrick Ion continued as chair until the end of the WG's extended charter.

Participants in the Working Group responsible for MathML 2.0, second edition were:

- Ron Ausbrooks, Mackichan Software, Las Cruces NM, USA
- Laurent Bernardin, Waterloo Maple, Inc., Waterloo ON, CAN
- Stephen Buswell, Stilo Technology Ltd., Bristol, UK
- David Carlisle, NAG Ltd., Oxford, UK
- Stéphane Dalmas, INRIA, Sophia Antipolis, FR
- Stan Devitt, Stratum Technical Services Ltd., Waterloo ON, CAN (earlier with Waterloo Maple, Inc., Waterloo ON, CAN)
- Max Froumentin, W3C, Sophia-Antipolis, FRA
- Patrick Ion, Mathematical Reviews (American Mathematical Society), Ann Arbor MI, USA
- Michael Kohlhase, DFKI, GER
- Robert Miner, Design Science Inc., Long Beach CA, USA
- Luca Padovani, University of Bologna, IT
- Ivor Philips, Boeing, Seattle WA, USA
- Murray Sargent III, Microsoft, Redmond WA, USA
- Neil Soiffer, Wolfram Research Inc., Champaign IL, USA
- Paul Topping, Design Science Inc., Long Beach CA, USA
- Stephen Watt, University of Western Ontario, London ON, CAN

Earlier active participants of the W3C Math Working Group (2001 – 2003) have included:

- Angel Diaz, IBM Research Division, Yorktown Heights NY, USA
- Sam Dooley, IBM Research, Yorktown Heights NY, USA
- Barry MacKichan, MacKichan Software, Las Cruces NM, USA

The W3C Math Working Group was co-chaired by Patrick Ion of the AMS, and Angel Diaz of IBM from July 1998 to December 2000.

Participants in the Working Group responsible for MathML 2.0 were:

- Ron Ausbrooks, Mackichan Software, Las Cruces NM, USA
- Laurent Bernardin, Maplesoft, Waterloo ON, CAN
- Stephen Buswell, Stilo Technology Ltd., Cardiff, UK
- David Carlisle, NAG Ltd., Oxford, UK
- Stéphane Dalmas, INRIA, Sophia Antipolis, FR
- Stan Devitt, Stratum Technical Services Ltd., Waterloo ON, CAN (earlier with Waterloo Maple, Inc., Waterloo ON, CAN)
- Angel Diaz, IBM Research Division, Yorktown Heights NY, USA
- Ben Hinkle, Waterloo Maple, Inc., Waterloo ON, CAN

- Stephen Hunt, MATH.EDU Inc., Champaign IL, USA
- Douglas Lovell, IBM Hawthorne Research, Yorktown Heights NY, USA
- Patrick Ion, Mathematical Reviews (American Mathematical Society), Ann Arbor MI, USA
- Robert Miner, Design Science Inc., Long Beach CA, USA (earlier with Geometry Technologies Inc., Minneapolis MN, USA)
- Ivor Philips, Boeing, Seattle WA, USA
- Nico Poppelier, Penta Scope, Amersfoort, NL (earlier with Salience and Elsevier Science, NL)
- Dave Raggett, W3C (Openwave), Bristol, UK (earlier with Hewlett-Packard)
- T.V. Raman, IBM Almaden, Palo Alto CA, USA (earlier with Adobe Inc., Mountain View CA, USA)
- Murray Sargent III, Microsoft, Redmond WA, USA
- Neil Soiffer, Wolfram Research Inc., Champaign IL, USA
- Irene Schena, Universitá di Bologna, Bologna, IT
- Paul Topping, Design Science Inc., Long Beach CA, USA
- Stephen Watt, University of Western Ontario, London ON, CAN

Earlier active participants of this second W3C Math Working Group have included:

- Sam Dooley, IBM Research, Yorktown Heights NY, USA
- Robert Sutor, IBM Research, Yorktown Heights NY, USA
- Barry MacKichan, MacKichan Software, Las Cruces NM, USA

At the time of release of MathML 1.0 [MathML1] the Math Working Group was co-chaired by Patrick Ion and Robert Miner, then of the Geometry Center. Since that time several changes in membership have taken place. In the course of the update to MathML 1.01, in addition to people listed in the original membership below, corrections were offered by David Carlisle, Don Gignac, Kostya Serebriany, Ben Hinkle, Sebastian Rahtz, Sam Dooley and others.

Participants in the Math Working Group responsible for the finished MathML 1.0 specification were:

- Stephen Buswell, Stilo Technology Ltd., Cardiff, UK
- Stéphane Dalmas, INRIA, Sophia Antipolis, FR
- Stan Devitt, Maplesoft Inc., Waterloo ON, CAN
- Angel Diaz, IBM Research Division, Yorktown Heights NY, USA
- Brenda Hunt, Wolfram Research Inc., Champaign IL, USA
- Stephen Hunt, Wolfram Research Inc., Champaign IL, USA
- Patrick Ion, Mathematical Reviews (American Mathematical Society), Ann Arbor MI, USA
- Robert Miner, Geometry Center, University of Minnesota, Minneapolis MN, USA
- Nico Poppelier, Elsevier Science, Amsterdam, NL
- Dave Raggett, W3C (Hewlett Packard), Bristol, UK
- T.V. Raman, Adobe Inc., Mountain View CA, USA
- Bruce Smith, Wolfram Research Inc., Champaign IL, USA
- Neil Soiffer, Wolfram Research Inc., Champaign IL, USA
- Robert Sutor, IBM Research, Yorktown Heights NY, USA
- Paul Topping, Design Science Inc., Long Beach CA, USA
- Stephen Watt, University of Western Ontario, London ON, CAN
- Ralph Youngen, American Mathematical Society, Providence RI, USA

Others who had been members of the W3C Math WG for periods at earlier stages were:

- Stephen Glim, Mathsoft Inc., Cambridge MA, USA
- Arnaud Le Hors, W3C, Cambridge MA, USA
- Ron Whitney, Texterity Inc., Boston MA, USA
- Lauren Wood, SoftQuad, Surrey BC, CAN

- Ka-Ping Yee, University of Waterloo, Waterloo ON, CAN

E.2 Acknowledgments

The Working Group benefited from the help of many other people in developing the specification for MathML 1.0. We would like to particularly name Barbara Beeton, Chris Hamlin, John Jenkins, Ira Polans, Arthur Smith, Robby Villegas and Joe Yurvati for help and information in assembling the character tables in Chapter 7, as well as Peter Flynn, Russell S.S. O'Connor, Andreas Strotmann, and other contributors to the www-math mailing list for their careful proofreading and constructive criticisms.

As the Math Working Group went on to MathML 2.0, it again was helped by many from the W3C family of Working Groups with whom we necessarily had a great deal of interaction. Outside the W3C, a particularly active relevant front was the interface with the Unicode Technical Committee (UTC) and the NTSC WG2 dealing with ISO 10646. There the STIX project put together a proposal for the addition of characters for mathematical notation to Unicode, and this work was again spearheaded by Barbara Beeton of the AMS. The whole problem ended split into three proposals, two of which were advanced by Murray Sargent of Microsoft, a Math WG member and member of the UTC. But the mathematical community should be grateful for essential help and guidance over a couple of years of refinement of the proposals to help mathematics provided by Kenneth Whistler of Sybase, and a UTC and WG2 member, and by Asmus Freytag, also involved in the UTC and WG2 deliberations, and always a stalwart and knowledgeable supporter of the needs of scientific notation.

Appendix F

Changes (Non-Normative)

F.1 Changes between MathML 2.0 Second Edition and MathML 3.0

- Changes to Chapter 2.
  - The attribute `href` added to the common MathML attributes, Section 2.1.6 to allow hypertext links.
  - Additional attributes added to the `math` element, see Section 2.2.1.
- Changes to Chapter 3.
  - Introduced mechanisms for controlling the Directionality of layout, as described in Section 3.1.5.
  - Introduced mechanisms for controlling linebreaking Section 3.1.7.
  - Extended `mglyph` to support general image inclusion, Section 3.2.9.
  - The facilities for adjusting spacing with `mpadded` have been extended and rationalised, Section 3.3.6.
  - Introduced new presentation elements for elementary math layouts, Section 3.6: `mstack`, `mlongdiv`, `msgroup`, `msrow`, `mscarries` and `mscarry`.
- Changes to Chapter 4.
  - Introduced new content elements `bind`, `share`, `cerror`, `cs` and `cbytes`.
  - Removed deprecated content elements `reln` and `fn`.
  - Removed content element `declare`.
  - The concept of Strict Content MathML and the use of OpenMath Content Dictionaries has been introduced, and the whole chapter restructured.
- Changes to Chapter 5.
- New Chapter: Chapter 6.
- Changes to Chapter 7.This chapter is much reduced from the corresponding chapter in previous releases of MathML. All the tables and much of the other content of this chapter is now maintained as a separate document [Entities]
- Changes to Appendix A.The Normative version of the grammar is now expressed in Relax NG, with DTD and XSD versions being derived.
- New Appendix: Appendix B.
- Changes to Appendix C.The Operator Dictionary table has been updated and rationalised and presented in a new format.
- MathML DOMThe chapter and appendices relating to the MathML DOM have been removed from this specification, with the intention of updating them and publishing them as a separate document at a later time.

Appendix G

References (Non-Normative)

[AAP-math] ANSI/NISO Z39.59-1998; *AAP Math DTD*, Standard for Electronic Manuscript Preparation and MarkUp. (Association of American Publishers, Inc., Washington, DC) Bethesda, MD, 1988.

[Abramowitz1997] Abramowitz, Milton, Irene A. Stegun (editors); *Mathematical Fuctions: With Formulas, Graphs, and Mathematical Tables*. Dover Publications Inc., December 1977, ISBN: 0-4866-1272-4.

[Behaviors] Vidur Apparao, Daniel Glazman, and Chris Wilson (editors) *Behavioral Extensions to CSS* World Wide Web Consortium Working Draft, 4 August 1999. (`http://www.w3.org/TR/1999/WD-becss-19990804`)

[Bidi] Mark Davis; *The Bidirectional Algorithm*, Unicode Standard Annex #9, August 2000. (`http://www.unicode.org/unicode/reports/tr9/`)

[Buswell1996] Buswell, S., Healey, E.R. Pike, and M. Pike; *SGML and the Semantic Representation of Mathematics*. UIUC Digital Library Initiative SGML Mathematics Workshop, May 1996 and SGML Europe 96 Conference, Munich 1996.

[CSS1] Lie, Håkon Wium and Bert Bos; *Cascading Style Sheets, level 1*, W3C Recommendation, 17 December 1996. (`http://www.w3.org/TR/1999/REC-CSS1-19990111`)

[CSS2] Bert Bos, Håkon Wium Lie, Chris Lilley, and Ian Jacobs (editors); *Cascading Style Sheets, level 2 CSS2 Specification*, W3C Recommendation, 12 May 1998. (`http://www.w3.org/TR/1998/REC-CSS2-19980512/`)

[CSS21] Bert Bos, Tantek Çelik, Ian Hickson, Håkon Wium Lie (editors); *Cascading Style Sheets, Level 2 Revision 1 (CSS 2.1) Specification*, W3C Candidate Recommendation 19 July 2007. (`http://www.w3.org/TR/CSS21/`)

[Cajori1928] Cajori, Florian; *A History of Mathematical Notations*, vol. I & II. Open Court Publishing Co., La Salle Illinois, 1928 & 1929 republished Dover Publications Inc., New York, 1993, xxviii+820 pp. ISBN 0-486-67766-4 (paperback).

[Carroll1871] Carroll, Lewis [Rev. C.L. Dodgson]; *Through the Looking Glass and What Alice Found There*. Macmillian & Co., 1871.

[Chaundy1954] Chaundy, T.W., P.R. Barrett, and C. Batey; *The Printing of Mathematics. Aids for authors and editors and rules for compositors and readers at the University Press, Oxford*. Oxford University Press, London, 1954, ix+105 pp.

[DOM] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne (editors); *Document Object Model (DOM) Level 2 Core Specification* World Wide Web Consortium Recommendation, 13 November, 2000. (`http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/`)

[Entities] David Carlisle, Patrick Ion *XML Entity definitions for Characters (Editor's draft)* Draft 17 November 2007 (http://www.w3.org/2003/entities/2007doc/)

[HTML4] Raggett, Dave, Arnaud Le Hors and Ian Jacobs (editors); *HTML 4.01 Specification*, 24 December 1999. (http://www.w3.org/TR/1999/REC-html401-19991224/); section on data types.

[HTTP11] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee; *Hypertext Transfer Protocol – HTTP/1.1*, June 1999.

[Higham1993] Higham, Nicholas J.; *Handbook of writing for the mathematical sciences*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1993. xii+241 pp. ISBN: 0-89871-314-5.

[IEEE754] IEEE *IEEE Standard for Floating-Point Arithmetic* (http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4610935)

[ISO-12083] ISO 12083:1993; *ISO 12083 DTD* Information and Documentation - Electronic Manuscript Preparation and Markup. International Standards Organization, Geneva, Switzerland, 1993.

[ISOIEC10646-1] ISO/IEC JTC1/SC2/WG2; *ISO/IEC 10646-1: 2000*, Information technology – Universal-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane. International Standards Organization, Geneva, Switzerland, 2000.

[Knuth1986] Knuth, Donald E.; *The TEXbook*. American Mathematical Society, Providence, RI and Addison-Wesley Publ. Co., Reading, MA, 1986, ix+483 pp. ISBN: 0-201-13448-9.

[MathML1] Patrick Ion, Robert Miner, *Mathematical Markup Language (MathML) 1.01 Specification* W3C Recommendation, revision of 7 July 1999 (http://www.w3.org/TR/REC-MathML/)

[MathML2] David Carlisle, Patrick Ion, Robert Miner, Nico Poppelier, *Mathematical Markup Language (MathML) Version 2.0* W3C Recommendation 21 February 2001 (http://www.w3.org/TR/2001/REC-MathML2-20010221/)

[MathMLTypes] Stan Devitt, Michael Kohlhase, Max Froumentin (Editors); *Structured Types in MathML 2.0*, (http://www.w3.org/TR/mathml-types/) W3C Working Group Note 10 November 2003.

[MathMLforCSS] Bert Bos, David Carlisle, George Chavchanidze, Patrick D. F. Ion, Bruce R. Miller *A MathML for CSS profile* W3C Working Draft 24 September 2007 (http://www.w3.org/TR/2007/WD-mathml-for-css-20070924/)

[Modularization] Robert Adams, Murray Altheim, Frank Boumphrey, Sam Dooley, Shane McCarron, Sebastian Schnitzenbaumer, Ted Wugofski (editors); *Modularization of XHTML[tm]*, World Wide Web Consortium Recommendation, 10 April 2001. (http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410/)

[Namespaces] Tim Bray, Dave Hollander, Andrew Layman (editors); *Namespaces in XML*, World Wide Web Consortium Recommendation, 14 January 1999. (http://www.w3.org/TR/1999/REC-xml-names-19990114/)

[OMDoc1.2] Michael Kohlhase *OMDoc - An open markup format for mathematical documents [Version 1.2]* LNAI 4180, Springer Verlag, 2006 (http://omdoc.org/pubs/omdoc1.2.pdf) .

[OpenMath2000] O. Caprotti, D. P. Carlisle, A. M. Cohen (editors); *The OpenMath Standard*, February 2000. (http://www.openmath.org/standard)

[OpenMath2004] S. Buswell, O. Caprotti, D. P. Carlisle, M. C. Dewar, M. Gaëtano and M. Kohlhase (editors); *The OpenMath Standard Version 2.0*, June 2004. (http://www.openmath.org/standard/om20-2004-06-30/)

[Pierce1961] Pierce, John R.; *An Introduction to Information Theory. Symbols, Signals and Noise.*. Revised edition of *Symbols, Signals and Noise: the Nature and Process of Communication* (1961). Dover Publications Inc., New York, 1980, xii+305 pp. ISBN 0-486-24061-4.

[Poppelier1992] Poppelier, N.A.F.M., E. van Herwijnen, and C.A. Rowley; *Standard DTD's and Scientific Publishing*, EPSIG News 5 (1992) #3, September 1992, 10-19.

[RELAX-NG] Clark, James and Makoto Murata; *RELAX NG Specification*. The Organization for the Advancement of Structured Information Standards [OASIS] 2001.

[RFC2045] N. Freed and N. Borenstein; *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC 2045, November 1996. (`http://www.ietf.org/rfc/rfc2045.txt`)

[RFC2046] N. Freed and N. Borenstein; *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, RFC 2045, November 1996. (`http://www.ietf.org/rfc/rfc2046.txt`)

[RFC3023] M. Murata, S. St.Laurent and D. Kohn; *XML Media Types*, RFC 3023, January 2001. (`http://www.ietf.org/rfc/rfc3023.txt`)

[RFC3986] T. Berners-Lee, R. Fielding and L. Masinter; *Uniform Resource Identifier (URI): Generic Syntax*, RFC 3986, January 2005. (`http://tools.ietf.org/html/rfc3986`)

[RFC4288] N. Freed and J. Klensin; *Media Type Specifications and Registration Procedures*, RFC 4288, December 2005. (`http://www.ietf.org/rfc/rfc4288.txt`)

[RelaxNG] *A Schema Language for XML* (`http://www.relaxng.org`)

[RelaxNGBook] Eric van der Vlist; *RELAXNG: A simple schema language for XML* O'Reilly 2004

[RodWatt2000] Igor Rodionov, Stephen Watt; *Content-Faithful Stylesheets for MathML*. Technical Report TR-00-23, Ontario Research Center for Computer Algebra, December 2000. (`http://www.orcca.on.ca/TechReports/2000/TR-00-24.html`)

[SVG1.1] Dean Jackson, Jon Ferraiolo, Jun Fujisawa, eds. *Scalable Vector Graphics (SVG) 1.1 Specification* W3C Recommendation, 14 January 2003 (`http://www.w3.org/TR/2003/REC-SVG11-20030114/`)

[Spivak1986] Spivak, M. D.; *The Joy of TEX A gourmet guide to typesetting with the AMS-TEX macro package*. American Mathematical Society, Providence, RI, MA 1986, xviii+290 pp. ISBN: 0-8218-2999-8.

[Swanson1979] Swanson, Ellen; *Mathematics into type: Copy editing and proofreading of mathematics for editorial assistants and authors*. Revised edition. American Mathematical Society, Providence, R.I., 1979. x+90 pp. ISBN: 0-8218-0053-1.

[Swanson1999] Swanson, Ellen; *Mathematics into type: Updated Edition*. American Mathematical Society, Providence, R.I., 1999. 102 pp. ISBN: 0-8218-1961-5.

[Thieme1983] Thieme, Romeo; *Satz und bedeutung mathematischer Formeln [Typesetting and meaning of mathematical formulas]*. Reprint of the 1934 original. Edited by Karl Billmann, Helmut Bodden and Horst Nacke. Werner-Verlag Gmbh, Dusseldorf, 1983, viii + 92 pp. ISBN 3-8041-3549-8.

[UAX15] Unicode Standard Annex 15, Version 4.0.0; *Unicode Normalization Forms*, The Unicode Consortium, 2003-04-17. (`http://www.unicode.org/reports/tr15/tr15-23.html`)

[URI] The Internet Society; *Uniform Resource Identifier (URI): Generic Syntax*, Addison-Wesley Professional. ISBN 0321480910. (`http://labs.apache.org/webarch/uri/rfc/rfc3986.html`)

[Unicode] The Unicode Consortium; *The Unicode Standard, Version 5.0*, Addison-Wesley Professional. ISBN 0321480910. (`http://www.unicode.org/unicode/standard/standard.html`)

[XHTML] Steve Pemberton, Murray Altheim, et al.; *XHTML[tm] 1.0: The Extensible HyperText Markup Language* World Wide Web Consortium Recommendation, 26 January 2000. (http://www.w3.org/TR/2000/REC-xhtml1-20000126/)

[XHTML-MathML-SVG] Masayasu Ishikawa, ed., *An HTML + MathML + SVG Profile* W3C Working Draft, 9 August 2002. (http://www.w3.org/TR/2002/WD-XHTMLplusMathMLplusSVG-20020809/)

[XLink] Steve DeRose, Eve Maler, David Orchard (editors); *XML Linking Language (XLink) Version 1.0*, World Wide Web Consortium Recommendation, 27 June 2001. (http://www.w3.org/TR/2001/REC-xlink-20010627/)

[XML] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen and Eve Maler (editors); *Extensible Markup Language (XML)*, (http://www.w3.org/TR/xml)

[XMLSchemaDatatypes] Paul V. Biron, Ashok Malhotra, editors; *XML Schema Part 2: Datatypes Second Edition*, World Wide Web Consortium Recommendation, 28 Oct 2004. (http://www.w3.org/TR/2001/REC-xmlschema-2-20041028/)

[XMLSchemas] David C. Fallside, editor; *XML Schema Part 0: Primer*, World Wide Web Consortium Recommendation, 2 May 2001. (http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/)

[XPath] James Clark, Steve DeRose(editors); *XML Path Language Version 1.0*, World Wide Web Consortium Recommendation, 16. November 1999. (http://www.w3.org/TR/1999/REC-xpath-19991116)

[XPointer] Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh (editors); *XML Pointer Framework*, World Wide Web Consortium Recommendation, 25 March 2003. (http://www.w3.org/TR/2003/REC-xptr-framework-20030325/)

[XSLT] James Clark (editor); *XSL Transformations (XSLT) Version 1.0*, World Wide Web Consortium Recommendation, 16 November 1999. (http://www.w3.org/TR/1999/REC-xslt-19991116)

[Zwillinger1988] Daniel Zwillinger (editor); *Standard Mathematical Tables and Formulae (30th Edition)*. CRC Press LLC, January 1996, ISBN: 0-8493-2479-3.

[mathml3cds] Carlisle, Davenport, Kohlhase, eds; *The MathML3 Content Dictionaries*. Joint Document by OpenMath Society and W3C Math WG, under development.

[owl] Deborah L. McGuinness and Frank van Harmelen (editors); *OWL Web Ontology Language Overview* February 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/

[rdf] Graham Klyne, Jeremy J. Carroll, Brian McBride (editors); *Resource Description Framework (RDF): Concepts and Abstract Syntax*,February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[roadmap] Patrick Ion, Robert Minor, Math Working Group Roadmap 2007/8 (http://www.w3.org/Math/Roadmap/)

[sgml-xml] J. Clark; *Comparison of SGML and XML*, W3C Note, December 1997. (http://www.w3.org/TR/NOTE-sgml-xml-971215.html)

[xml11] World Wide Web Consortium *Extensible Markup Language (XML) 1.1*. W3C Recommendation, February 2004 (http://www.w3.org/TR/2004/REC-xml11-20040204/)

Appendix H

Index (Non-Normative)

H.1 MathML Elements

References to sections in which an element is defined are marked in bold.

H.2 MathML Attributes

In addition to the standard MathML attributes, some attributes from other namespaces such as Xlink or XML Schema are also listed here.