Mathematical Markup Language (MathML) Version 3.0 2nd Edition

W3C Recommendation 10 April 2014

This version:

http://www.w3.org/TR/2014/REC-MathML3-20140410/

Latest MathML 3 version:

http://www.w3.org/TR/MathML3/

Latest MathML Recommendation:

http://www.w3.org/TR/MathML/

Previous versions:

http://www.w3.org/TR/2014/PER-MathML3-20140211/
http://www.w3.org/TR/2010/REC-MathML3-20101021/

Editors' version:

http://www.w3.org/Math/draft-spec/

Editors:

David Carlisle, NAG
Patrick Ion, Mathematical Reviews, American Mathematical Society
Robert Miner (deceased), 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 (deceased), Chris Rowley, Murray Sargent, Bruce Smith, Neil Soiffer, Robert Sutor, Stephen Watt

Please refer to the **errata** for this document, which may include some normative corrections.

In addition to the HTML version, this document is also available in these non-normative formats: diff marked HTML version, XHTML+MathML version, single page HTML5+MathML version, and PDF version.

See also **translations**.

Abstract

This specification defines the Mathematical Markup Language, or MathML. MathML is a markup language 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, authors typically will use equation editors, conversion programs, and other specialized software tools to generate MathML. Several versions of such MathML tools exist, both freely available software and commercial products, and more are under development.

MathML was originally specified as an XML application and most of the examples in this specification assume that syntax. Other syntaxes are possible most notably [HTML5] specifies the syntax for MathML in HTML. Unless explictly noted, the examples in this specification are also valid HTML syntax.

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 was produced by the W3C Math Working Group as a Recommendation and is 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). The authors of this document are the W3C Math Working Group members. A list of participants in the W3C Math Working Group is available.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

All reported errata to the first edition have been addressed in this addition, and a full change log appears in Appendix F Changes. The diff-marked version linked in the frontmatter highlights all changes between the first and second editions. In addition to incorporating errata, the main change in this addition is to recognise that MathML parsing is also specified in [HTML5] and where necessary to note where HTML and XML usage differ.

The Working Group maintains a comprehensive Test Suite. This is publicly available and developers are encouraged to submit their results for display. The Test Results are public. They show at least two interoperable implementations for each essential test. Further details may be found in the Implementation Report.

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 did 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. The MathML 3 Specification has also been restructured.

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.

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.

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. Much has been moved to separate documents containing explanatory material, material on characters and entities and on the MathML DOM. The discussion of character entities has led to the document *XML Entity Definitions for Characters* [Entities], which is now a W3C Recommendation. The concern with use of CSS with MathML has led to the document *A MathML for CSS Profile* [MathMLforCSS], which was a W3C Recommendation accompanying MathML 3.0.

The biggest differences from MathML 2.0 (Second Edition) are in Chapters 4 and 5, although there have been smaller improvements throughout the specification. A more detailed description of changes from the previous Recommendation follows.

- 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 to be 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 has been edited to reflect the changes elsewhere in the document, and in the rapidly evolving Web environment. Some of the text in them went back to early days of the Web and XML, and its explanations are now commonplace.
- Chapter 3, on presentation-oriented markup, adds new material on linebreaking, and on markup for elementary math notations used in many countries (`mstack`, `mlongdiv` and other associated elements). Other changes include revisions to the `mglyph`, `mpadded` and `maction` elements and significant unification and cleanup of attribute values. 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. 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 has been 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 *XML Entity Definitions for Characters*, which incorporates those used for mathematics has become a a W3C Recommendation, [Entities].
- 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 appropriately updated.
- A fuller discussion of the document's evolution can be found in Appendix F Changes.

Table of Contents

Appendices

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 notation. As J. R. Pierce writes in his book on communication theory, mathematics and its notation should not be viewed as one and the same thing [Pierce1961]. Mathematical ideas can exist independently of the notation that represents 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 Mathematical 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 notation of written and printed mathematics as well as the new potential for interconnectivity in electronic media.

Mathematical notation evolves constantly as people continue to innovate in ways of approaching and expressing ideas. Even the common notation of arithmetic has 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 and subtle. 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 sixteenth century. Spacing around symbols for operations such as +, -, × 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 numerous conventions such as paragraphs, capital letters, font families and cases, and even the device of decimal-like numbering of sections such as is used 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. Appropriate standards coupled with computers enable a broadening of access to mathematics beyond the world of print. The markup methods for mathematics in use just before the Web rose to prominence importantly included TEX (also written TeX) [Knuth1986] and approaches based on SGML ([AAP-math], [Poppelier1992] and [ISO-12083]).

It is remarkable how widespread the current conventions of mathematical notation 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

MathML has been designed from the beginning with the following ultimate goals in mind.

MathML should ideally:

- Encode mathematical material suitable for all educational and scientific communication.
- 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
  - international print media, including braille

  It is 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 useful only if it is implemented well. The W3C Math Working Group has identified 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 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. 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 a markup language for describing mathematics. It is usually expressed in XML syntax, although HTML and other syntaxes are possible. A special aspect of MathML is that there are two main strains of markup: Presentation markup, discussed in Chapter 3 Presentation Markup, is used to display mathematical expressions; and Content markup, discussed in Chapter 4 Content Markup, is used to convey mathematical meaning. Content markup is specified in particular detail. This specification makes use of an XML format called Content Dictionaries 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 MathML Fundamentals, while the means for combining these strains, as well as external markup, into single MathML objects are discussed in Chapter 5 Mixing Markup Languages for Mathematical Expressions. How MathML interacts with applications is covered in Chapter 6 Interactions with the Host Environment. Finally, a discussion of special symbols, and issues regarding characters, entities and fonts, is given in Chapter 7 Characters, Entities and Fonts.

1.4 A First Example

The quadratic formula provides a simple but instructive illustration of MathML markup.

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

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. An example of this type is given just below. The second flavor emphasizes the mathematical content and an example of it follows the first one.

```
<mrow>
  <mi>x</mi>
  <mo>=</mo>
  <mfrac>
    <mrow>
      <mrow>
        <mo>-</mo>
        <mi>b</mi>
      </mrow>
      <mo>&#xB1;<!--PLUS-MINUS SIGN--></mo>
      <msqrt>
        <mrow>
          <msup>
            <mi>b</mi>
            <mn>2</mn>
          </msup>
          <mo>-</mo>
          <mrow>
            <mn>4</mn>
            <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
            <mi>a</mi>
            <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
            <mi>c</mi>
          </mrow>
        </mrow>
      </msqrt>
    </mrow>
    <mrow>
      <mn>2</mn>
      <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
      <mi>a</mi>
    </mrow>
  </mfrac>
</mrow>
```

Consider the superscript 2 in this formula. It represents the squaring operation here, but the meaning of a superscript in other situations 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/>
  <ci>x</ci>
  <apply>
    <divide/>
    <apply>
      <plus/>
      <apply>
        <minus/>
        <ci>b</ci>
      </apply>
      <apply>
        <root/>
        <apply>
          <minus/>
          <apply>
            <power/>
            <ci>b</ci>
            <cn>2</cn>
          </apply>
          <apply>
            <times/>
            <cn>4</cn>
            <ci>a</ci>
            <ci>c</ci>
          </apply>
        </apply>
      </apply>
    </apply>
    <apply>
      <times/>
      <cn>2</cn>
      <ci>a</ci>
    </apply>
```

```
   </apply>
</apply>
```

2 MathML Fundamentals

2.1 MathML Syntax and Grammar

2.1.1 General Considerations

The basic ‘syntax’ of MathML is defined using XML syntax, but other syntaxes that can encode labeled trees are possible. Notably the HTML parser may also be used with MathML. 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 Parsing MathML.

MathML's character set consists of legal characters as specified by Unicode [Unicode], further restricted by the characters not allowed in XML. The use of Unicode characters for mathematics is discussed in Chapter 7 Characters, Entities and Fonts.

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 when using the XML serialisation of MathML, 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>
```

HTML does not support namespace extensibility in the same way, the HTML parser has in-built knowledge of the HTML, SVG and MathML namespaces. `xmlns` attributes are just treated as normal attributes. Thus when using the HTML serialisation of MathML, prefixed element names must not be used. `xmlns`="http://www.w3.org/1998/Math/MathML" may be used on the `math element`, it will be ignored by the HTML parser, which always places `math` elements and its descendents in the MathML namespace (other than special rules described in Appendix A Parsing MathMLfor invalid input, and for `annotation-xml`. If a MathML expression is likely to be in contexts where it may be parsed by an XML parser or an HTML parser, it SHOULD use the following form to ensure maximum compatibility:

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

2.1.3 Children versus Arguments

Most MathML elements act as ‘containers’; such an element's children are not distinguished from each other except as individual members of the list of children. Commonly there is no limit imposed on the number of children an element may have. This is the case for most presentation elements and some content elements such as `set`. But many MathML elements require a specific number of children, or attach a particular meaning to children in certain positions. Such elements are best considered to represent constructors of mathematical objects, and hence thought of as functions of their children. Therefore children of such a MathML element will often be referred to as its *arguments* instead of merely as children. Examples of this can be found, say, in Section 3.1.3 Required Arguments.

There are presentation elements that conceptually accept only a single argument, but which for convenience have been written to 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 Inferred `<mrow>`s.

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 specified. This information is also tabulated for the presentation elements in Section 3.1.3 Required Arguments.

2.1.4 MathML and Rendering

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

Nevertheless, some parts of this specification describe these recommended visual rendering rules in 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 throughout this document. The meanings of attributes and their allowed values are described within the specification of each element. The syntax notation explained in this section is used in specifying allowed values.

Except when explicitly forbidden by the specification for an attribute, MathML attribute values may contain any legal characters specified by the XML recommendation. See Chapter 7 Characters, Entities and Fonts for further clarification.

2.1.5.1 Syntax notation used in the MathML specification

To describe the MathML-specific syntax of attribute values, the following conventions and notations are used for most attributes in the present document. We use below the notation beginning with U+ that is recommended by Unicode for referring to Unicode characters [see [Unicode], page xxviii].

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 ranges U+0030 to U+0039, U+0041 to U+0046 and U+0061 to U+0066
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	an optional "-" (U+002D), followed by a string of *decimal digit*s, and representing an integer
unsigned-number	a string of *decimal digit*s with up to one decimal point (U+002E), representing a non-negative terminating decimal number (a type of rational number)
number	an optional prefix of "-" (U+002D), followed by an unsigned number, representing a terminating decimal number (a type of rational number)
character	a single non-whitespace character
string	an arbitrary, nonempty and finite, string of *character*s
length	a length, as explained below, Section 2.1.5.2 Length Valued Attributes
unit	a unit, typically used as part of a length, as explained below, Section 2.1.5.2 Length Valued Attributes
namedlength	a named length, as explained below, Section 2.1.5.2 Length Valued Attributes
color	a color, as explained below, Section 2.1.5.3 Color Valued Attributes
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]. Note that the attribute value is typed in the schema as anyURI which allows any sequence of XML characters. Systems needing to use this string as a URI must encode the bytes of the UTF-8 encoding of any characters not allowed in URI using %HH encoding where HH are the byte value in hexadecimal. This ensures that such an attribute value may be interpreted as an IRI, or more generally a LEIRI, see [IRI].
italicized word	values as explained in the text for each attribute; see Section 2.1.5.4 Default values of attributes
"literal"	quoted 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. The whole attribute value must be delimited by single (') or double (") quotation marks in the marked up document. Note that double quotation marks are often used in this specification to mark up literal expressions; an example is the "-" in line 5 of the table above.

In the table below a form *f* means an instance of a type described in the table above. The combining operators are shown in order of precedence from highest to lowest:

Notation	What it matches
(*f*)	same as *f*
f?	an optional instance of *f*
*f**	zero or more instances of *f*, with separating whitespace characters
f+	one or more instances of *f*, with separating whitespace characters
$f_1\ f_2 \ldots f_n$	one instance of each form f_i, in sequence, with no separating whitespace

Notation	What it matches
f_1, f_2, ..., f_n	one instance of each form f_i, in sequence, with separating whitespace characters (but no commas)
f_1 | f_2 | ... | f_n	any one of the specified forms f_i

The notation we have chosen here is in the style of the syntactical notation of the RelaxNG used for MathML's basic schema, Appendix A Parsing MathML.

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).

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 should be no space between the number and the unit of a length.

The possible *unit*s and *namedspace*s, 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 termed pseudo-units and specified in the description of those particular elements; see, for instance, Section 3.3.6 Adjust Space Around Content `<mpadded>`.

A trailing "%" represents a percent of a reference value; unless otherwise stated, the reference value is the default value. The default value, or how it is obtained, is listed in the table of attributes for each element along with the reference value when it differs from the default. (See also Section 2.1.5.4 Default values of attributes.) A number without a unit is intepreted as a multiple of the reference value. This form is primarily for backward compatibility and should be avoided, prefering explicit units for clarity.

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 *unit*s in MathML are:

Unit	Description
`em`	an em (font-relative unit traditionally used for horizontal lengths)

Unit	Description
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 reference value

Some additional aspects of units are discussed further below, in Section 2.1.5.2.1 Additional notes about units.

The following constants, *namedspace*s, may also be used where a length is needed; they are typically used for spacing or padding between tokens. Recommended default values for these constants are shown; the actual spacing used is implementation specific.

namedspace	Recommended default
"veryverythinmathspace"	1/18 em
"verythinmathspace"	2/18 em
"thinmathspace"	3/18 em
"mediummathspace"	4/18 em
"thickmathspace"	5/18 em
"verythickmathspace"	6/18 em
"veryverythickmathspace"	7/18 em
"negativeveryverythinmathspace"	-1/18 em
"negativeverythinmathspace"	-2/18 em
"negativethinmathspace"	-3/18 em
"negativemediummathspace"	-4/18 em
"negativethickmathspace"	-5/18 em
"negativeverythickmathspace"	-6/18 em
"negativeveryverythickmathspace"	-7/18 em

2.1.5.2.1 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 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 Script and Limit Schemata 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` and `scriptlevel`, must be processed before evaluating other length valued attributes.

If, and how, lengths might affect non-visual media is implementation specific.

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, for example, by the digit-doubling rule specified under Colors in [CSS21] `#123` is a short form for `#112233`.

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 a surrounding element 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 recommended MathML visual rendering rules do not define the precise extent of the region whose background is affected by using the `background` attribute on an element, except that, when the element'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 element, 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 Style Change `<mstyle>`, 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 around attribute values in examples in the text, so that the pieces of code shown are correct.

Note that, in general, there is no mechanism in MathML to simulate the effect of not specifying attributes which are *inherited* or *automatic*. Giving the words "inherited" or "automatic" explicitly will not work, and is not generally allowed. Furthermore, the `mstyle` element (Section 3.3.4 Style Change `<mstyle>`) can even be used to change the default values of presentation attributes for its children.

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

In general, there are a number of properties of MathML rendering that may be thought of as overall properties of a document, or at least of sections of a large document. Examples might be `mathsize` (the math font size: see Section 3.2.2 Mathematics style attributes common to token elements), or the behavior in setting limits on operators such as integrals or sums (e.g., `movablelimits` or `displaystyle`), or upon breaking formulas over lines (e.g. `linebreakstyle`); for such attributes see several elements in Section 3.2 Token Elements. These may be thought to be inherited from some such containing scope. Just above we have mentioned the setting of default values of MathML attributes as *inherited* or *automatic*; there is a third source of global default values for behavior in rendering MathML, a MathML operator dictionary. A default example is provided in Appendix C Operator Dictionary. This is also discussed in Section 3.2.5.7.1 The operator dictionary and examples are given in Section 3.2.5.2.1 Dictionary-based attributes.

2.1.6 Attributes Shared by all MathML Elements

In addition to the attributes described specifically for each element, the attributes in the following table are allowed on every MathML element. Also allowed are attributes from the xml namespace, such as `xml:lang`, and attributes from namespaces other than MathML, which are ignored by default.

Name	values	default
`id`	*id*	*none*
	Establishes a unique identifier associated with the element to support linking, cross-references and parallel markup. See `xref` and Section 5.4 Parallel Markup.	
`xref`	*idref*	*none*
	References another element within the document. See `id` and Section 5.4 Parallel Markup.	
`class`	*string*	*none*
	Associates the element with a set of style classes for use with [XSLT] and [CSS21]. Typically this would be a space separated sequence of words, but this is not specified by MathML. See Section 6.5 Using CSS with MathML for discussion of the interaction of MathML and CSS.	
`style`	*string*	*none*
	Associates style information with the element for use with [XSLT] and [CSS21]. This typically would be an inline CSS style, but this is not specified by MathML. See Section 6.5 Using CSS with MathML 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.

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

The attribute `other`, is deprecated (Section 2.3.3 Attributes for unspecified data) in favor of the use of attributes from other namespaces.

Name	values	default
`other`	*string*	*none*
	DEPRECATED but in MathML 1.0.	

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; see also the discussion of whitespace in Section 2.3 of [XML].

MathML ignores whitespace occurring outside token elements. Non-whitespace characters are not allowed there. Whitespace occurring within the content of token elements, except for `<cs>`, is normalized as follows. All whitespace at the beginning and end of the content is removed, and whitespace internal to content of the element 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>` or `<mtext>Theorem&#x20;1:</mtext>`.

Authors wishing to encode white space 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 `&nbsp;` (U+00A0) or other non-marking characters that are not trimmed. For example, compare the above use of an `mtext` element with

```
<mtext>
&#xA0;<!--NO-BREAK SPACE-->Theorem &#xA0;<!--NO-BREAK SPACE-->1:
</mtext>
```

When the first example is rendered, there is nothing before "Theorem", one Unicode space character between "Theorem" and "1:", and nothing after "1:". In the second example, a single space character is to be rendered before "Theorem"; two spaces, one a Unicode space character and one a Unicode no-break space character, are to be rendered before "1:"; and there is nothing after the "1:".

Note that the value of the `xml:space` attribute is not relevant in this situation since XML processors pass whitespace in tokens to a MathML processor; it is the requirements of MathML processing 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`, `cs`, `csymbol` and `annotation`, an `mspace` element should be used, as opposed to an `mtext` element containing only white-space 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; in other words, every valid MathML expression is wrapped in outer `<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 Transferring MathML.

The `math` element can contain an arbitrary number of child elements. They render by default as if they were contained in an `mrow` element.

2.2.1 Attributes

The `math` element accepts any of the attributes that can be set on Section 3.3.4 Style Change `<mstyle>`, including the common attributes specified in Section 2.1.6 Attributes Shared by all MathML Elements. In particular, it accepts the `dir` attribute for setting the overall directionality; the `math` element is usually the most useful place to specify the directionality (See Section 3.1.5 Directionality for further discussion). Note that the `dir` attribute defaults to "ltr" on the `math` element (but *inherits* on all other elements which accept the `dir` attribute); this provides for backward compatibility with MathML 2.0 which had no notion of directionality. Also, it accepts the `mathbackground` attribute in the same sense as `mstyle` and other presentation elements to set the background color of the bounding box, rather than specifying a default for the attribute (see Section 3.1.10 Mathematics style attributes common to presentation elements)

In addition to those attributes, 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 when `display`="inline", `displaystyle` is initialized to "false"; in both cases `scriptlevel` is initialized to 0 (See Section 3.1.6 Displaystyle and Scriptlevel). Moreover, when the math element is embedded in a larger document, a block math element should be treated as a block element as appropriate for the document type (typically as a new vertical block), whereas an inline math element should be treated as inline (typically exactly as if it were a sequence of words in normal text). In particular, this applies to spacing and linebreaking: for instance, there should not be spaces or line breaks inserted between inline math and any immediately following punctuation. When the display attribute is missing, a rendering agent is free to initialize as appropriate to the context.	
`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.	

Name	values	default
`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* | "top" | "middle" | "bottom"	0ex
	specifies the vertical alignment of the image with respect to adjacent inline material. A positive value of `altimg-valign` shifts the bottom of the image above the current baseline, while a negative value lowers it. The keyword "top" aligns the top of the image with the top of adjacent inline material; "center" aligns the middle of the image to the middle of adjacent material; "bottom" aligns the bottom of the image to the bottom of adjacent material (not necessarily the baseline). This attribute only has effect when `display`="inline". 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*
	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 Content Symbols `<csymbol>`. When no `cdgroup` attribute is explicitly specified, the document format embedding this `math` element may provide a method for determining CD bases. Otherwise the system must determine a CD base; in the absence of specific information `http://www.openmath.org/cd` is assumed as the CD base for all `csymbol`, `annotation`, and `annotation-xml` elements. 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 Linebreaking of Expressions for further discussion.
"scroll"	The window provides a viewport into the larger complete display of the mathematical expression. Horizontal or vertical scroll bars are added to the window as necessary to allow the viewport to be moved to a different position.

Value	Meaning
"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.

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.	
`mode`	"display" | "inline"	inline
	specified whether the enclosed MathML expression should be rendered in a display style or an inline style. This attribute is deprecated in favor of the `display` attribute.	

2.3 Conformance

Information nowadays is 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.

This section gives guidelines that describe different types of MathML support and make 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.

2.3.1 MathML Conformance

A valid MathML expression is an XML construct determined by the MathML RelaxNG 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 or produce a valid MathML expression. A MathML processor that both accepts and produces valid MathML expressions may be able to "round-trip" MathML. Perhaps the simplest example of an application that might round-trip a MathML expression would be an editor that writes it to a new file without modifications.

Three forms of MathML conformance are specified:

1. A MathML-input-conformant processor must accept all valid MathML expressions; it should appropriately 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, appropriately representing all application-specific data.
3. A MathML-round-trip-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 relevant 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 recommended rendering rules throughout Chapter 3 Presentation Markup. However, in general, developers are given wide latitude to interpret 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 the W3C 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 3.0 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-round-trip-conformant, a processor need only preserve MathML equivalence on expressions containing no deprecated features.

2.3.1.3 MathML Extension Mechanisms and Conformance

MathML 3.0 defines 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 code point in the future. At that time, representing the character directly by its Unicode code point 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 implementers 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 intended to allow for good presentation and content markup. However it is never possible to cover all users' needs for 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 Bind Action to Sub-Expression `<maction>`), 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.

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, a single row of identifiers and operators will often be represented by multiple nested `mrow` elements rather than a single `mrow`. For example, "$x + a/b$" typically is represented as:

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

Similarly, superscripts are attached to the full expression constituting their base rather than to the just preceding character. This structure permits better-quality rendering of mathematics, especially when details of the rendering environment, such as display widths, are not known ahead of time to the document author. It also greatly eases automatic interpretation of the represented mathematical structures.

Certain characters are used to name identifiers or operators that in traditional notation render the same as other symbols or usually rendered invisibly. For example, the entities `&DifferentialD;`, `&ExponentialE;`, and `&ImaginaryI;` denote notational symbols semantically distinct from visually identical letters used as simple variables. Likewise, the entities `&InvisibleTimes;`, `&ApplyFunction;`, `&InvisibleComma;` and the character U+2064 (INVISIBLE PLUS) usually render invisibly but represent significant information. These entities have distinct spoken renderings, may influence visual linebreaking and spacing, and may effect the evaluation or meaning of particular expressions. Accordingly, authors should use these entities wherever they are applicable. For instance, the expression represented visually as "$f(x)$" would usually be spoken in English as "f of x" rather than just "$f\ x$". MathML conveys this meaning by using 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 MathML Syntax and Grammar 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 Children versus Arguments, and the attribute value notations and conventions described in Section 2.1.5 MathML Attribute Values.

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. *Layout schemata* build expressions out of parts and can have only elements as content (except for whitespace, which they ignore). These are subdivided into General Layout, Script and Limit, Tabular Math and Elementary Math schemata. 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 used to represent text or whitespace that has more aesthetic than mathematical significance and other elements 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 Summary of Presentation Elements. The *content elements* are the MathML elements defined in Chapter 4 Content Markup.

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 Content Markup in Presentation Markup). 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 Space `<mspace/>` 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`, `mpadded`, `mphantom`, `menclose`, `mtd`, `mscarry`, 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.

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. Although the `math` element is not a presentation element, it is listed below for completeness.

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`	0 or more	
`mlongdiv`	3 or more	*divisor result dividend* (*msrow* | *msgroup* | *mscarries* | *msline*)*
`msgroup`	0 or more	
`msrow`	0 or more	
`mscarries`	0 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 Space `<mspace/>`. This definition affects some of the suggested rendering rules for `mo` elements (Section 3.2.5 Operator, Fence, Separator or Accent `<mo>`).

Certain elements, e.g. msup, are able to embellish operators that are their first argument. These elements are listed in Section 3.2.5 Operator, Fence, Separator or Accent <mo>, 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 discussed 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 Top-Level <math> Element). 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.

For example, in a RTL layout, 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 layout details for elements whose behaviour depends on directionality are given in the discussion of the element. In those discussions, the terms leading and trailing are used to specify a side of an object when which side to use depends on the directionality; ie. leading means left in LTR but right in RTL. The terms left and right may otherwise be safely assumed to mean left and right.

The overall directionality is usually set on the math, but may also be switched for individual subformula by using the dir attribute on mrow or mstyle elements. When not specified, all elements inherit the directionality of their 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 Overall Directionality of Mathematics Formulas) and affects how weakly directional characters are treated and how runs are nested. (The dir attribute is thus allowed on token elements to specify the initial directionality that may be needed in rare cases.) Any mglyph or malignmark elements appearing within a token element are effectively *neutral* and have no effect on ordering.

The important thing to notice is that the bidirectional algorithm is applied independently to the contents of each token element; each token element is an independent run of characters.

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, calligraphic 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.

Please note that for the transfinite cardinals represented by Hebrew characters, the code points 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’ formulas, those appearing on a line by themselves, typically make more generous use of vertical space than inline formulas, 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 Style Change `<mstyle>`. 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 Mathematics style attributes common to token elements) 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 Length Valued Attributes) 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 Mathematics style attributes common to token elements), 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" corresponds to \displaystyle, while `display`="inline" corresponds 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 Inferred `<mrow>`s) or `mfenced`. The breaks typically take place at `mo` elements and also, for backwards compatibility, at `mspace`. Renderers may also choose to place automatic linebreaks at other points such as between adjacent `mi` elements or even within a token element such as a very long `mn` element. MathML does not provide a means to specify such linebreaks, but if a render chooses to linebreak at such a point, it should indent the following line according to the indentation attributes that are in effect at that point.

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 line wrapping 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 possibly 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 3.2.5.2.2 Linebreaking attributes.)
- *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 3.2.5.2.2 Linebreaking attributes.)
- *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` elements or inherited from `mstyle` or `math` elements. (See Section 3.2.5.2.3 Indentation attributes.)

When a math element appears in an inline context, it may obey whatever paragraph flow rules are employed by the document's text rendering engine. Such rules are necessarily outside of the scope of this specification. Alternatively, it may use the value of the `math` element's overflow attribute. (See Section 2.2.1 Attributes.)

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 `mrow`s. The more deeply nested the break point, the higher the penalty.
3. Does a linebreak here make layout of the next line difficult? 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 must be considered; i.e., 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 token elements times the number of lines.

3.1.7.3 Linebreaking Algorithm for Inline Expressions (Informative)

A common method for breaking inline expressions that are too long for the space remaining on the current line is to pick an appropriate break point for the expression and place the expression up to that point on the current line and place the remainder of the expression on the following line. This can be done by:

1. Querying the text processing engine for the minimum and maximum amount of space available on the current line.
2. Using a variation of the automatic linebreaking algorithm given above), and/or using hints provided by linebreak attributes on `mo` or `mspace` elements, to choose a line break. The goal is that the first part of the formula fits "comfortably" on the current line while breaking at a point that results in keeping related parts of an expression on the same line.
3. The remainder of the formula begins on the next line, positioned both vertically and horizontally according to the paragraph flow; MathML's indentation attributes are ignored in this algorithm.
4. If the remainder does not fit on a line, steps 1 - 3 are repeated for the second and subsequent lines. Unlike the for the first line, some part of the expression must be placed these lines so that the algorithm terminates.

3.1.8 Warning about fine-tuning of presentation

Some use-cases require precise control of the math layout and presentation. Several MathML elements and attributes expressly support such fine-tuning of the rendering. However, MathML rendering agents exhibit wide variability in their presentation of the same MathML expression due to difference in platforms, font availability, and requirements particular to the agent itself (see Section 3.1 Introduction). The overuse of explicit rendering control may yield 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: non-portability of "tweaking"

For particular expressions, authors may be tempted to use the `mpadded`, `mspace`, `mphantom`, and `mtext` elements to improve ("tweak") the spacing generated by a specific renderer.

Without explicit spacing rules, various MathML renders may use different spacing algorithms. Consequently, different MathML renderers may position symbols in different locations relative to each other. Say that renderer B, for example, provides improved spacing for a particular expression over renderer A. Authors are strongly warned that "tweaking" the layout for renderer A may produce very poor results in renderer B, very likely worse than without any explicit adjustment 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 Alignment Markers `<maligngroup/>`, `<malignmark/>`) may give more portable results than the direct insertion of extra space using `mspace` or `mtext`. Advice given to the implementers 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 `mo`, 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 Annotation Framework 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

Additionally, the `mglyph` element may be used within Token elements to represent non-standard symbols as images.

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
`msline`	horizontal line inside of `mstack`

3.1.9.6 Enlivening Expressions

`maction`	bind actions to a sub-expression

3.1.10 Mathematics style attributes common to presentation elements

In addition to the attributes listed in Section 2.1.6 Attributes Shared by all MathML Elements, all MathML presentation elements accept the following two attributes:

Name	values	default
`mathcolor`	*color*	*inherited*
	Specifies the foreground color to use when drawing the components of this element, such as the content for token elements or any lines, surds, or other decorations. It also establishes the default `mathcolor` used for child elements when used on a layout element.	
`mathbackground`	*color* | "transparent"	transparent
	Specifies the background color to be used to fill in the bounding box of the element and its children. The default, "transparent", lets the background color, if any, used in the current rendering context to show through.	

These style attributes are primarily intended for visual media. They are not expected to affect the intended semantics of displayed expressions, but are for use in highlighting or drawing attention to the affected subexpressions. For example, a red "x" is not assumed to be semantically different than a black "x", in contrast to variables with different `mathvariant` (See Section 3.2.2 Mathematics style attributes common to token elements).

Since MathML expressions are often embedded in a textual data format such as HTML, the MathML renderer should inherit the foreground color used in the context in which the MathML appears. Note, however, that MathML doesn't specify the mechanism by which style information is inherited from the rendering environment. See Section 3.2.2 Mathematics style attributes common to token elements for more details.

Note that the suggested MathML visual rendering rules do not define the precise extent of the region whose background is affected by the `mathbackground` attribute, except that, when the 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, 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 `mathbackground` attribute is not defined by these rules.

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.

Token elements represent identifiers (mi), numbers (mn), operators (mo), text (mtext), strings (ms) and spacing (mspace). The mglyph element may be used *within* token elements to represent non-standard symbols by images. Preceding detailed discussion of the individual elements, the next two subsections discuss the allowable content of token elements and the attributes common to them.

3.2.1 Token Element Content Characters, <mglyph/>

Character data in MathML markup is only allowed to occur as part of the content of token elements. Whitespace between elements is ignored. With the exception of the empty mspace element, token elements can contain any sequence of zero or more Unicode characters, or mglyph or malignmark elements. The mglyph element is used to represent non-standard characters or symbols by images; the malignmark element establishes an alignment point for use within table constructs, and is otherwise invisible (See Section 3.5.5 Alignment Markers <maligngroup/>, <malignmark/>).

Characters can be either represented directly as Unicode character data, or indirectly via numeric or character entity references. See Chapter 7 Characters, Entities and Fonts 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. Also, see Section 7.7 Anomalous Mathematical Characters for a discussion of the appropriate character content to choose for certain applications.

Token elements (other than mspace) should be rendered as their content, if any, (i.e. in the visual case, as a closely-spaced horizontal row of standard glyphs for the characters or images for the mglyphs in their content). An mspace element is rendered as a blank space of a width determined by its attributes. 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. The directional characteristics of the content must also be respected (see Section 3.1.5.2 Bidirectional Layout in Token Elements).

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 mathvariant attribute is described in more detail in Section 3.2.2 Mathematics style attributes common to token elements, and a complete technical description of the corresponding characters is given in Section 7.5 Mathematical Alphanumeric Symbols.

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

3.2.1.2.1 Description

The `mglyph` element provides a mechanism for displaying images to represent non-standard symbols. It may be used within the content of the token elements `mi`, `mn`, `mo`, `mtext` or `ms` 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 `mglyph`s with the same values for `src` should be considered identical by applications that must determine whether two characters/glyphs are identical.

3.2.1.2.2 Attributes

The `mglyph` element accepts the attributes listed in Section 3.1.10 Mathematics style attributes common to presentation elements, but note that `mathcolor` has no effect. 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.	
`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*	0ex
	Specifies the baseline alignment point of the image with respect to the current baseline. A positive value shifts the bottom of the image above the current baseline while a negative value lowers it. A value of 0 (the default) means that the baseline of the image is at the bottom of the image.	
`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.	

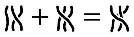
Note that the `src` and `alt` attributes are *required* for correct usage in MathML 3, however this is not enforced by the schema due to the deprecated usage described below.

3.2.1.2.3 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="2 3 braid"/></mi>
  <mo>+</mo>
  <mi><mglyph src="my-braid-132" alt="1 3 2 braid"/></mi>
  <mo>=</mo>
  <mi><mglyph src="my-braid-13" alt="1 3 braid"/></mi>
</mrow>
```

This might render as:



3.2.1.2.4 Deprecated Attributes

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 attributes were previously defined:

Name	values
`fontfamily`	*string*
	the name of a font that may be available to a MathML renderer, or a CSS font specification; See Section 6.5 Using CSS with MathML and CSS [CSS21] for more information.
`index`	*integer*
	Specified a position of the desired glyph within the font named by the `fontfamily` attribute (see Section 3.2.2.1 Deprecated style attributes on token elements).

In MathML 1 and 2, both were required attributes; they are now optional and should be ignored unless the `src` attribute is missing.

Additionally, in MathML 2, `mglyph` accepted the attributes described in Section 3.2.2 Mathematics style attributes common to token elements (`mathvariant` and `mathsize`, along with the attributes deprecated there); to make clear that `mglyph` is *not* a token element, and since these attributes have no effect in any case, these attributes have been deprecated.

3.2.2 Mathematics style attributes common to token elements

In addition to the attributes defined for all presentation elements (Section 3.1.10 Mathematics style attributes common to presentation elements), MathML includes two *mathematics style* attributes as well as a directionality attribute valid on all presentation token elements, as well as the `math` and `mstyle` elements; `dir` is also valid on `mrow` elements. 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" |	normal (*except on* `<mi>`)

<table>
<tr><th>Name</th><th>values</th><th>default</th></tr>
<tr><td></td><td>"sans-serif-bold-italic" | "monospace" | "initial" | "tailed" | "looped" | "stretched"</td><td></td></tr>
<tr><td></td><td colspan="2">Specifies the logical class of the token. Note that this class is more than styling, it typically conveys semantic intent; see the discussion below.</td></tr>
<tr><td>`mathsize`</td><td>"small" | "normal" | "big" | *length*</td><td>*inherited*</td></tr>
<tr><td></td><td colspan="2">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.</td></tr>
<tr><td>`dir`</td><td>"ltr" | "rtl"</td><td>*inherited*</td></tr>
<tr><td></td><td colspan="2">specifies the initial directionality for text within the token: `ltr` (Left To Right) or `rtl` (Right To Left). This attribute should only be needed in rare cases involving weak or neutral characters; see Section 3.1.5.1 Overall Directionality of Mathematics Formulas for further discussion. It has no effect on `mspace`.</td></tr>
</table>

The `mathvariant` attribute defines logical classes of token elements. Each class provides a collection of typographically-related symbolic tokens. Each token has a specific meaning within a given mathematical expression and, therefore, needs to be visually distinguished and protected from inadvertent document-wide style changes which might change its meaning. Each token is identified by the combination of the `mathvariant` attribute value and the character data in the token element.

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 Using CSS with MathML 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 Identifier `<mi>` 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.

In principle, any `mathvariant` value may be used with any character data to define a specific symbolic token. In practice, only certain combinations of character data and `mathvariant` values will be visually distinguished by a given renderer. For example, there is no clear-cut rendering for a "fraktur alpha" or a "bold italic Kanji" character, and the `mathvariant` values "initial", "tailed", "looped", and "stretched" are appropriate only for Arabic characters.

Certain combinations of character data and `mathvariant` values are equivalent to assigned Unicode code points that encode mathematical alphanumeric symbols. These Unicode code points are the ones in the Arabic Mathematical Alphabetic Symbols block U+1EE00 to U+1EEFF, Mathematical Alphanumeric Symbols block U+1D400 to U+1D7FF, listed in the Unicode standard, and the ones in the Letterlike Symbols range U+2100 to U+214F that represent "holes" in the alphabets in the SMP, listed in Section 7.5 Mathematical Alphanumeric Symbols. These characters are described in detail in section 2.2 of UTR #25. The description of each such character in the Unicode standard provides an unstyled character to which it would be equivalent except for a font change that corresponds to a `mathvariant` value. A token element that uses the unstyled character in combination with the corresponding `mathvariant` value is equivalent to a token element that uses the mathematical alphanumeric symbol character without the mathvariant attribute. Note that the appearance of a mathematical alphanumeric symbol character should not be altered by surrounding mathvariant or other style declarations.

Renderers should support those combinations of character data and mathvariant values that correspond to Unicode characters, and that they can visually distinguish using available font characters. Renderers may ignore or support those combinations of character data and mathvariant values that do not correspond to an assigned Unicode code point, and authors should recognize that support for mathematical symbols that do not correspond to assigned Unicode code points may vary widely from one renderer to another.

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 Displaystyle and Scriptlevel.

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 Mathematics style attributes common to token elements), 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 Using CSS with MathML and CSS [CSS21] 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.	
background	*color* | "transparent"	transparent
	Specified the background color to be used to fill in the bounding box of the element and its children. Deprecated in favor of mathbackground.	

3.2.2.2 Embedding HTML in MathML

MathML can be combined with other formats as described in Section 6.4 Combining MathML and Other Formats. The recommendation is to embed other formats in MathML by extending the MathML schema to allow additional elements to be children of the `mtext` element or other leaf elements as appropriate to the role they serve in the expression (see Section 3.2.6.4 Mixing text and mathematics). The directionality, font size, and other font attributes should inherit from those that would be used for characters of the containing leaf element (see Section 3.2.2 Mathematics style attributes common to token elements).

Here is an example of embedding SVG inside of `mtext` in an HTML context:

```
<mtable>
  <mtr>
    <mtd>
      <mtext><input type="text" placeholder="what shape is this?"/></mtext>
    </mtd>
  </mtr>
  <mtr>
    <mtd>
      <mtext>
        <svg xmlns="http://www.w3.org/2000/svg"
             width="4cm" height="4cm" viewBox="0 0 400 400">
          <rect x="1" y="1" width="398" height="398"
                style="fill:none; stroke:blue"/>
          <path d="M 100 100 L 300 100 L 200 300 z"
                style="fill:red; stroke:blue; stroke-width:3"/>
        </svg>
      </mtext>
    </mtd>
  </mtr>
</mtable>
```

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 its content (See Section 3.2.1 Token Element Content Characters, `<mglyph/>`), with no extra spacing around it (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 Mixing text and mathematics.

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 as 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 Content Markup.

3.2.3.2 Attributes

`mi` elements accept the attributes listed in Section 3.2.2 Mathematics style attributes common to token elements, 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 Mathematical Alphanumeric Symbols and Section 3.2.1.1 Alphanumeric symbol characters) 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 Operator, Fence, Separator or Accent `<mo>`.

```
<mrow>
  <mi> sin </mi>
  <mo> &#x2061;<!--FUNCTION APPLICATION--> </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> &#x2026;<!--HORIZONTAL ELLIPSIS--> </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> &#x3C0;<!--GREEK SMALL LETTER PI--> </mi>
<mi> &#x2148;<!--DOUBLE-STRUCK ITALIC SMALL I--> </mi>
<mi> &#x2147;<!--DOUBLE-STRUCK ITALIC SMALL E--> </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 its content (See Section 3.2.1 Token Element Content Characters, <mglyph/>), 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 be 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 Content Markup.

3.2.4.2 Attributes

mn elements accept the attributes listed in Section 3.2.2 Mathematics style attributes common to token elements.

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> &#x2062;<!--INVISIBLE TIMES--> </mo>
    <mi> &#x2148;<!--DOUBLE-STRUCK ITALIC SMALL I--> </mi>
  </mrow>
</mrow>
<mfrac> <mn> 1 </mn> <mn> 2 </mn> </mfrac>
<mi> &#x3C0;<!--GREEK SMALL LETTER PI--> </mi>
<mi> &#x2147;<!--DOUBLE-STRUCK ITALIC SMALL E--> </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 content (See Section 3.2.1 Token Element Content Characters, `<mglyph/>`), 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 to render an `mo` element as not precisely the characters in its content in some cases. For example, `<mo> &le; </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> &le; </mo>` and `<mo> &lt;= </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 <=.

All 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 Text `<mtext>`. See also Chapter 4 Content Markup 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 Linebreaking of Expressions, 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 Mathematics style attributes common to token elements 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, from the containing document, or from the Section 3.2.5.7.1 The operator dictionary. 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.

3.2.5.2.1 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 3.2.5.7.2 Default value of the form attribute.	
`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 3.2.5.7.5 Spacing around an operator. (Note that before is on the right in a RTL context; see Section 3.1.5 Directionality).	
`rspace`	*length*	*set by dictionary* (thickmathspace)
	Specifies the trailing space appearing after the operator; see Section 3.2.5.7.5 Spacing around an operator. (Note that after is on the left in a RTL context; see Section 3.1.5 Directionality).	
`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 Stretching of operators, fences and accents.	
`symmetric`	"true" | "false"	*set by dictionary* (false)
	Specifies whether the operator should be kept symmetric around the math axis when stretchy. Note this property only applies to vertically stretched symbols. See Section 3.2.5.8 Stretching of operators, fences and accents.	
`maxsize`	*length* | "infinity"	*set by dictionary* (infinity)
	Specifies the maximum size of the operator when stretchy; see Section 3.2.5.8 Stretching of operators, fences and accents. Unitless or percentage values indicate a multiple of the reference size, being the size of the unstretched glyph.	

Name	values	default
`minsize`	*length*	*set by dictionary* (100%)
	Specifies the minimum size of the operator when stretchy; see Section 3.2.5.8 Stretching of operators, fences and accents. Unitless or percentage values indicate a multiple of the reference size, being the size of the unstretched glyph.	
`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 Displaystyle and Scriptlevel 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 Displaystyle and Scriptlevel 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`.	

3.2.5.2.2 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* (before)
	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).	

Name	values	default
`linebreakmultchar`	*string*	*inherited* (&InvisibleTimes;)
	Specifies the character used to make an &InvisibleTimes; operator visible at a linebreak. For example, `linebreakmultchar`="&#xB7;" would make the multiplication visible as a center dot.	

`linebreak` values on adjacent `mo` and `mspace`elements do not interact; `linebreak`="nobreak" on a `mo` does not, in itself, inhibit a break on a preceding or following (possibly nested) `mo` or `mspace` element and does not interact with the `linebreakstyle` attribute value of the preceding or following `mo` element. It does prevent breaks from occurring on either side of the `mo` element in all other situations.

3.2.5.2.3 Indentation attributes

The following attributes affect indentation of the lines making up a formula. Primarily these attributes control the positioning of new lines following a linebreak, whether automatic or manual. However, `indentalignfirst` and `indentshiftfirst` also control the positioning of single line formula without any linebreaks. When these attributes appear on `mo` or `mspace` they apply if a linebreak occurs at that element. When they appear on `mstyle` or `math` elements, they determine defaults for the style to be used for any linebreaks occurring 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 the available 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
`indentalign`	"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.	
`indentshift`	*length*	*inherited* (0)
	Specifies an additional indentation offset relative to the position determined by `indentalign`. When the value is a percentage value or number without unit, the value is relative to the horizontal space that a MathML renderer has available, this is the current target width as used for linebreaking as specified in Section 3.1.7 Linebreaking of Expressions	
`indenttarget`	*idref*	*inherited (none)*
	Specifies the id of another element whose horizontal position determines the position of indented lines when `indentalign`="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 or in a position that does not allow for alignment. In such cases, the `indentalign` should revert to "auto".	
`indentalignfirst`	"left" | "center" | "right" | "auto" | "id" | "indentalign"	*inherited* (indentalign)
	Specifies the indentation style to use for the first line of a formula; the value "indentalign" (the default) means to indent the same way as used for the general line.	

Name	values	default
`indentshiftfirst`	*length* | "indentshift"	*inherited* (indentshift)
	Specifies the offset to use for the first line of a formula; the value "indentshift" (the default) means to use the same offset as used for the general line. Percentage values and numbers without unit are interpreted as described for `indentshift`	
`indentalignlast`	"left" | "center" | "right" | "auto" | "id" | "indentalign"	*inherited* (indentalign)
	Specifies the indentation style to use for the last line when a linebreak occurs within a given `mrow`; the value "indentalign" (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 `indentalign`.	
`indentshiftlast`	*length* | "indentshift"	*inherited* (indentshift)
	Specifies the offset to use for the last line when a linebreak occurs within a given `mrow`; the value "indentshift" (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 `indentshift`. Percentage values and numbers without unit are interpreted as described for `indentshift`	

The legal values of indentalign 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 `indentalign` value

3.2.5.3 Examples with ordinary operators

```
<mo> + </mo>
```

```
<mo> &lt; </mo>
```

```
<mo> &#x2264;<!--LESS-THAN OR EQUAL TO--> </mo>
```

```
<mo> &lt;= </mo>
```

```
<mo> ++ </mo>
```

```
<mo> &#x2211;<!--N-ARY SUMMATION--> </mo>
```

```
<mo> .NOT. </mo>
```

```
<mo> and </mo>
```

```
<mo> &#x2062;<!--INVISIBLE TIMES--> </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 Expression Inside Pair of Fences `<mfenced>`.

$(a+b)$

```
<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> &#x2061;<!--FUNCTION APPLICATION--> </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

Character	Entity name	Short name	Examples of use
U+2063	`&InvisibleComma;`	`&ic;`	m_{12}
U+2064			$2\frac{3}{4}$

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

```
<mrow>
  <mi> f </mi>
  <mo> &#x2061;<!--FUNCTION APPLICATION--> </mo>
  <mrow>
    <mo> ( </mo>
    <mi> x </mi>
    <mo> ) </mo>
  </mrow>
</mrow>
```

```
<mrow>
  <mi> sin </mi>
  <mo> &#x2061;<!--FUNCTION APPLICATION--> </mo>
  <mi> x </mi>
</mrow>
```

```
<mrow>
  <mi> x </mi>
  <mo> &#x2062;<!--INVISIBLE TIMES--> </mo>
  <mi> y </mi>
</mrow>
```

```
<msub>
  <mi> m </mi>
  <mrow>
    <mn> 1 </mn>
    <mo> &#x2063;<!--INVISIBLE SEPARATOR--> </mo>
    <mn> 2 </mn>
  </mrow>
</msub>
```

```
<mrow>
  <mn> 2 </mn>
  <mo> &#x2064;<!--INVISIBLE PLUS--> </mo>
  <mfrac>
    <mn> 3 </mn>
    <mn> 4 </mn>
  </mfrac>
</mrow>
```

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.

3.2.5.7.1 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 Operator Dictionary) 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.

3.2.5.7.2 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 with more than one argument (ignoring all space-like arguments (see Section 3.2.7 Space <mspace/>) 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 with more than one argument (ignoring all space-like arguments), the postfix form is used;
- if it is the only element in an implicit or explicit mrow and if it is in a script position of one of the elements listed in Section 3.4 Script and Limit Schemata, 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 Inferred `<mrow>`s.

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`.

3.2.5.7.3 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 Annotation Framework), 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.

3.2.5.7.4 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> &#x2146;<!--DOUBLE-STRUCK ITALIC SMALL D--> </mo>
  <mrow>
    <mo> &#x2146;<!--DOUBLE-STRUCK ITALIC SMALL D--> </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 Horizontally Group Sub-Expressions `<mrow>`). 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.

3.2.5.7.5 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 and reasonable for that character. It is up to the renderer to know in which directions it is reasonable to stretch a character, if it can stretch the 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=`"300%", 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> &#x5F;<!--LOW LINE--> </mo>
  </munder>
  <mfrac>
    <mi> a </mi>
    <mi> b </mi>
  </mfrac>
  <mover>
    <mo> ) </mo>
    <mo> &#x203E;<!--OVERLINE--> </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.

3.2.5.8.1 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="100%"> ( </mo>
  <mfrac>
    <mi> a </mi> <mi> b </mi>
  </mfrac>
  <mo maxsize="100%"> ) </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`="100%", they would render with different sizes.

3.2.5.8.2 Vertical Stretching Rules

The general rules governing stretchy operators are:

- 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.
- The preceding rules also apply in situations where the mrow element is inferred.
- The rules for symmetric stretching only apply if symmetric="true" and if the stretching occurs in an mrow or in an mtr whose rowalign value is either "baseline" or "axis".

The following algorithm specifies the height and depth of vertically stretched characters:

1. Let maxheight and maxdepth be the maximum height and depth of the *non*-stretchy siblings within the same mrow or mtr. Let axis be the height of the math axis above the baseline.

   Note that even if a minsize or maxsize value is set on a stretchy operator, it is *not* used in the initial calculation of the maximum height and depth of an mrow.
2. If symmetric="true", then the computed height and depth of the stretchy operator are:

   ```
   height=max(maxheight-axis, maxdepth+axis) + axis
   depth =max(maxheight-axis, maxdepth+axis) - axis
   ```

   Otherwise the height and depth are:

   ```
   height= maxheight
   depth = maxdepth
   ```
3. If the total size = height+depth is less than minsize or greater than maxsize, increase or decrease both height and depth proportionately so that the effective size meets the constraint.

By default, most vertical arrows, along with most opening and closing fences are defined in the operator dictionary to stretch 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 Attributes for details about the rowspan attribute.

3.2.5.8.3 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.
- 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 Attributes 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> &#x2192;<!--RIGHTWARDS ARROW--> </mo>
    <mtext> maps to </mtext>
  </munder>
  <mi> y </mi>
</mrow>
```

This displays as $x \xrightarrow[\text{maps to}]{} y$.

3.2.5.8.4 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 that, then the specified (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>&#x2061;<!--FUNCTION APPLICATION--></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'
                    indentalign='id' indenttarget='eq1-equals'>=</mo>
  <mrow>
```

```
   <msup> <mi>x</mi> <mn>4</mn> </msup>
   <mo id='eq1-plus'>+</mo>
   <mrow>  <mn>4</mn> <mo>&#x2062;<!--INVISIBLE TIMES--></mo> <msup> <mi>x</mi>
                                                 <mn>3</mn> </msup>  </mrow>
   <mo>+</mo>
   <mrow>  <mn>6</mn> <mo>&#x2062;<!--INVISIBLE TIMES--></mo> <msup> <mi>x</mi>
                                                 <mn>2</mn> </msup>  </mrow>

   <mo linebreak='newline' linebreakstyle='before'
                     indentalignlast='id' indenttarget='eq1-plus'>+</mo>
   <mrow>  <mn>4</mn> <mo>&#x2062;<!--INVISIBLE TIMES--></mo> <mi>x</mi>  </mrow>
   <mo>+</mo>
   <mn>1</mn>
  </mrow>
 </mrow>
</mrow>
```

This displays as

$$\begin{aligned} f(x) &= (x+1)^4 \\ &= x^4 + 4x^3 + 6x^2 \\ &\qquad + 4x + 1 \end{aligned}$$

Note that because `indentalignlast` defaults to "indentalign", in the above example `indentalign` could have been used in place of `indentalignlast`. 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 Collapsing Whitespace in Input. 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 Mathematics style attributes common to token elements.

See also the warnings about the legal grouping of "space-like elements" in Section 3.2.7 Space `<mspace/>`, and about the use of such elements for "tweaking" in Section 3.1.8 Warning about fine-tuning of presentation.

3.2.6.3 Examples

```
<mtext> Theorem 1: </mtext>
<mtext> &#x2009;<!--THIN SPACE--> </mtext>
<mtext> &#x205F;&#x200A;<!--space of width 5/18 em-->&#x205F;&#x200A;<!--space of width 5/18 em-->
                                                                                  </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> &#x3B4;<!--GREEK SMALL LETTER DELTA--> </mi>
      <mo> &gt; </mo>
      <mn> 0 </mn>
    </mrow>
    <mo> such that </mo>
    <mrow>
      <mrow>
        <mi> f </mi>
        <mo> &#x2061;<!--FUNCTION APPLICATION--> </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 Identifier `<mi>`).

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 Warning about fine-tuning of presentation. 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 Mathematics style attributes common to token elements, but note that `mathvariant` and `mathcolor` have no effect and that `mathsize` only affects the interpretation of units in sizing attributes (see Section 2.1.5.2 Length Valued Attributes). `mspace` also accepts the indentation attributes described in Section 3.2.5.2.3 Indentation attributes.

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. This attribute should be ignored if any dimensional attribute is set.	

Linebreaking was originally specified on `mspace` in MathML2, but controlling linebreaking on `mo` is to be preferred starting with MathML 3. MathML 3 adds new linebreaking attributes only to `mo`, not `mspace`. However, because a linebreak can be specified on `mspace`, control over the indentation that follows that break can be specified using the attributes listed in Section 3.2.5.2.3 Indentation attributes.

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.

3.2.7.3 Examples

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

<mrow>
  <mi>a</mi>
  <mo id="firstop">+</mo>
  <mi>b</mi>
  <mspace linebreak="newline" indentalign="id" indenttarget="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 white-space, 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 Operator, Fence, Separator or Accent `<mo>`. 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 Warning about fine-tuning of presentation.

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 Text `<mtext>`, 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 Collapsing Whitespace in Input, 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 Mathematics style attributes common to token elements, and additionally:

Name	values	default
`lquote`	*string*	`&quot;`
	Specifies the opening quote to enclose the content. (not necessarily ‘left quote’ in RTL context).	
`rquote`	*string*	`&quot;`
	Specifies the closing quote to enclose the content. (not necessarily ‘right quote’ in RTL context).	

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 `mrow`s in Section 3.1.3 Required Arguments. See also `mfenced` (Section 3.3.8 Expression

Inside Pair of Fences `<mfenced>`), 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 within a context with LTR directionality, or right to left within a context with RTL directionality. 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 Operator, Fence, Separator or Accent `<mo>` 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 `mrow`s 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 `mrow`s, whether they are explicitly marked up in the document, or inferred (See Section 3.1.3.1 Inferred `<mrow>`s), although the control of linebreaking is effected through attributes on other elements (See Section 3.1.7 Linebreaking of Expressions).

3.3.1.2 Attributes

`mrow` elements accept the attribute listed below in addition to those listed in Section 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
`dir`	"ltr" | "rtl"	*inherited*
	specifies the overall directionality `ltr` (Left To Right) or `rtl` (Right To Left) to use to layout the children of the row. See Section 3.1.5.1 Overall Directionality of Mathematics Formulas for further discussion.	

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.

3.3.1.3.1 `<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. If there are attributes on the `mrow` element, 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 Operator, Fence, Separator or Accent

`<mo>` and Section 3.2.7 Space `<mspace/>`. See also the discussion of equivalence of MathML expressions in Section 2.3 Conformance.

3.3.1.3.2 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, 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 Operator Dictionary). In all other cases, nested `mrow`s 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 Operator, Fence, Separator or Accent `<mo>`.)

3.3.1.4 Examples

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

```
<mrow>
  <mrow>
    <mn> 2 </mn>
    <mo> &#x2062;<!--INVISIBLE TIMES--> </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 `mrow`s:

```
<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 Displaystyle and Scriptlevel.)

3.3.2.2 Attributes

`mfrac` elements accept the attributes listed below in addition to those listed in Section 3.1.10 Mathematics style attributes common to presentation elements. The fraction line, if any, should be drawn using the color specified by `mathcolor`.

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 (e.g. `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:

$$\genfrac{(}{)}{0pt}{}{a}{b}\genfrac{}{}{2pt}{}{\frac{a}{b}}{\frac{c}{d}}$$

An example illustrating the bevelled form is shown 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 (See Section 3.1.5.1 Overall Directionality of Mathematics Formulas for clarification). 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="200%">
   <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 Required Arguments. 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 Displaystyle and Scriptlevel.)

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 3.1.10 Mathematics style attributes common to presentation elements. The surd and overbar should be drawn using the color specified by `mathcolor`.

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. Firstly, as a presentation element, it accepts the attributes described in Section 3.1.10 Mathematics style attributes common to presentation elements. Additionally, it can be given any attribute accepted by any other presentation element, except for the attributes described below. Finally, the `mstyle` element can be given certain special attributes listed in the next subsection.

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

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, unless 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 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 descendant element in the mstyle's content only if that attribute is not given a value by the descendant element. On any element for which the attribute is set explicitly, the value specified overrides the inherited value. The only exception to this rule is when the attribute value 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 descendants also setting it. Thus it never matters for attributes, such as mathsize, which can only be set on token elements (or on mstyle itself).

MathML specifies that when the attributes height, depth or width are specified on an mstyle element, they apply only to mspace elements, and not to the corresponding attributes of mglyph, mpadded, or mtable. Similarly, when rowalign, columnalign, or groupalign are specified on an mstyle element, they apply only to the mtable element, and not the mtr, mlabeledtr, mtd, and maligngroup elements. When the lspace attribute is set with mstyle, it applies only to the mo element and not to mpadded. To be consistent, the voffset attribute of the mpadded element can not be set on mstyle. When the deprecated fontfamily attribute is specified on an mstyle element, it does not apply to the mglyph element. The deprecated index attribute cannot be set on mstyle. When the align attribute is set with mstyle, it applies only to the munder, mover, and munderover elements, and not to the mtable and mstack elements. The required attributes src and alt on mglyph, and actiontype on maction, cannot be set on mstyle.

As a presentation element, mstyle directly accepts the mathcolor and mathbackground attributes. Thus, the mathbackground specifies the color to fill the bounding box of the mstyle element itself; it does *not* specify the default background color. This is an incompatible change from MathML 2, but we feel it is more useful and intuitive. Since the default for mathcolor is inherited, this is no change in its behaviour.

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 3.1.10 Mathematics style attributes common to presentation elements.

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 and Scriptlevel.	
`displaystyle`	"true" | "false"	*inherited*
	Changes the `displaystyle` in effect for the children. See Section 3.1.6 Displaystyle and Scriptlevel.	
`scriptsizemultiplier`	*number*	0.71
	Specifies the multiplier to be used to adjust font size due to changes in `scriptlevel`. See Section 3.1.6 Displaystyle and Scriptlevel.	
`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 Displaystyle and Scriptlevel.	
`infixlinebreakstyle`	"before" | "after" | "duplicate"	before
	Specifies the default linebreakstyle to use for infix operators; see Section 3.2.5.2.2 Linebreaking attributes	
`decimalpoint`	*character*	.
	specifies the character used to determine the alignment point within `mstack` and `mtable` columns when the "decimalpoint" value is used to specify the alignment. The default, ".", is the decimal separator used to separate the integral and decimal fractional parts of floating point numbers in many countries. (See Section 3.6 Elementary Math and Section 3.5.5 Alignment Markers `<maligngroup/>`, `<malignmark/>`).	

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`.

3.3.4.2.1 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

Name	values	default
`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="100%"> ( </mo>
   <mfrac> <mi> a </mi> <mi> b </mi> </mfrac>
   <mo maxsize="100%"> ) </mo>
</mrow>
```

can be rewritten using `mstyle` as:

```
<mstyle maxsize="100%">
   <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 Required Arguments.

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 Handling of Errors.

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 3.1.10 Mathematics style attributes common to presentation elements.

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 child content, but with the size of the child's bounding box and the relative positioning point of its content modified according to `mpadded`'s attributes. It does not rescale (stretch or shrink) its content. The name of the element reflects the typical use of `mpadded` to add padding, or extra space, around its content. However, `mpadded` can be used to make more general adjustments of size and positioning, and some combinations, e.g. negative padding, can cause the content of `mpadded` to overlap the rendering of neighboring content. See Section 3.1.8 Warning about fine-tuning of presentation for warnings about several potential pitfalls of this effect.

The `mpadded` element accepts a single argument which may be an inferred `mrow` of multiple children; see Section 3.1.3 Required Arguments.

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 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
`height`	("+" | "-")? *unsigned-number* (("%" *pseudo-unit*?) | *pseudo-unit* | *unit* | *namedspace*)?	*same as content*
	Sets or increments the height of the `mpadded` element. See below for discussion.	

Name	values	default
`depth`	("+" | "-")? *unsigned-number* (("%" *pseudo-unit*?) | *pseudo-unit* | *unit* | *namedspace*)?	*same as content*
	Sets or increments the depth of the `mpadded` element. See below for discussion.	
`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*)?	0em
	Sets the horizontal position of the child content. See below for discussion.	
`voffset`	("+" | "-")? *unsigned-number* (("%" *pseudo-unit*?) | *pseudo-unit* | *unit* | *namedspace*)?	0em
	Sets the vertical position of the child content. See below for discussion.	

The *pseudo-unit* syntax symbol is described below. Also, `height`, `depth` and `width` attributes are referred to as size attributes, while `lspace` and `voffset` attributes are position attributes.

These attributes specify the size of the bounding box of the `mpadded` element relative to the size of the bounding box of its child content, and specify the position of the child content of the `mpadded` element relative to the natural positioning of the `mpadded` element. The typographical layout parameters determined by these attributes are described in the next subsection. Depending on the form of the attribute value, a dimension may be set to a new value, or specified relative to the child content's corresponding dimension. Values may be given as multiples or percentages of any of the dimensions of the normal rendering of the child content using so-called pseudo-units, or they can be set directly using standard units Section 2.1.5.2 Length Valued Attributes.

If the value of a size attribute begins with a `+` or `-` sign, it specifies an *increment* or *decrement* to the corresponding dimension by the following length value. Otherwise the corresponding dimension is set directly to the following length value. Note that since a leading minus sign indicates a decrement, the size attributes (`height`, `depth`, `width`) cannot be set directly to negative values. In addition, specifying a decrement that would produce a net negative value for these attributes has the same effect as setting the attribute to zero. In other words, the effective bounding box of an `mpadded` element always has non-negative dimensions. However, negative values are allowed for the relative positioning attributes `lspace` and `voffset`.

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 *unit* (excepting `%`). The possible *pseudo-units* are the keywords `height`, `depth`, and `width`. They represent the length of the same-named dimension of the `mpadded` element's child content.

For any of these length formats, the resulting length is the product of the number (possibly including the `%`) and the following *pseudo-unit*, *unit*, *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="1.0height"` 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="-0height"> ... </mpadded>
<mpadded width="100%"> ... </mpadded>
<mpadded width="100%width"> ... </mpadded>
<mpadded width="1width"> ... </mpadded>
<mpadded width="1.0width"> ... </mpadded>
<mpadded> ... </mpadded>
```

Note that the examples in the Version 2 of the MathML specification showed spaces within the attribute values, suggesting that this was the intended format. Formally, spaces are not allowed within these values, but implementers may wish to ignore such spaces to maximize backward compatibility.

3.3.6.3 Meanings of size and position attributes

See Appendix D Glossary for definitions of some of the typesetting terms used here.

The content of an mpadded element defines a fragment of mathematical notation, such as a character, fraction, or expression, that can be regarded as a single typographical element with a natural positioning point relative to its natural bounding box.

The size of the bounding box of an mpadded element is defined as the size of the bounding box of its content, except as modified by the mpadded element's height, depth, and width attributes. The natural positioning point of the child content of the mpadded element is located to coincide with the natural positioning point of the mpadded element, except as modified by the lspace and voffset attributes. Thus, the size attributes of mpadded can be used to expand or shrink the apparent bounding box of its content, and the position attributes of mpadded can be used to move the content relative to the bounding box (and hence also neighboring elements). Note that MathML doesn't define the precise relationship between "ink", bounding boxes and positioning points, which are implementation specific. Thus, absolute values for mpadded attributes may not be portable between implementations.

The height attribute specifies the vertical extent of the bounding box of the mpadded element above its baseline. Increasing the height increases the space between the baseline of the mpadded element and the content above it, and introduces padding above the rendering of the child content. Decreasing the height reduces the space between the baseline of the mpadded element and the content above it, and removes space above the rendering of the child content. Decreasing the height may cause content above the mpadded element to overlap the rendering of the child content, and should generally be avoided.

The depth attribute specifies the vertical extent of the bounding box of the mpadded element below its baseline. Increasing the depth increases the space between the baseline of the mpadded element and the content below it, and introduces padding below the rendering of the child content. Decreasing the depth reduces the space between the baseline of the mpadded element and the content below it, and removes space below the rendering of the child content. Decreasing the depth may cause content below the mpadded element to overlap the rendering of the child content, and should generally be avoided.

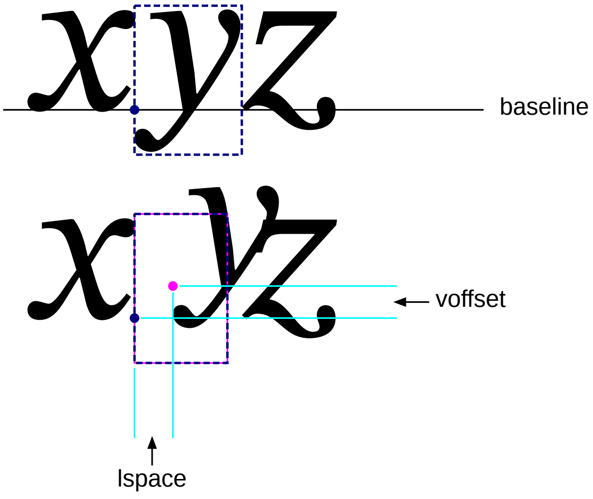
The width attribute specifies the horizontal distance between the positioning point of the mpadded element and the positioning point of the following content. Increasing the width increases the space between the positioning point of the mpadded element and the content that follows it, and introduces padding after the rendering of the child content. Decreasing the width reduces the space between the positioning point of the mpadded element and the content that follows it, and removes space after the rendering of the child content. Setting the width to zero causes following content to be positioned at the positioning point of the mpadded element. Decreasing the width should generally be avoided, as it may cause overprinting of the following content.

The lspace attribute ("leading" space; see Section 3.1.5.1 Overall Directionality of Mathematics Formulas) specifies the horizontal location of the positioning point of the child content with respect to the positioning point of the mpadded element. By default they coincide, and therefore absolute values for lspace have the same effect as relative values. Positive values for the lspace attribute increase the space between the preceding content and the child content, and introduce padding before the rendering of the child content. Negative values for the lspace attributes reduce the space between the preceding content and the child content, and may cause overprinting of the preceding content, and should generally be avoided. Note that the lspace attribute does not affect the width of the mpadded element, and so the lspace attribute will also affect the space between the child content and following content, and may cause overprinting of the following content, unless the width is adjusted accordingly.

The voffset attribute specifies the vertical location of the positioning point of the child content with respect to the positioning point of the mpadded element. Positive values for the voffset attribute raise the rendering of the child content above the baseline. Negative values for the voffset attribute lower the rendering of the child content below the baseline. In either case, the voffset attribute may cause overprinting of neighboring content, which should generally be avoided. Note that t he voffset attribute does not affect the height or depth of the mpadded element, and so the voffset attribute will also affect the space between the child content and neighboring content, and may cause overprinting of the neighboring content, unless the height or depth is adjusted accordingly.

MathML renderers should ensure that, except for the effects of the attributes, the relative spacing between the contents of the mpadded element and surrounding MathML elements would not be modified by replacing an mpadded element with an mrow element with the same content, even if linebreaking occurs within the mpadded element. MathML does not define how non-default attribute values of an mpadded element interact with the linebreaking algorithm.

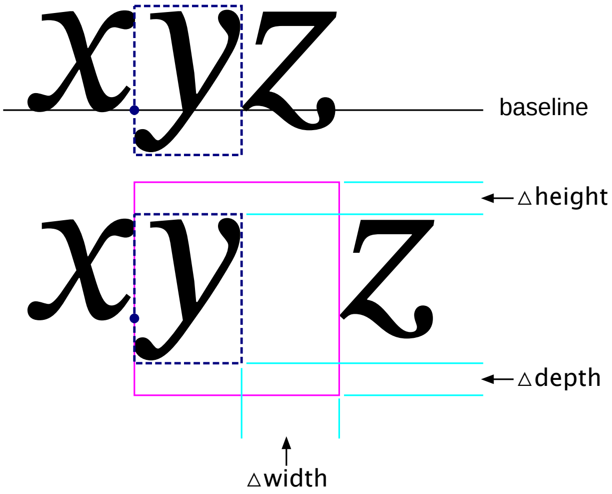
The effects of the size and position attributes are illustrated below. The following diagram illustrates the use of lspace and voffset to shift the position of child content without modifying the mpadded bounding box.



The corresponding MathML is:

```
<mrow>
  <mi>x</mi>
  <mpadded lspace="0.2em" voffset="0.3ex">
    <mi>y</mi>
  </mpadded>
  <mi>z</mi>
</mrow>
```

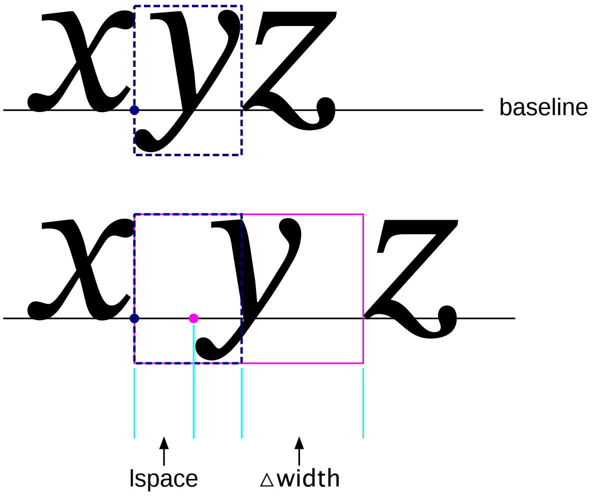
The next diagram illustrates the use of `width`, `height` and `depth` to modifying the `mpadded` bounding box without changing the relative position of the child content.



The corresponding MathML is:

```
<mrow>
  <mi>x</mi>
  <mpadded width="+90%width" height="+0.3ex" depth="+0.3ex">
    <mi>y</mi>
  </mpadded>
  <mi>z</mi>
</mrow>
```

The final diagram illustrates the generic use of `mpadded` to modify both the bounding box and relative position of child content.



The corresponding MathML is:

```
<mrow>
  <mi>x</mi>
  <mpadded lspace="0.3em" width="+0.6em">
    <mi>y</mi>
  </mpadded>
  <mi>z</mi>
</mrow>
```

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 Required Arguments.

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 Space `<mspace/>`) 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 3.1.10 Mathematics style attributes common to presentation elements (the `mathcolor` has no effect).

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 Operator, Fence, Separator or Accent `<mo>`.) 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

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

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 Operator, Fence, Separator or Accent `<mo>`. 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 Horizontally Group Sub-Expressions `<mrow>`). 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 3.1.10 Mathematics style attributes common to presentation elements. The delimiters and separators should be drawn using the color specified by `mathcolor`.

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 Collapsing Whitespace in Input.	
`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 Collapsing Whitespace in Input.	
`separators`	*string*	,
	Specifies a sequence of zero or more separator characters, optionally separated by whitespace. 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 an 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 Horizontally Group Sub-Expressions <mrow>.

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 Horizontally Group Sub-Expressions <mrow>), 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 Operator, Fence, Separator or Accent `<mo>`) 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> &#x2061;<!--FUNCTION APPLICATION--> </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 Required Arguments.

3.3.9.2 Attributes

`menclose` elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements. The notations should be drawn using the color specified by `mathcolor`.

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" | "phasorangle" | "radical" | "box" | "roundedbox" | "circle" | "left" | "right" | "top" | "bottom" | "updiagonalstrike" | "downdiagonalstrike" | "verticalstrike" | "horizontalstrike" | "northeastarrow" | "madruwb" | text) +	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. Of the predefined notations, only the following are affected by the directionality (see Section 3.1.5.1 Overall Directionality of Mathematics Formulas):

- "radical"
- "phasorangle"

When `notation` has the value "longdiv", the contents are drawn enclosed by a long division symbol. MathML 3 adds the `mlongdiv` element (Section 3.6.2 Long Division `<mlongdiv>`). This element supports notations for long division used in several countries and can be used to create a complete example of long division as shown in Section 3.6.8.3 Long Division. 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 "northeastarrow", "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 "northeastarrow" is a recommended value to implement because it can be used to implement TeX's \cancelto command. If a renderer implements other arrows for `menclose`, it is recommended that the arrow names are chosen from the following full set of names for consistancy and standardization among renderers:

- "uparrow"
- "rightarrow"
- "downarrow"
- "leftarrow"
- "northwestarrow"
- "southwestarrow"
- "southeastarrow"
- "northeastarrow"
- "updownarrow"
- "leftrightarrow"
- "northwestsoutheastarrow"
- "northeastsouthwestarrow"

The value "madruwb" should generate an enclosure representing an Arabic factorial (‘madruwb’ is the transliteration of the Arabic مضروب 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

$$\boxed{x+y}$$

An example of using `menclose` for actuarial notation is

```
<msub>
  <mi>a</mi>
  <mrow>
    <menclose notation='actuarial'>
      <mi>n</mi>
    </menclose>
    <mo>&#x2063;<!--INVISIBLE SEPARATOR--></mo>
    <mi>i</mi>
  </mrow>
</msub>
```

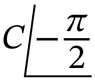
which renders roughly as

$$a_{\overline{n|}i}$$

An example of "phasorangle", which is used in circuit analysis, is:

```
<mi>C</mi>
<mrow>
  <menclose notation='phasorangle'>
    <mrow>
      <mo>&#x2212;<!--MINUS SIGN--></mo>
      <mfrac>
        <mi>&#x3C0;<!--GREEK SMALL LETTER PI--></mi>
        <mn>2</mn>
      </mfrac>
    </mrow>
  </menclose>
</mrow>
```

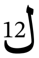
which renders roughly as



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 Displaystyle and Scriptlevel. 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 Displaystyle and Scriptlevel.)

3.4.1.2 Attributes

`msub` elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

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 Displaystyle and Scriptlevel.)

3.4.2.2 Attributes

msup elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

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 Displaystyle and Scriptlevel.)

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 3.1.10 Mathematics style attributes common to presentation elements.

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,

$$\int_0^1 e^x \, dx$$

would be represented as

```
<mrow>
  <msubsup>
    <mo> &#x222B;<!--INTEGRAL--> </mo>
    <mn> 0 </mn>
    <mn> 1 </mn>
  </msubsup>
  <mrow>
    <msup>
      <mi> &#x2147;<!--DOUBLE-STRUCK ITALIC SMALL E--> </mi>
      <mi> x </mi>
    </msup>
    <mo> &#x2062;<!--INVISIBLE TIMES--> </mo>
    <mrow>
      <mo> &#x2146;<!--DOUBLE-STRUCK ITALIC SMALL D--> </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 Displaystyle and Scriptlevel.)

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 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
`accentunder`	"true" | "false"	*automatic*
	Specifies 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. As specfied in Section 3.2.5.8.3 Horizontal Stretching Rules, the core of underscripts that are embellished operators should stretch to cover the base, but the alignment is based on the entire underscript.	

The default value of `accentunder` is false, unless *underscript* is an `mo` element or an embellished operator (see Section 3.2.5 Operator, Fence, Separator or Accent `<mo>`). 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> &#x23DF;<!--BOTTOM CURLY BRACKET--> </mo>
  </munder>
  <mtext>&#xA0;<!--NO-BREAK SPACE-->versus&#xA0;<!--NO-BREAK SPACE--></mtext>
  <munder accentunder="false">
    <mrow>
      <mi> x </mi>
      <mo> + </mo>
      <mi> y </mi>
      <mo> + </mo>
      <mi> z </mi>
    </mrow>
    <mo> &#x23DF;<!--BOTTOM CURLY BRACKET--> </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 Displaystyle and Scriptlevel.)

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 `&sum;`.

3.4.5.2 Attributes

`mover` elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
`accent`	"true" | "false"	*automatic*
	Specifies 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.	

<table>
<tr><th>Name</th><th>values</th><th>default</th></tr>
<tr><td>align</td><td>"left" | "right" | "center"</td><td>center</td></tr>
<tr><td></td><td colspan="2">Specifies whether the script is aligned left, center, or right under/over the base. As specfied in Section 3.2.5.8.3 Horizontal Stretching Rules, the core of overscripts that are embellished operators should stretch to cover the base, but the alignment is based on the entire overscript.</td></tr>
</table>

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 $\overset{\overbrace{}}{x+y+z}$. 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 Operator, Fence, Separator or Accent `<mo>`). 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> &#x5E;<!--CIRCUMFLEX ACCENT--> </mo>
  </mover>
  <mtext>&#xA0;<!--NO-BREAK SPACE-->versus&#xA0;<!--NO-BREAK SPACE--></mtext>
  <mover accent="false">
    <mi> x </mi>
    <mo> &#x5E;<!--CIRCUMFLEX ACCENT--> </mo>
  </mover>
</mrow>
```

```
<mrow>
  <mover accent="true">
    <mrow>
      <mi> x </mi>
      <mo> + </mo>
      <mi> y </mi>
      <mo> + </mo>
      <mi> z </mi>
    </mrow>
    <mo> &#x23DE;<!--TOP CURLY BRACKET--> </mo>
  </mover>
  <mtext>&#xA0;<!--NO-BREAK SPACE-->versus&#xA0;<!--NO-BREAK SPACE--></mtext>
  <mover accent="false">
    <mrow>
      <mi> x </mi>
      <mo> + </mo>
      <mi> y </mi>
      <mo> + </mo>
      <mi> z </mi>
    </mrow>
    <mo> &#x23DE;<!--TOP CURLY BRACKET--> </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 Displaystyle and Scriptlevel).

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 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
`accent`	"true" | "false"	*automatic*
	Specifies 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*
	Specifies 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. As specfied in Section 3.2.5.8.3 Horizontal Stretching Rules, the core of underscripts and overscripts that are embellished operators should stretch to cover the base, but the alignment is based on the entire underscript or overscript.	

The `munderover` element is used instead of separate `munder` and `mover` elements 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> &#x222B;<!--INTEGRAL--> </mo>
      <mn> 0 </mn>
    </munder>
    <mi> &#x221E;<!--INFINITY--> </mi>
  </mover>
  <mtext>&#xA0;<!--NO-BREAK SPACE-->versus&#xA0;<!--NO-BREAK SPACE--></mtext>
  <munderover>
    <mo> &#x222B;<!--INTEGRAL--> </mo>
    <mn> 0 </mn>
    <mi> &#x221E;<!--INFINITY--> </mi>
  </munderover>
</mrow>
```

3.4.7 Prescripts and Tensor Indices `<mmultiscripts>`, `<mprescripts/>`, `<none/>`

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 and prescripts. 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 displayed 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. For each sub and superscript pair, horizontal-alignment of the elements in the pair should be towards the base of the mmultiscripts. That is, pre-scripts should be right aligned, and post-scripts should be left aligned.

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 element `mprescripts` is only allowed as direct sub-expression of `mmultiscripts`.

3.4.7.2 Attributes

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

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 Displaystyle and Scriptlevel.)

3.4.7.3 Examples

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> &#x2061;<!--FUNCTION APPLICATION--> </mo>
  <mrow>
    <mo> ( </mo>
    <mrow>
      <mo> ; </mo>
      <mi> a </mi>
      <mo> ; </mo>
      <mi> z </mi>
    </mrow>
    <mo> ) </mo>
  </mrow>
</mrow>
```

$R_i{}^j{}_{kl}$ (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>
```

$${}_{123}^{1}X_{123}^{1}$$

```
<mmultiscripts>
  <mi>  X </mi>
  <mn> 123 </mn>
  <mn> 1 </mn>
  <mprescripts/>
  <mn> 123 </mn>
  <mn> 1 </mn>
</mmultiscripts>
```

An additional example of `mmultiscripts` shows how the binomial coefficient

$$\binom{5}{12}$$

can be displayed in Arabic style

$${}_{12}\text{ل}^{5}$$

```
<mstyle dir="rtl">
  <mmultiscripts><mo>&#x0644;<!--ARABIC LETTER LAM--></mo>
    <mn>12</mn><none/>
    <mprescripts/>
    <none/><mn>5</mn>
  </mmultiscripts>
</mstyle>
```

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 Elementary Math 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 3.1.10 Mathematics style attributes common to presentation elements. Any rules drawn as part of the `mtable` should be drawn using the color specified by `mathcolor`.

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), rather than the table as a whole, is aligned in the way described above with the exceptions noted below. If *rownumber* is negative, it counts rows from the bottom. When the value of *rownumber* is out of range or not an integer, it is ignored. If a row number is specified and the alignment value is "baseline" or "axis", the row's baseline or axis is used for alignment. Note this is only well defined when the `rowalign` value is "baseline" or "axis"; MathML does not specify how "baseline" or "axis" alignment should occur for other values of `rowalign`.	
`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 Alignment Markers `<maligngroup/>`, `<malignmark/>`.]	

Name	values	default
`alignmentscope`	("true" | "false") +	true
	[this attribute is described with the alignment elements, `maligngroup` and `malignmark`, in Section 3.5.5 Alignment Markers `<maligngroup/>`, `<malignmark/>`.]	
`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 or number without unit, the value is relative to the horizontal space that a MathML renderer has available, this is the current target width as used for linebreaking as specified in Section 3.1.7 Linebreaking of Expressions; this allows the author to specify, for example, a table being full width of the display. 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 columns. (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 (how the dashes are spaced is implementation dependent). (See the note below about multiple values).	
`columnlines`	("none" | "solid" | "dashed") +	none
	specifies whether and what kind of lines should be added between each column: "none" means no lines; "solid" means solid lines; "dashed" means dashed lines (how the dashes are spaced is implementation dependent). (See the note below about multiple values).	
`frame`	"none" | "solid" | "dashed"	none
	specifies whether and what kind of lines should be drawn around the table. "none" means no lines; "solid" means solid lines; "dashed" means dashed lines (how the dashes are spaced is implementation dependent).	
`framespacing`	*length, length*	0.4em 0.5ex
	specifies the additional spacing added between the table and frame, if `frame` is not "none". The first value specifies the spacing on the right and left; the second value specifies the spacing above and below.	

Name	values	default
`equalrows`	"true" | "false"	false
	specifies whether to force all rows to have the same total height.	
`equalcolumns`	"true" | "false"	false
	specifies whether to force all columns to have the same total width.	
`displaystyle`	"true" | "false"	false
	specifies the value of `displaystyle` within each cell, (`scriptlevel` is not changed); see Section 3.1.6 Displaystyle and Scriptlevel.	
`side`	"left" | "right" | "leftoverlap" | "rightoverlap"	right
	specifies on what side of the table labels from enclosed `mlabeledtr` (if any) should be placed. The variants "leftoverlap" and "rightoverlap" are useful when the table fits with the allowed width when the labels are omitted, but not when they are included: in such cases, the labels will overlap the row placed above it if the `rowalign` for that row is "top", otherwise it is placed below it.	
`minlabelspacing`	*length*	0.8em
	specifies the minimum space allowed between a label and the adjacent cell in the row.	

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 MathML Attribute Values). 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) in order, that is starting from the top row for rows or first column (left or right, depending on directionality) for columns. 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 areas 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.

The `displaystyle` attribute is allowed on the `mtable` element to set the inherited value of the attribute. If the attribute is not present, the `mtable` element sets `displaystyle` to "false" within the table elements. (See Section 3.1.6 Displaystyle and Scriptlevel.)

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 Table or Matrix `<mtable>`, `mtr` elements are effectively padded 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 3.1.10 Mathematics style attributes common to presentation elements.

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 Alignment Markers `<malligngroup/>`, `<malignmark/>`.]	

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`, and should be enclosed in an `mtd`. The rest of the children represent the contents of the row and are treated identically to the children of `mtr`; consequently all of the children 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, implementers 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` element 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. Alternatively, in a CSS context, one could use an empty `mtd` as the first child of `mlabeledtr` and use CSS counters and generated content to fill in the equation number using a CSS style such as

```
body {counter-reset: eqnum;}
mtd.eqnum {counter-increment: eqnum;}
mtd.eqnum:before {content: "(" counter(eqnum) ")"}
```

3.5.3.4 Example

```
<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>&#x2062;<!--INVISIBLE TIMES--></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 Required Arguments.

3.5.4.2 Attributes

`mtd` elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
`rowspan`	*positive-integer*	1
	causes the cell to be treated as if it occupied the number of rows specified. The corresponding `mtd` 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 `mtds` 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`.	

Name	values	default
groupalign	*group-alignment-list*	*inherited*
	[this attribute is described with the alignment elements, maligngroup and malignmark, in Section 3.5.5 Alignment Markers <maligngroup/>, <malignmark/>.]	

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 Space <mspace/>) 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 *i*th group consisting of the elements entirely after the *i*th `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 `mrow`s 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 Horizontally Group Sub-Expressions `<mrow>`. 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, but excluding `mrow` which contains a change of direction using the `dir` attribute;
- an `mstyle` element, but excluding those which change direction using the `dir` attribute;
- 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 an `maligngroup` element that is an argument of an `mfenced` element. Similarly, when inserting an `maligngroup` element in an element whose arguments have positional significance, it is necessary to introduce a new `mrow` element containing just the maligngroup element and the child element it precedes in order to preserve the proper expression structure. For example, to insert an maligngroup before the denominator child of an mfrac element, it is necessary to enclose the maligngroup and the denominator in an mrow to avoid introducing an illegal third child in the mfrac. 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 Space <mspace/> for an analogous example involving malignmark.

For the table cells that are divided into alignment groups, every element in their content must be part of exactly one alignment group, except for 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 (or left side, in a RTL context) 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 Space `<mspace/>`).

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 a LTR context; with X coming after Y in a RTL context). 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 Content Markup and Chapter 5 Mixing Markup Languages for Mathematical Expressions 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 3.1.10 Mathematics style attributes common to presentation elements (however, neither `mathcolor` nor `mathbackground` have any effect).

<table>
<tr><th>Name</th><th>values</th><th>default</th></tr>
<tr><td rowspan="2">edge</td><td>"left" | "right"</td><td>left</td></tr>
<tr><td colspan="2">see the discussion below.</td></tr>
</table>

The `edge` attribute 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 3.1.10 Mathematics style attributes common to presentation elements (however, neither `mathcolor` nor `mathbackground` have any effect).

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 character immediately before the left-most 'decimal point', i.e. matching the character specified by the `decimalpoint` attribute of `mstyle` (default ".", U+002E) 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 `mrow`s), `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.

Characters other than "." can be used as "decimal points" for alignment by using mstyle; more arbitrary alignment points can chosen by embedding 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 (from left to right, in an LTR context, or from right to left in an RTL context), 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 MathML Attribute Values, | 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:

$$\begin{array}{lllllll} 8.44 & x & + & 55 & y & = & 0 \\ 3.1 & x & - & 0.7 & y & = & -1.1 \end{array}$$

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> &#x2062;<!--INVISIBLE TIMES--> </mo>
            <maligngroup/>
            <mi> x </mi>
          </mrow>
          <maligngroup/>
          <mo> + </mo>
          <mrow>
            <maligngroup/>
            <mn> 55 </mn>
            <mo> &#x2062;<!--INVISIBLE TIMES--> </mo>
            <maligngroup/>
            <mi> y </mi>
          </mrow>
        </mrow>
        <maligngroup/>
        <mo> = </mo>
        <maligngroup/>
        <mn> 0 </mn>
      </mrow>
    </mtd>
  </mtr>
  <mtr>
    <mtd>
      <mrow>
        <mrow>
          <mrow>
            <maligngroup/>
            <mn> 3.1 </mn>
            <mo> &#x2062;<!--INVISIBLE TIMES--> </mo>
            <maligngroup/>
            <mi> x </mi>
          </mrow>
          <maligngroup/>
          <mo> - </mo>
          <mrow>
            <maligngroup/>
            <mn> 0.7 </mn>
            <mo> &#x2062;<!--INVISIBLE TIMES--> </mo>
            <maligngroup/>
            <mi> y </mi>
          </mrow>
        </mrow>
        <maligngroup/>
        <mo> = </mo>
        <maligngroup/>
        <mrow>
          <mo> - </mo>
          <mn> 1.1 </mn>
        </mrow>
      </mrow>
    </mtd>
  </mtr>
</mtable>
```

3.5.5.9 Further details of alignment elements

The alignment elements maligngroup 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 renderer 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 Table or Matrix `<mtable>`).

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 is scanned. The *i*th scan covers the *i*th 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 2*n* horizontally adjacent sections, called L(*i*) and R(*i*) for *i* from 1 to *n*, using the 2*n* maximum side-widths computed above; for each *i*, the width of all sections called L(*i*) is the maximum width of any cell's *i*th 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 *i*th alignment group to the right of its alignment point.

Each alignment group is then shifted horizontally as a block to a unique position that places: in the section called L(*i*) that part of the *i*th group to the left of its alignment point; in the section called R(*i*) that part of the *i*th group to the right of its alignment point. This results in the alignment point of each *i*th group being on the boundary between adjacent sections L(*i*) and R(*i*), so that all alignment points of *i*th 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`
: align rows of digits and operators

`msgroup`
: groups rows with similar alignment

`msrow`
: groups digits and operators into a row

`msline`
: draws lines between rows of the stack

`mscarries`
: annotates the following row with optional borrows/carries and/or crossouts

`mscarry`
: a borrow/carry and/or crossout for a single digit

`mlongdiv`
: specifies a divisor and a quotient for long division, along with a stack of the intermediate computations

`mstack` and `mlongdiv` are the parent elements for all elementary math layout. Any children of `mstack`, `mlongdiv`, and `msgroup`, besides `msrow`, `msgroup`, `mscarries` and `msline`, are treated as if implicitly surrounded by an `msrow` (See Section 3.6.4 Rows in Elementary Math `<msrow>` for more details about rows).

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 Directionality).

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:

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

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 Elementary Math Examples.

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 Rows in Elementary Math `<msrow>` 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 row.
4. The maximum 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.

3.6.1.2 Attributes

`mstack` elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

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" | "decimalpoint"	decimalpoint
	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 alignment. Rows with non-zero `position`, or affected by a `shift`, are treated as if the requisite number of empty columns were added on the appropriate side; see Section 3.6.3 Group Rows with Similiar Positions `<msgroup>` and Section 3.6.4 Rows in Elementary	

Name	values	default
	Math <msrow>. For "decimalpoint", the column used is the left-most column in each row that contains the decimalpoint character specified using the decimalpoint attribute of mstyle (default "."). If there is no decimalpoint character in the row, an implied decimal is assumed on the right of the first number in the row; See "decimalpoint" for a discussion of "decimalpoint".	
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* | "loose" | "medium" | "tight"	medium
	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. The keywords "loose", "medium", and "tight" automatically adjust spacing to when carries or other entries in a column are wide. The three values allow authors to some flexibility in choosing what the layout looks like without having to figure out what values works well. In all cases, the spacing between columns is a fixed amount and does not vary between different columns.	

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 divisor and the result of the division, in that order. 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.

The result or divisor may be an elementary math element or may be none. In particular, if msgroup is used, the elements in that group may or may not form their own mstack or be part of the dividend's mstack, depending upon the value of the longdivstyle attribute. For example, in the US style for division, the result is treated as part of the dividend's mstack, but divisor is not. MathML does not specify when the result and divisor form their own mstack, nor does it specify what should happen if msline or other elementary math elements are used for the result or divisor and they do not participate in the dividend's mstack layout.

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 3.1.10 Mathematics style attributes common to presentation elements), 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. Any rules drawn as part of division layout should be drawn using the color specified by mathcolor.

Name	values	default
`longdivstyle`	"lefttop" | "stackedrightright" | "mediumstackedrightright" | "shortstackedrightright" | "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 Long Division 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 Britain, and elsewhere

"stackedrightright"
: a notation that is commonly used in France and elsewhere

"mediumrightright"
: a notation that is commonly used in Russia and elsewhere

"shortstackedrightright"
: a notation that is commonly used in Brazil and elsewhere

"righttop"
: a notation that is commonly used in China, Sweden, and elsewhere

"left/\right"
: a notation that is commonly used in Netherlands

"left)(right"
: a notation that is commonly used in India

":right=right "
: a notation that is commonly used in Germany

"stackedleftleft "
: a notation that is commonly used in Arabic countries

"stackedleftlinetop"
: a notation that is commonly 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` and `mlongdiv` elements that have a similar position relative to the alignment of stack. If not explicitly given, the children representing the stack in `mstack` and `mlongdiv` are treated as if they are implicitly surrounded by an `msgroup` element.

3.6.3.2 Attributes

`msgroup` elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
`position`	*integer*	0
	specifies the horizontal position of the rows within this group relative the position determined by the containing `msgroup` (according to its `position` and `shift` attributes). The resulting position value is relative to the column specified by `stackalign` of the containing `mstack` or `mlongdiv`. 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	

Name	values	default
	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`	*integer*	0
	specifies an incremental shift of position for successive children (rows or groups) within this group. The value is interpreted as with position, but specifies the position of each child (except the first) with respect to the previous child in the group.	

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 "-" alongside 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 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 Directionality.

3.6.4.2 Attributes

`msrow` elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
`position`	*integer*	0
	specifies the horizontal position of the rows within this group relative the position determined by the containing `msgroup` (according to its `position` and `shift` attributes). The resulting position value is relative to the column specified by `stackalign` of the containing `mstack` or `mlongdiv`. 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.	

3.6.5 Carries, Borrows, and Crossouts `<mscarries>`

3.6.5.1 Description

The `mscarries` element is used for various annotations such as carries, borrows, and crossouts that occur in elementary math. The children are associated with elements in the *following* row of the `mstack`. It is an error for `mscarries` to be the last element of an `mstack` or `mlongdiv` element. Each child of the `mscarries` applies to the same column in the following row. As 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 Directionality.

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. The `mscarries` element sets `displaystyle` to "false", and increments `scriptlevel` by 1, so the children are typically displayed in a smaller font. (See Section 3.1.6 Displaystyle and Scriptlevel.) It also changes the default value of `scriptsizemultiplier`. The effect is that the inherited value of `scriptsizemultiplier` should still override the default value, but the default value, inside `mscarries`, should be "0.6". `scriptsizemultiplier` can be set on the `mscarries` element, and the value should override the inherited value as usual.

If two rows of carries are adjacent to each other, the first row of carries annotates the second (following) row as if the second row had `location`="n". This means that the second row, even if it does not draw, visually uses some (undefined by this specification) amount of space when displayed.

3.6.5.2 Attributes

`mscarries` elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
`position`	*integer*	0
	specifies the horizontal position of the rows within this group relative the position determined by the containing `msgroup` (according to its `position` and `shift` attributes). The resulting position value is relative to the column specified by `stackalign` of the containing `mstack` or `mlongdiv`. 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 Elementary Math Examples for examples of the different values. The crossout is only applied for columns which have a corresponding `mscarry`. The crossouts should be drawn using the color specified by `mathcolor`.	
`scriptsizemultiplier`	*number*	*inherited (0.6)*
	specifies the factor to change the font size by. See Section 3.1.6 Displaystyle and Scriptlevel 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

The mscarry element accepts the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

Name	values	default
location	"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"	*inherited*
	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")*	*inherited*
	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. The crossout should be drawn using the color specified by mathcolor.	

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. If the length is specified, the line is positioned and drawn as if it were a number with the given number of digits.

3.6.7.2 Attributes

msline elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements. The line should be drawn using the color specified by mathcolor.

Name	values	default
position	*integer*	0
	specifies the horizontal position of the rows within this group relative the position determined by the containing msgroup (according to its position and shift attributes). The resulting position value is relative to the column specified by stackalign of the containing mstack or mlongdiv. 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. Note that since the default line length spans the entire mstack, the position has no effect unless the length is specified as non-zero.	
length	*unsigned-integer*	0
	Specifies the number of columns that should be spanned by the line. A value of '0' (the default) means that *all* columns in the row are spanned (in which case position and stackalign have no effect).	
leftoverhang	*length*	0
	Specifies an extra amount that the line should overhang on the left of the leftmost column spanned by the line.	

Name	values	default
`rightoverhang`	*length*	0
	Specifies an extra amount that the line should overhang on the right of the rightmost column spanned by the line.	
`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 Fractions `<mfrac>` for discussion of the thickness keywords "medium", "thin" and "thick".)	

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.

$$\begin{array}{cccc} & 4 & 2 & 4 \\ + & & 3 & 3 \\ \hline \end{array}$$

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 is on the implied decimal point to the right of the numbers, it is not necessary to pad any row.

$$\begin{array}{llll} 1 & 2 & 3 & \\ 4 & 5 & 6 & + \\ \hline 5 & 7 & 9 & \end{array}$$

```
<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.

$$\begin{array}{r} 2,\overset{2}{\cancel{3}}\overset{12}{\cancel{2}}7 \\ -1,156 \\ \hline 1,171 \end{array} \qquad \begin{array}{r} 2,\overset{2}{\cancel{3}}{}^{1}27 \\ -1,156 \\ \hline 1,171 \end{array}$$

The MathML for the first example is:

```
<mstack>
  <mscarries crossout='updiagonalstrike'>
    <mn>2</mn>  <mn>12</mn>  <mscarry crossout='none'> <none/> </mscarry>
  </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' location='n'> <mn>2</mn> </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):

$$\begin{array}{r} \cancel{5}\overset{\underline{10}}{2} \\ -7 \\ \hline 45 \end{array}$$

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.

$$\begin{array}{ccccc}
 & & 1 & 2 & 3 \\
 & \times & 3 & 2 & 1 \\
\hline
 & & 1 & 2 & 3 \\
 & 2 & 4 & 6 & \\
3 & 6 & 9 & & \\
\hline
\end{array}$$

```
<mstack>
  <msgroup>
    <mn>123</mn>
    <msrow><mo>&#xD7;<!--MULTIPLICATION SIGN--></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.

$$\begin{array}{ccccccccc}
 & & & & & & \scriptstyle 1 & \scriptstyle 1 & \\
 & & & & & & \scriptstyle 1 & \scriptstyle 1 & \\
 & & & & 1 & , & 2 & 3 & 4 \\
 & & & \times & 4 & , & 3 & 2 & 1 \\
\hline
\scriptstyle 1 & & \scriptstyle 1 & \scriptstyle 1 & \scriptstyle 1 & & \scriptstyle 1 & & \\
 & & & & 1 & , & 2 & 3 & 4 \\
 & & & 2 & 4 & , & 6 & 8 & \\
 & & 3 & 7 & 0 & , & 2 & & \\
4 & , & 9 & 3 & 6 & & & & \\
\hline
5 & , & 3 & 3 & 2 & , & 1 & 1 & 4
\end{array}$$

```
<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;<!--MULTIPLICATION SIGN--></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">
```

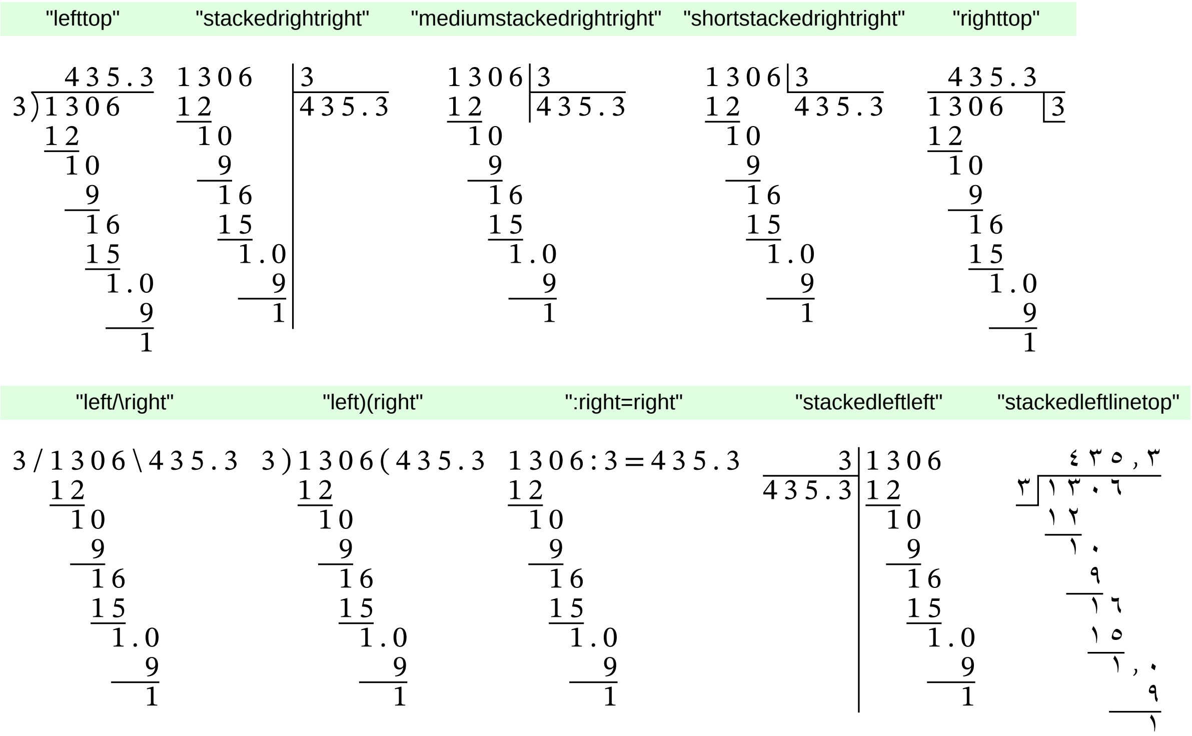
```
    <mn>1,234</mn>
    <mn>24,68</mn>
    <mn>370,2</mn>
    <msrow position="1"> <mn>4,936</mn> </msrow>
  </msgroup>
  <msline/>

  <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 apparent 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 Attributes



The MathML for the first example is shown below. It illustrates the use of nested `msgroups` and how the `position` is calculated in those usages.

```
<mlongdiv longdivstyle="lefttop">
  <mn> 3 </mn>
  <mn> 435.3</mn>

  <mn> 1306</mn>

  <msgroup position="2" shift="-1">
    <msgroup>
```

```
      <mn> 12</mn>
      <msline length="2"/>
    </msgroup>
    <msgroup>
      <mn> 10</mn>
      <mn> 9</mn>
      <msline length="2"/>
    </msgroup>
    <msgroup>
      <mn> 16</mn>
      <mn> 15</mn>
      <msline length="2"/>
      <mn> 1.0</mn>          <!-- aligns on '.', not the right edge ('0')
                                                                          -->
    </msgroup>
    <msgroup position='-1'>   <!-- extra shift to move to the right of the
                                                                      "." -->
       <mn> 9</mn>
      <msline length="3"/>
      <mn> 1</mn>
    </msgroup>
  </msgroup>
</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, as shown below.

```
<mstyle decimalpoint="&#x066B;"><!--ARABIC DECIMAL SEPARATOR-->
<mlongdiv longdivstyle="stackedleftlinetop">
  <mn> &#x0663;<!--ARABIC-INDIC DIGIT THREE--> </mn>
  <mn> &#x0664;<!--ARABIC-INDIC DIGIT FOUR-->&#x0663;<!--ARABIC-INDIC
                    DIGIT THREE-->&#x0665;<!--ARABIC-INDIC DIGIT FIVE-->&#x066B;<!--ARABIC DECIMAL
                              SEPARATOR-->&#x0663;<!--ARABIC-INDIC DIGIT THREE--></mn>

  <mn> &#x0661;<!--ARABIC-INDIC DIGIT ONE-->&#x0663;<!--ARABIC-INDIC DIGIT
              THREE-->&#x0660;<!--ARABIC-INDIC DIGIT ZERO-->&#x0666;<!--ARABIC-INDIC DIGIT SIX--></mn>
  <msgroup position="2" shift="-1">
    <msgroup>
      <mn> &#x0661;<!--ARABIC-INDIC DIGIT ONE-->&#x0662;<!--ARABIC-INDIC DIGIT TWO--></mn>
      <msline length="2"/>
    </msgroup>
    <msgroup>
      <mn> &#x0661;<!--ARABIC-INDIC DIGIT ONE-->&#x0660;<!--ARABIC-INDIC DIGIT ZERO--></mn>
      <mn> &#x0669;<!--ARABIC-INDIC DIGIT NINE--></mn>
      <msline length="2"/>
    </msgroup>
    <msgroup>
      <mn> &#x0661;<!--ARABIC-INDIC DIGIT ONE-->&#x0666;<!--ARABIC-INDIC DIGIT SIX--></mn>
      <mn> &#x0661;<!--ARABIC-INDIC DIGIT ONE-->&#x0665;<!--ARABIC-INDIC DIGIT FIVE--></mn>
      <msline length="2"/>
        <mn> &#x0661;<!--ARABIC-INDIC DIGIT ONE-->&#x066B;<!--ARABIC DECIMAL
                              SEPARATOR-->&#x0660;<!--ARABIC-INDIC DIGIT ZERO--></mn>
   </msgroup>
   <msgroup position='-1'>
```

```
       <mn> &#x0669;<!--ARABIC-INDIC DIGIT NINE--></mn>
      <msline length="3"/>
      <mn> &#x0661;<!--ARABIC-INDIC DIGIT ONE--></mn>
    </msgroup>
  </msgroup>
</mlongdiv>
</mstyle>
```

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:

$$0.333\overline{3}$$

$$0.\overline{142857}$$

$$0.\underline{142857}$$

$$0.\dot{1}4285\dot{7}$$

The MathML for these involves using `mstack`, `msrow`, and `msline` in a straightforward manner. The MathML for the preceding examples above is given below.

```
<mstack stackalign="right">
  <msline length="1"/>
  <mn> 0.3333 </mn>
</mstack>
```

```
<mstack stackalign="right">
  <msline length="6"/>
  <mn> 0.142857 </mn>
</mstack>
```

```
<mstack stackalign="right">
  <mn> 0.142857 </mn>
  <msline length="6"/>
</mstack>
```

```
<mstack stackalign="right">
  <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>`

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 6.4.4 Linking.

3.7.1.1 Attributes

`maction` elements accept the attributes listed below in addition to those specified in Section 3.1.10 Mathematics style attributes common to presentation elements.

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 Handling of Errors.

Name	values	default
`actiontype`	*string*	*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 values listed below.	
`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 action types 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 Transferring MathML), 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.

When a MathML application receives a mouse event that may be processed by two or more nested maction elements, the innermost maction element of each action type should respond to the event.

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 in XML; in HTML the attributes must begin with "data-". An XML example 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 Schema (see Section 2.3.3 Attributes for unspecified data). 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`="application/mathml-presentation+xml" or `encoding`= "MathML-Presentation" may be used and presentation MathML processors should use this value for the presentation.

See Section 5.1 Annotation Framework for more details about the `semantics` and `annotation-xml` elements.

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 varies across cultures and geographical regions. For example, many notations for long division are in use in different parts of the world today. 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 Content MathML Elements Encoding Expression Structure, 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. The next two sections introduce markup that is more convenient than Strict markup for some purposes, somewhat less formal and verbose. In Section 4.3 Content MathML for Specific Structures, 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 Content MathML for Specific Operators and Constants, elements are introduced for many common function, operators and constants. This section contains many examples, including equivalent Strict Content expressions. In Section 4.5 Deprecated Content Elements, elements from MathML 1 and 2 whose use is now discouraged are listed. Finally, Section 4.6 The Strict Content MathML Transformation summarizes the algorithm 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 "$\cos(\pi)$" 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 referring to mathematical concepts outside of the base collection, allowing them to be represented, as well.

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 native implementations of the MathML elements to declarative notation definitions, to XSLT style sheets. Because rendering requirements for Content MathML vary widely, MathML 3 does not provide a normative specification for rendering. Instead, typical renderings are suggested 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, but sufficient, 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.

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`

Strict Content MathML	OpenMath
ci	OMV
cs	OMSTR
apply	OMA
bind	OMBIND
bvar	OMBVAR
share	OMR
semantics	OMATTR
annotation, annotation-xml	OMATP, OMFOREIGN
cerror	OME
cbytes	OMB

In MathML 3, formal semantics Content MathML expressions are given by specifying equivalent Strict Content MathML expressions. Since Strict Content MathML expressions all have carefully-defined semantics given in terms of OpenMath Content Dictionaries, all Content MathML expressions inherit well-defined semantics in this way. To make the correspondence exact, an algorithm is given in terms of transformation rules that are applied to rewrite non-Strict MathML constructs into a strict equivalents. The individual rules are introduced in context throughout the chapter. In Section 4.6 The Strict Content MathML Transformation, 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 Operator Classes where these classes are defined. Some special case rules for specific elements are given in Section Section 4.4 Content MathML for Specific Operators and Constants. Transformations in Section 4.2 Content MathML Elements Encoding Expression Structure concern non-Strict usages of the core Content MathML elements, those in Section 4.3 Content MathML for Specific Structures concern the rewriting of some additional structures not directly supported in Strict Content MathML.

The full algorithm described inSection 4.6 The Strict Content MathML Transformation is complete in the sense that 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 expressions 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 Symbols `<csymbol>`.

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.

4.1.5 Content MathML Concepts

The basic building blocks of Content MathML expressions are numbers, identifiers and symbols. These building blocks are combined using function applications and binding operators. It is important to have a basic understanding of these key mathematical concepts, and how they are reflected in the design of Content MathML. For the convenience of the reader, these concepts are reviewed here.

In the expression "$x + y$", x is a mathematical variable, meaning an identifier that represents a quantity with no fixed value. It may have other properties, such as being an integer, but its value is not a fixed property. By contrast, the plus sign is an identifier that represents a fixed and externally defined object, namely the addition function. Such an identifier is termed a *symbol*, to distinguish it from a variable. Common elementary functions and operators all have fixed, external definitions, and are hence symbols. Content MathML uses the `ci` element to represent variables, and the `csymbol` to represent symbols.

The most fundamental way in which symbols and variables are combined is function application. Content MathML makes a crucial semantic distinction between a function itself (a symbol such as the sine function, or a variable such as f) and the result of applying the function to arguments. The `apply` element groups the function with its arguments syntactically, and represents the expression resulting from applying that function to its arguments.

Mathematically, variables are divided into *bound* and *free* variables. Bound variables are variables that are assigned a special role by a binding operator within a certain scope. For example, the index variable within a summation is a bound variable. They can be characterized as variables with the property that they can be renamed consistently throughout the binding scope without changing the underlying meaning of the expression. Variables that are not bound are termed free variables. Because the logical distinction between bound and free variables is important for well-defined semantics, Content MathML differentiates between the application of a function to a free variable, e.g. $f(x)$ and the operation of binding a variable within a scope. The `bind` element is used the delineate the binding scope, and group the binding operator with its bound variables, which are indicated by the `bvar` element.

In Strict Content markup, the `bind` element is the only way of performing variable binding. In non-Strict usage, however, markup is provided that more closely resembles well-known idiomatic notations, such as the "limit" notations for sums and integrals. These constructs often implicitly bind variables, such as the variable of integration, or the index variable in a sum. MathML terms the elements used to represent the auxiliary data such as limits required by these constructions *qualifier* elements.

Expressions involving qualifiers follow one of a small number of idiomatic patterns, each of which applies to class of similar binding operators. For example, sums and products are in the same class because they use index variables following the same pattern. The Content MathML operator classes are described in detail in Section 4.3.4 Operator Classes.

Each Content MathML element is described in a section below that begins with a table summarizing the key information about the element. For elements that have different Strict and non-Strict usage, these syntax tables are divided to clearly separate the two cases. The element's content model is given in the **Content** row, linked to the MathML Schema in Appendix A Parsing MathML. The **Attributes**, and **Attribute Values** rows similarly link to the schema. 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 Operator Classes. Finally, the **Qualifiers** row clarifies whether the operator takes qualifiers and if so, which. Note **Class** and **Qualifiers** specify how many siblings may follow the operator element in an `apply`, or the children of the element for container elements; see Section 4.2.5 Function Application `<apply>` and Section 4.3.3 Qualifiers for details).

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 Content MathML for Specific Structures and Section 4.4 Content MathML for Specific Operators and Constants, and allows a richer content model presented in this section. Differences in Strict and non-Strict usage of are highlighted in the sections discussing each of the Strict element below.

4.2.1 Numbers `<cn>`

	Schema Fragment (Strict)		Schema Fragment (Full)	
Class	Cn		Cn	
Attributes	CommonAtt, type		CommonAtt, DefEncAtt, type?, base?	
`type` Attribute Values	"integer" | "real" | "double" | "hexdouble"		"integer" | "real" | "double" | "hexdouble" | "e-notation" | "rational" | "complex-cartesian" | "complex-polar" | "constant" | text	default is real
`base` Attribute Values			integer	default is 10
Content	text		(text | mglyph | sep | PresentationExpression)*	

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>`, `<sep/>` - 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 uses of `<cn>`

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 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 NaN in the example, cannot be represented in the lexical format for the "double" type.

Content MathML

```
<cn type="hexdouble">7F800000</cn>
```

Sample Presentation

```
<mn>0x7F800000</mn>
```

0x7F800000

4.2.1.3 Non-Strict 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 base 10 number 32736.

Content MathML

```
<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, support for additional characters (if any) that may be used for digits when `base` > 36 is application specific.

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/>`.

Content MathML

```
<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.

Content MathML

```
<cn type="complex-cartesian"> 12.3 <sep/> 5 </cn>
```

Sample Presentation

```
<mrow>
 <mn>12.3</mn><mo>+</mo><mn>5</mn><mo>&#x2062;<!--INVISIBLE TIMES--></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.

Content MathML

```
<cn type="complex-polar"> 2 <sep/> 3.1415 </cn>
```

Sample Presentation

```
<mrow>
 <mn>2</mn>
 <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
 <msup>
  <mi>e</mi>
  <mrow><mi>i</mi><mo>&#x2062;<!--INVISIBLE TIMES--></mo><mn>3.1415</mn>
                                                                 </mrow>
 </msup>
</mrow>
```

$$2e^{i3.1415}$$

```
<mrow>
 <mi>Polar</mi>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mfenced><mn>2</mn><mn>3.1415</mn></mfenced>
</mrow>
```

$$\mathrm{Polar}(2, 3.1415)$$

constant

If the value `type` is "constant", then the content should be a Unicode representation 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 using these representations or a Strict csymbol representation is preferred.

In addition to the additional values of the type attribute, the content of `cn` element can contain (in addition to the `sep` element allowed in Strict Content MathML) `mglyph` elements to refer to characters not currently available in Unicode, or a general presentation construct (see Section 3.1.9 Summary of Presentation Elements), which is used for rendering (see Section 4.1.2 The Structure and Scope of Content MathML Expressions).

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="nums1">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 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

Rewrite: cn presentation mathml

If the `cn` contains Presentation MathML markup, then it may be rewritten to Strict MathML using variants of the rules above where the arguments of the constructor are `ci` elements annotated with the supplied Presentation MathML.

A `cn` expression with non-text content of the form

```
<cn type="rational"><mi>P</mi><sep/><mi>Q</mi></cn>
```

is transformed to Strict Content MathML by rewriting it to

```
<apply><csymbol cd="nums1">rational</csymbol>
 <semantics>
  <ci>p</ci>
  <annotation-xml encoding="MathML-Presentation">
```

```
    <mi>P</mi>
   </annotation-xml>
  </semantics>
  <semantics>
   <ci>q</ci>
   <annotation-xml encoding="MathML-Presentation">
    <mi>Q</mi>
   </annotation-xml>
  </semantics>
</apply>
```

Where the identifier names, p and q, (which have 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).

A related special case arises when a `cn` element contains character data not permitted in Strict Content MathML usage, e.g. non-digit, alphabetic characters. Conceptually, this is analogous to a `cn` element containing a presentation markup `mtext` element, and could be rewritten accordingly. However, since the resulting annotation would contain no additional rendering information, such instances should be rewritten directly as `ci` elements, rather than as a `semantics` construct.

4.2.2 Content Identifiers `<ci>`

	Schema Fragment (Strict)	Schema Fragment (Full)
Class	Ci	Ci
Attributes	CommonAtt, type?	CommonAtt, DefEncAtt, 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	text | mglyph | PresentationExpression

Content MathML uses the `ci` element (mnemonic for "content identifier") to construct a variable. Content identifiers represent "mathematical variables" which have properties, but no fixed value. For example, x and y are variables in the expression "$x + y$", and the variable x would be represented as

```
<ci>x</ci>
```

In MathML, variables are distinguished from symbols, which have fixed, external definitions, and are represented by the csymbol element.

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 Bindings and Bound Variables `<bind>` and `<bvar>` for a discussion).

4.2.2.1 Strict uses of `<ci>`

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>
```

Note that "complex" should be considered an alias for "complex-cartesian" and rewritten to the same complex_cartesian_type symbol. It is perhaps a more natural type name for use with `ci` as the distinction between cartesian and polar form really only affects the interpretation of literals encoded with `cn`.

4.2.2.2 Non-Strict 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.

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 [MathMLTypes] for more examples.

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>
```

The `ci` element can contain `mglyph` elements to refer to characters not currently available in Unicode, or a general presentation construct (see Section 3.1.9 Summary of Presentation Elements), which is used for rendering (see Section 4.1.2 The Structure and Scope of Content MathML Expressions).

Rewrite: ci presentation mathml

A `ci` expression with non-text content of the form

```
<ci><mi>P</mi></ci>
```

is transformed to Strict Content MathML by rewriting it to

```
<semantics>
  <ci>p</ci>
  <annotation-xml encoding="MathML-Presentation">
    <mi>P</mi>
  </annotation-xml>
</semantics>
```

Where the identifier name, p, (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, DefEncAtt, type?, cd?
Content	SymbolName	text | mglyph | PresentationExpression
Qualifiers		BvarQ, DomainQ, degree, momentabout, logbase

A `csymbol` is used to refer to a specific, mathematically-defined concept with an external definition. In the expression "$x + y$", the plus sign is a symbol since it has a specific, external definition, namely the addition function. MathML 3 calls such an identifier a *symbol*. Elementary functions and common mathematical operators are all examples of symbols. Note that the term "symbol" is used here in an abstract sense and has no connection with any particular presentation of the construct on screen or paper.

4.2.3.1 Strict uses of `<csymbol>`

The `csymbol` identifies the specific mathematical concept it represents by referencing its definition 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, several longstanding efforts have been organized to develop systematic, public repositories of mathematical definitions. Most notable of these, the OpenMath Society repository of Content Dictionaries (CDs) is extensive, open and active. 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. This is the form required in Strict Content MathML. 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. It is described in the following section

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 Attributes. 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 Non-Strict 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 Summary of Presentation Elements), which is used for rendering (see Section 4.1.2 The Structure and Scope of Content MathML Expressions). In this case, when writing to Strict Content MathML, the csymbol should be treated as a `ci` element, and rewritten using Rewrite: ci presentation mathml.

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
```

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.

Mapping to Strict Content MathML

In non-Strict usage `csymbol` allows the use of a `type` attribute.

Rewrite: csymbol type annotation

In Strict Content, type attributes are represented via semantic attribution. An expression of the form

```
<csymbol type="T">symbolname</csymbol>
```

is rewritten to

```
<semantics>
  <csymbol>symbolname</csymbol>
  <annotation-xml cd="mathmltypes" name="type" encoding="MathML-Content">
    <ci>T</ci>
  </annotation-xml>
</semantics>
```

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 (Strict)	Schema Fragment (Full)
Class	Cs	Cs
Attributes	CommonAtt	CommonAtt, DefEncAtt
Content	text	text

The `cs` element encodes "string literals" which may be used in Content MathML expressions.

The content of cs is text; no Presentation MathML constructs are allowed even when used in non-strict markup. Specifically, `cs` may not contain `mglyph` elements, and the content does not undergo white space normalization.

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>&#xa0;<!--NO-BREAK SPACE-->&#xa0;<!--NO-BREAK SPACE--></ms>
 <mo>}</mo>
</mrow>
```

{"A","B"," "}

4.2.5 Function Application `<apply>`

	Schema Fragment (Strict)	Schema Fragment (Full)
Class	Apply	Apply
Attributes	CommonAtt	CommonAtt, DefEncAtt
Content	ContExp+	ContExp+ | (ContExp, BvarQ, 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 application 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, $ax + b$ 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 is used.

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 *<mi>F</mi>* denotes the rendering of *<ci>f</ci>* and *<mi>Ai</mi>* the rendering of *<ci>ai</ci>*, the sample rendering of a simple application is as follows:

Content MathML

```
<apply><ci>f</ci>
  <ci>a1</ci>
  <ci>a2</ci>
  <ci>...</ci>
  <ci>an</ci>
</apply>
```

Sample Presentation

```
<mrow>
  <mi>F</mi>
  <mo>&#x2061;</mo>
```

```
 <mrow>
  <mo fence="true">(</mo>
  <mi>A1</mi>
  <mo separator="true">,</mo>
  <mi>...</mi>
  <mo separator="true">,</mo>
  <mi>A2</mi>
  <mo separator="true">,</mo>
  <mi>An</mi>
  <mo fence="true">)</mo>
 </mrow>
</mrow>
```

Non-Strict MathML applications may also be used with qualifiers. In the absence of any more specific rendering rules for well-known operators, rendering should follow the sample presentation below, motivated by the typical presentation for `sum`. Let `<mi>Op</mi>` denote the rendering of `<ci>op</ci>`, `<mi>X</mi>` the rendering of `<ci>x</ci>`, and so on. Then:

Content MathML

```
<apply><ci>op</ci>
  <bvar><ci>x</ci></bvar>
  <domainofapplication><ci>d</ci></domainofapplication>
  <ci>expression-in-x</ci>
</apply>
```

Sample Presentation

```
<mrow>
 <munder>
  <mi>Op</mi>
  <mrow><mi>X</mi><mo>&#x2208;</mo><!--ELEMENT OF--><mi>D</mi></mrow>
 </munder>
 <mo>&#x2061;</mo><!--FUNCTION APPLICATION-->
 <mrow>
  <mo fence="true">(</mo>
  <mi>Expression-in-X</mi>
  <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, DefEncAtt
Content	ContExp, BvarQ*, ContExp	ContExp, BvarQ*, 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, for example integral operator. 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, DefEncAtt
Content	ci | semantics-ci	(ci | semantics-ci), degree? | degree?, (ci | semantics-ci

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 Qualifiers and Section 4.4 Content MathML for Specific Operators and Constants.

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 *<ci>b</ci>* and *<ci>s</ci>* are Content MathML expressions that render as the Presentation MathML expressions *<mi>B</mi>* and *<mi>S</mi>* then the sample rendering of a binding element is as follows:

Content MathML

```
<bind><ci>b</ci>
  <bvar><ci>x1</ci></bvar>
  <bvar><ci>...</ci></bvar>
  <bvar><ci>xn</ci></bvar>
  <ci>s</ci>
</bind>
```

Sample Presentation

```
<mrow>
 <mi>B</mi>
 <mrow>
  <mi>x1</mi>
  <mo separator="true">,</mo>
  <mi>...</mi>
  <mo separator="true">,</mo>
  <mi>xn</mi>
 </mrow>
 <mo separator="true">.</mo>
 <mi>S</mi>
</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, src
`src` 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 equivalent to 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.

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="badid1"` dominates its third child, which dominates the `share` element, which dominates its target: the element with `id="badid1"`. 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 $\frac{1}{1+\frac{1}{1+\frac{1}{1+\cdots}}}$. 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">
  <csymbol cd="arith1">plus</csymbol>
  <cn>1</cn>
  <share href="#baz"/>
</apply>
```

```
<apply id="baz">
  <csymbol cd="arith1">plus</csymbol>
  <cn>1</cn>
  <share href="#bar"/>
</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

There are several acceptable renderings for the `share` element. These include rendering the element as a hypertext link to the referenced element and using the rendering of the element referenced by the `href` attribute.

4.2.8 Attribution via `semantics`

Content elements can be annotated with additional information via the `semantics` element. MathML uses the `semantics` element to wrap the annotated element and the `annotation-xml` and `annotation` elements used for representing the annotations themselves. The use of the `semantics`, `annotation` and `annotation-xml` is described in detail Chapter 5 Mixing Markup Languages for Mathematical Expressions.

The `semantics` element is 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.

4.2.9 Error Markup `<cerror>`

	Schema Fragment (Strict)	Schema Fragment (Full)
Class	Error	Error
Attributes	CommonAtt	CommonAtt, DefEncAtt
Content	csymbol, ContExp*	csymbol, ContExp*

A content error expression is made up of a `csymbol` followed by a sequence of zero or more MathML expressions. The initial expression must be a `csymbol` indicating the kind of error. Subsequent children, if present, indicate the context in which the error occurred.

The `cerror` element 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 containing a `DivisionByZero` symbol, as in the following expression:

```
<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>
```

The default presentation of a `cerror` element is an `merror` expression whose first child is a presentation of the error symbol, and whose subsequent children are the default presentations of the remaining children of the cerror. In particular, if one of the remaining children of the cerror is a presentation MathML expression, it is used literally in the corresponding merror.

Content MathML

```
<cerror>
  <csymbol cd="aritherror">DivisionByZero</csymbol>
  <apply><csymbol cd="arith1">divide</csymbol><ci>x</ci><cn>0</cn></apply>
</cerror>
```

Sample Presentation

```
<merror>
  <mtext>DivisionByZero:&#160;</mtext>
  <mfrac><mi>x</mi><mn>0</mn></mfrac>
</merror>
```

DivisionByZero: $\frac{x}{0}$

Note that when the context where an error occurs is so nonsensical that its default presentation would not be useful, an application may provide an alternative representation of the error context. For example:

```
<cerror>
  <csymbol cd="error">Illegal bound variable</csymbol>
  <cs> &lt;bvar&gt;&lt;plus/&gt;&lt;/bvar&gt; </cs>
</cerror>
```

4.2.10 Encoded Bytes <cbytes>

	Schema Fragment (Strict)	Schema Fragment (Full)
Class	Cbytes	Cbytes
Attributes	CommonAtt	CommonAtt, DefEncAtt
Content	base64	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 Content MathML for Specific Operators and Constants.

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 Content MathML for Specific Operators and Constants. 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 Operator Classes.

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 Content MathML for Specific Operators and Constants. The complete algorithm for rewriting arbitrary Content MathML as Strict Content markup is summarized at the end of the Chapter in Section 4.6 The Strict Content MathML Transformation.

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. The markup may define the set elements explicitly by enumerating them, or implicitly by rule, using qualifier elements. However, in either case, the markup for the elements is contained in the markup for the set, and consequently this style of representation is termed *container markup* in MathML. By contrast, Strict markup represents an instance of a set as the result of applying a function or *constructor symbol* to arguments. In this style of markup, the markup for the set construction is a sibling of the markup for the set elements in an enclosing `apply` element.

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 Content MathML for Specific Operators and Constants. Then apply the rewrite rules for that specific operator class as described in Section 4.3.4 Operator Classes.

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>
```

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 Content MathML for Specific Operators and Constants.

4.3.3.1 Uses of <domainofapplication>, <interval>, <condition>, <lowlimit> and <uplimit>

Class	qualifier
Attributes	CommonAtt
Content	ContExp

(For the syntax of interval see Section 4.4.1.1 Interval <interval>.)

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, and thus require a preceding bvar, while the other qualifiers 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 general qualifier is `domainofapplication`, and therefore has a privileged role. It is the preferred form, unless there are particular idiomatic reasons to use one of the other qualifiers, e.g. limits for an integral. In MathML 3, the other forms are treated as shorthand notations for `domainofapplication` because they may all be rewritten as equivalent `domainofapplication` constructions. The rewrite rules to do this are given below. The other qualifier 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`.

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 function 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, and the integrand is a function. 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. Note also that when a bound variable is used, the integrand is an expression in the bound variable, not a function.

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><ci>H</ci>
  <bvar><ci>x</ci></bvar>
  <lowlimit><ci>a</ci></lowlimit>
  <uplimit><ci>b</ci></uplimit>
  <ci>C</ci>
</apply>
```

```
<apply><ci>H</ci>
  <bvar><ci>x</ci></bvar>
  <domainofapplication>
    <apply><csymbol cd="interval1">interval</csymbol>
      <ci>a</ci>
      <ci>b</ci>
    </apply>
  </domainofapplication>
  <ci>C</ci>
</apply>
```

The symbol used in this translation depends on the head of the application, denoted by <ci>H</ci> here. By default interval should be used, unless the semantics of the head term can be determined and indicate a more specific interval symbols. In particular, several predefined Content MathML element should be used with more specific interval symbols. If the head is `int` then oriented_interval is used. When the head term is `sum` or `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><ci>x1</ci></bvar>
<bvar><ci>xn</ci></bvar>
<condition><ci>P</ci></condition>
```

is rewritten to

```
<bvar><ci>x1</ci></bvar>
<bvar><ci>xn</ci></bvar>
<domainofapplication>
  <apply><csymbol cd="set1">suchthat</csymbol>
    <ci>R</ci>
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x1</ci></bvar>
      <bvar><ci>xn</ci></bvar>
      <ci>P</ci>
    </bind>
  </apply>
</domainofapplication>
```

If the `apply` has a `domainofapplication` (perhaps originally expressed as `interval` or an `uplimit`/`lowlimit` pair) then that is used for <ci>R</ci>. Otherwise <ci>R</ci> is a set determined by the `type` attribute of the bound variable as specified in Section 4.2.2.2 Non-Strict uses of `<ci>`, 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><ci>F</ci>
  <domainofapplication>
    <ci>C</ci>
  </domainofapplication>
  <ci>a1</ci>
  <ci>an</ci>
</apply>
```

may be written as:

```
<apply>
  <apply><csymbol cd="fns1">restriction</csymbol>
    <ci>F</ci>
    <ci>C</ci>
  </apply>
  <ci>a1</ci>
  <ci>an</ci>
</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 domainofapplication

A content MathML expression with bound variables and `domainofapplication`

```
<apply><ci>H</ci>
  <bvar><ci>v1</ci></bvar>
  ...
  <bvar><ci>vn</ci></bvar>
  <domainofapplication><ci>D</ci></domainofapplication>
    <ci>A1</ci>
    ...
    <ci>Am</ci>
</apply>
```

is rewritten to

```
<apply><ci>H</ci>
  <ci>D</ci>
  <bind><csymbol cd="fns1">lambda</csymbol>
```

```
    <bvar><ci>v1</ci></bvar>
    ...
    <bvar><ci>vn</ci></bvar>
    <ci>A1</ci>
  </bind>
  ...
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>v1</ci></bvar>
    ...
    <bvar><ci>vn</ci></bvar>
    <ci>Am</ci>
  </bind>
</apply>
```

If there is no `domainofapplication` qualifier the <ci>D</ci> child is omitted.

4.3.3.2 Uses of `<degree>`

Class	qualifier
Attributes	CommonAtt
Content	ContExp

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 Differentiation `<diff/>` and Section 4.4.4.3 Partial Differentiation `<partialdiff/>`.

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 Content MathML for Specific Operators and Constants. 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 Content MathML for Specific Operators and Constants. This is summarized in the following rule.

Rewrite: element

For example,

```
<plus/>
```

is equivalent to the Strict form

```
<csymbol cd="arith1">plus</csymbol>
```

In MathML 2, the `definitionURL` attribute could be used to redefine or modify the meaning of an operator element. When the `definitionURL` attribute is present, the value for the `cd` attribute on the `csymbol` should be determined by the `definitionURL` value if possible. The correspondence between `cd` and `definitionURL` values is described Section 4.2.3.2 Non-Strict uses of `<csymbol>`.

4.3.4.1 N-ary Operators (classes nary-arith, nary-functional, nary-logical, nary-linalg, nary-set, nary-constructor)

Many MathML operators may be used with an arbitrary number of arguments. The corresponding OpenMath symbols for elements in these classes also take an arbitrary number of arguments. In all such cases, either the arguments my be given explicitly as children of the `apply` or `bind` element, or the list may be specified implicitly via the use of qualifier elements.

4.3.4.1.1 Schema Patterns

The elements representing these n-ary operators are specified in the following schema patterns in Appendix A Parsing MathML: nary-arith.class, nary-functional.class, nary-logical.class, nary-linalg.class, nary-set.class, nary-constructor.class.

4.3.4.1.2 Rewriting to Strict Content MathML

If the argument list is given explicitly, the Rewrite: element rule applies.

Any use of qualifier elements is expressed in Strict Content MathML, via explicitly 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 *<ci>expression-in-x</ci>* is an arbitrary expression involving the bound variable(s)

```
<apply><union/>
  <bvar><ci>x</ci></bvar>
  <domainofapplication><ci>D</ci></domainofapplication>
  <ci>expression-in-x</ci>
</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><ci>x</ci></bvar>
      <ci>expression-in-x</ci>
    </bind>
    <ci>D</ci>
  </apply>
</apply>
```

The above rule applies to all symbols in the listed classes. In the case of **nary-set.class** the choice of Content Dictionary to use depends on the type attribute on the arguments, defaulting to set1, but multiset1 should be used if type="multiset".

Note that the members of the **nary-constructor.class**, such as vector, use *constructor* syntax where the arguments and qualifiers are given as children of the element rather than as children of a containing apply. In this case, the above rules apply with the analogous syntactic modifications.

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.

4.3.4.2.1 Schema Patterns

The elements representing these n-ary operators are specified in the schema pattern nary-setlist-constructor.class.

4.3.4.2.2 Rewriting to Strict Content MathML

If the argument list is given explicitly, the Rewrite: element rule applies.

When qualifiers are used to specify the list of arguments, the following rule is used.

Rewrite: n-ary setlist domainofapplication

An expression of the following form, where *<set/>* is either of the elements `set` or `list` and *<ci>expression-in-x</ci>* is an arbitrary expression involving the bound variable(s)

```
<set>
  <bvar><ci>x</ci></bvar>
  <domainofapplication><ci>D</ci></domainofapplication>
  <ci>expression-in-x</ci>
</set>
```

is rewritten to

```
<apply><csymbol cd="set1">map</csymbol>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>x</ci></bvar>
    <ci>expression-in-x</ci>
  </bind>
  <ci>D</ci>
</apply>
```

Note that when *<ci>D</ci>* is already a set or list of the appropriate type for the container element, and the lambda function created from *<ci>expression-in-x</ci>* is the identity, the entire container element should be rewritten directly as *<ci>D</ci>*.

In the case of `set`, the choice of Content Dictionary and symbol depends on the value of the `type` attribute of the arguments. By default the set symbol is used, but if one of the arguments has `type` attribute with value "multiset", the multiset symbol is used. If there is a `type` attribute with value other than "set" or "multiset" the set symbol should be used, and the arguments should be annotated with their type by rewriting the `type` attribute using the rule Rewrite: attributes.

4.3.4.3 N-ary Relations (classes nary-reln, nary-set-reln)

MathML allows transitive 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.

4.3.4.3.1 Schema Patterns

The elements representing these n-ary operators are specified in the following schema patterns in Appendix A Parsing MathML: nary-reln.class, nary-set-reln.class.

4.3.4.3.2 Rewriting to Strict Content MathML

Rewrite: n-ary relations

An expression of the form

```
<apply><lt/>
  <ci>a</ci><ci>b</ci><ci>c</ci><ci>d</ci>
</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>
  <ci>a</ci><ci>b</ci><ci>c</ci><ci>d</ci>
 </apply>
</apply>
```

Rewrite: n-ary relations bvar

An expression of the form

```
<apply><lt/>
 <bvar><ci>x</ci></bvar>
 <domainofapplication><ci>R</ci></domainofapplication>
 <ci>expression-in-x</ci>
</apply>
```

where <ci>*expression-in-x*</ci> is an arbitrary expression involving 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>
   <ci>R</ci>
   <bind><csymbol cd="fns1">lambda</csymbol>
     <bvar><ci>x</ci></bvar>
     <ci>expression-in-x</ci>
   </bind>
  </apply>
</apply>
```

The above rules apply to all symbols in classes **nary-reln.class** and **nary-set-reln.class**. 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 statistical 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 explicitly supplied arguments according to the following rule.

4.3.4.4.1 Schema Patterns

The elements representing these n-ary operators are specified in the following schema patterns in Appendix A Parsing MathML: nary-minmax.class, nary-stats.class.

4.3.4.4.2 Rewriting to Strict Content MathML

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, *<ci>a1</ci><ci>a2</ci><ci>an</ci>*

```
<apply><max/><ci>a1</ci><ci>a2</ci><ci>an</ci></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>
    <ci>a1</ci><ci>a2</ci><ci>an</ci>
  </apply>
</apply>
```

Like all MathML n-ary operators, The list of arguments may be specified implicitly 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 *<ci>expression-in-x</ci>* is an arbitrary expression involving the bound variable(s)

```
<apply><max/>
  <bvar><ci>x</ci></bvar>
  <domainofapplication><ci>D</ci></domainofapplication>
  <ci>expression-in-x</ci>
</apply>
```

is rewritten to

```
<apply><csymbol cd="minmax1">max</csymbol>
  <apply><csymbol cd="set1">map</csymbol>
    <bind><csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <ci>expression-in-x</ci>
    </bind>
    <ci>D</ci>
  </apply>
</apply>
```

Note that when <ci>D</ci> is already a set and the lambda function created from <ci>expression-in-x</ci> is the identity, the `domainofapplication` term should should be rewritten directly as <ci>D</ci>.

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/><ci>a</ci></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> <ci>a</ci> </apply>
```

Note: Earlier versions of MathML were not explicit 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 interpreted 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 explicit 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 operators take two arguments and simply map to OpenMath symbols via Rewrite: element 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 Interval `<interval>`

4.3.4.5.1 Schema Patterns

The elements representing these binary operators are specified in the following schema patterns in Appendix A Parsing MathML: binary-arith.class, binary-logical.class, binary-reln.class, binary-linalg.class, binary-set.class.

4.3.4.6 Unary Operators (classes unary-arith, unary-linalg, unary-functional, unary-set, unary-elementary, unary-veccalc)

Unary operators take a single argument and map to OpenMath symbols via Rewrite: element without the need of any special rewrite rules.

4.3.4.6.1 Schema Patterns

The elements representing these unary operators are specified in the following schema patterns in Appendix A Parsing MathML: unary-arith.class, unary-functional.class, unary-set.class, unary-elementary.class, unary-veccalc.class.

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 Strict Content MathML, they rewrite simply to the corresponding symbol listed in the syntax tables for these elements in Section 4.4.10 Constant and Symbol Elements.

4.3.4.7.1 Schema Patterns

The elements representing these constants are specified in the schema patterns constant-arith.class and constant-set.class.

4.3.4.8 Quantifiers (class quantifier)

The Quantifier class is used for the forall and exists quantifiers of predicate calculus.

4.3.4.8.1 Schema Patterns

The elements representing quantifiers are specified in the schema pattern quantifier.class.

4.3.4.8.2 Rewriting to Strict Content MathML

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 *<ci>expression-in-x</ci>* is an arbitrary expression involving the bound variable(s)

```
<apply><exists/>
  <bvar><ci>x</ci></bvar>
  <domainofapplication><ci>D</ci></domainofapplication>
  <ci>expression-in-x</ci>
</apply>
```

is rewritten to an expression

```
<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><ci>D</ci></apply>
  <ci>expression-in-x</ci>
  </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`.) When no `domainofapplication` is present, no logical conjunction is necessary, and the translation is direct.

4.3.4.9 Other Operators (classes lambda, interval, int, diff partialdiff, sum, product, limit)

Special purpose classes, described in the sections for the appropriate elements

4.3.4.9.1 Schema Patterns

The elements are specified in the following schema patterns in Appendix A Parsing MathML: lambda.class, interval.class, int.class, partialdiff.class, sum.class, product.class, limit.class.

4.3.5 Non-strict Attributes

A number of content MathML elements such as `cn` and `interval` allow attributes to specialize the semantics of the objects they represent. For these cases, special rewrite rules are given on a case-by-case basis in Section 4.4 Content MathML for Specific Operators and Constants. However, content MathML elements also accept attributes shared all MathML elements, and depending on the context, may also contain attributes from other XML namespaces. Such attributes must be rewritten in alternative form in Strict Content Markup.

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/plain">foo</annotation>
  <annotation-xml cd="mathmlattr" name="foreign" encoding="MathML-Content">
    <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.

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 rewriting 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.

4.4.1 Functions and Inverses

4.4.1.1 Interval `<interval>`

Class	interval
Attributes	CommonAtt, DefEncAtt, 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, DefEncAtt
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;<!--FUNCTION APPLICATION--></mo>
 <mfenced><mi>a</mi></mfenced>
</mrow>
```

$$A^{(-1)}(a)$$

4.4.1.3 Lambda `<lambda>`

Class	lambda
Attributes	CommonAtt, DefEncAtt
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 its 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;<!--GREEK SMALL LETTER LAMDA--></mi>
 <mi>x</mi>
 <mo>.</mo>
 <mfenced>
  <mrow>
   <mi>sin</mi>
   <mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--RIGHTWARDS ARROW FROM BAR--></mo>
  <mrow>
   <mi>sin</mi>
   <mo>&#x2061;<!--FUNCTION APPLICATION--></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><ci>x1</ci></bvar><bvar><ci>xn</ci></bvar>
  <ci>expression-in-x1-xn</ci>
</lambda>
```

rewrites to the Strict Content MathML

```
<bind><csymbol cd="fns1">lambda</csymbol>
  <bvar><ci>x1</ci></bvar><bvar><ci>xn</ci></bvar>
  <ci>expression-in-x1-xn</ci>
</bind>
```

Rewrite: lambda domainofapplication

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 restricted to the specified domain using restriction.

```
<lambda>
  <bvar><ci>x1</ci></bvar><bvar><ci>xn</ci></bvar>
  <domainofapplication><ci>D</ci></domainofapplication>
  <ci>expression-in-x1-xn</ci>
</lambda>
```

rewrites to the Strict Content MathML

```
<apply><csymbol cd="fns1">restriction</csymbol>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>x1</ci></bvar><bvar><ci>xn</ci></bvar>
    <ci>expression-in-x1-xn</ci>
  </bind>
  <ci>D</ci>
</apply>
```

4.4.1.4 Function composition `<compose/>`

Class	nary-functional
Attributes	CommonAtt, DefEncAtt
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 N-ary Operators (classes nary-arith, nary-functional, nary-logical, nary-linalg, nary-set, nary-constructor).

Content MathML

```
<apply><compose/><ci>f</ci><ci>g</ci><ci>h</ci></apply>
```

Sample Presentation

```
<mrow><mi>f</mi><mo>&#x2218;<!--RING OPERATOR--></mo><mi>g</mi><mo>&#x2218;<!--RING OPERATOR--></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;<!--RING OPERATOR--></mo><mi>g</mi><mo>)</mo></mrow>
  <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
  <mfenced><mi>x</mi></mfenced>
 </mrow>
 <mo>=</mo>
 <mrow>
```

```
  <mi>f</mi>
  <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
  <mfenced>
   <mrow>
    <mi>g</mi>
    <mo>&#x2061;<!--FUNCTION APPLICATION--></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, DefEncAtt
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;<!--RING OPERATOR--></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, DefEncAtt
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;<!--FUNCTION APPLICATION--></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, DefEncAtt
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;<!--FUNCTION APPLICATION--></mo><mfenced>
                                                <mi>f</mi></mfenced></mrow>
 <mo>=</mo>
 <mi mathvariant="double-struck">Q</mi>
</mrow>
```

$$\mathrm{codomain}(f) = \mathbb{Q}$$

4.4.1.8 Image `<image/>`

Class	unary-functional
Attributes	CommonAtt, DefEncAtt
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;<!--FUNCTION APPLICATION--></mo><mfenced>
                                            <mi>sin</mi></mfenced></mrow>
 <mo>=</mo>
 <mfenced open="[" close="]"><mn>-1</mn><mn>1</mn></mfenced>
</mrow>
```

$$\mathrm{image}(\sin) = [-1, 1]$$

4.4.1.9 Piecewise declaration `<piecewise>`, `<piece>`, `<otherwise>`

Class	Constructor
Attributes	CommonAtt, DefEncAtt
Content	piece* otherwise?
OM Symbols	piecewise

Syntax Table for `piecewise`

Class	Constructor
Attributes	CommonAtt, DefEncAtt
Content	ContExp ContExp
OM Symbols	piece

Syntax Table for `piece`

Class	Constructor
Attributes	CommonAtt, DefEncAtt
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.

Here is an example:

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;<!--MINUS SIGN--></mo><mi>x</mi></mrow></mtd>
   <mtd columnalign="left"><mtext>&#xa0;<!--NO-BREAK SPACE--> if &#xa0;<!--NO-BREAK SPACE--></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>&#xa0;<!--NO-BREAK SPACE--> if &#xa0;<!--NO-BREAK SPACE--></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>&#xa0;<!--NO-BREAK SPACE--> if &#xa0;<!--NO-BREAK SPACE--></mtext></mtd>
   <mtd><mrow><mi>x</mi><mo>&gt;</mo><mn>0</mn></mrow></mtd>
  </mtr>
 </mtable>
</mrow>
```

$$\begin{cases} -x & \text{if} \quad x < 0 \\ 0 & \text{if} \quad x = 0 \\ x & \text{if} \quad x > 0 \end{cases}$$

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>
    <cn>0</cn>
    <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>
```

4.4.2 Arithmetic, Algebra and Logic

4.4.2.1 Quotient `<quotient/>`

Class	binary-arith
Attributes	CommonAtt, DefEncAtt
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 $a = b \times q + r$, with $|r|$ less than $|b|$ and $a \times r$ 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;<!--LEFT FLOOR--></mo><mi>a</mi><mo>/</mo><mi>b</mi>
                                        <mo>&#x230b;<!--RIGHT FLOOR--></mo></mrow>
```

$\lfloor a/b \rfloor$

4.4.2.2 Factorial `<factorial/>`

Class	unary-arith
Attributes	CommonAtt, DefEncAtt
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 $n! = n \times (n-1) \times \cdots \times 1$.

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, DefEncAtt
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, DefEncAtt
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 qualifier elements as described in Section 4.3.4.4 N-ary/Unary Operators (classes nary-minmax, nary-stats). 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>
   <msup><mi>y</mi><mn>3</mn></msup>
   <mo>&#x2208;<!--ELEMENT OF--></mo>
   <mfenced open="[" close="]"><mn>0</mn><mn>1</mn></mfenced>
  </mrow>
  <mo>}</mo>
 </mrow>
</mrow>
```

$$\max\{y\,|\,y^3 \in [0,1]\}$$

4.4.2.5 Minimum `<min/>`

Class	nary-minmax
Attributes	CommonAtt, DefEncAtt
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 qualifier elements as described in Section 4.3.4.4 N-ary/Unary Operators (classes nary-minmax, nary-stats). 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><msup><mi>x</mi><mn>2</mn></msup><mo>|</mo>
  <mrow><mi>x</mi><mo>&#x2209;<!--NOT AN ELEMENT OF--></mo><mi>B</mi></mrow>
  <mo>}</mo>
</mrow>
</mrow>
```

$$\min\{x^2 \mid x \notin B\}$$

4.4.2.6 Subtraction `<minus/>`

Class	unary-arith, binary-arith
Attributes	CommonAtt, DefEncAtt
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;<!--MINUS SIGN--></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;<!--MINUS SIGN--></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, DefEncAtt
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, DefEncAtt
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, DefEncAtt
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 $a = b \times q + r$, with $|r|$ less than $|b|$ and $a \times r$ 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, DefEncAtt
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;<!--INVISIBLE TIMES--></mo><mi>b</mi></mrow>
```

$$ab$$

4.4.2.11 Root `<root/>`

Class	unary-arith, binary-arith
Attributes	CommonAtt, DefEncAtt
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, DefEncAtt
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 N-ary Operators (classes nary-arith, nary-functional, nary-logical, nary-linalg, nary-set, nary-constructor).

Content MathML

```
<apply><gcd/><ci>a</ci><ci>b</ci><ci>c</ci></apply>
```

Sample Presentation

```
<mrow>
 <mi>gcd</mi>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mfenced><mi>a</mi><mi>b</mi><mi>c</mi></mfenced>
</mrow>
```

$\mathrm{gcd}(a, b, c)$

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, DefEncAtt
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 N-ary Operators (classes nary-arith, nary-functional, nary-logical, nary-linalg, nary-set, nary-constructor).

Content MathML

```
<apply><and/><ci>a</ci><ci>b</ci></apply>
```

Sample Presentation

```
<mrow><mi>a</mi><mo>&#x2227;<!--LOGICAL AND--></mo><mi>b</mi></mrow>
```

$a \land b$

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;<!--N-ARY LOGICAL AND--></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, DefEncAtt
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;<!--LOGICAL OR--></mo><mi>b</mi></mrow>
```

$$a \vee b$$

4.4.2.15 Exclusive Or `<xor/>`

Class	nary-logical
Attributes	CommonAtt, DefEncAtt
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, DefEncAtt
Content	Empty
OM Symbols	not

The `not` 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;<!--NOT SIGN--></mo><mi>a</mi></mrow>
```

$$\neg a$$

4.4.2.17 Implies `<implies/>`

Class	binary-logical
Attributes	CommonAtt, DefEncAtt
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;<!--RIGHTWARDS DOUBLE ARROW--></mo><mi>B</mi></mrow>
```

$$A \Rightarrow B$$

4.4.2.18 Universal quantifier `<forall/>`

Class	quantifier
Attributes	CommonAtt, DefEncAtt
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 assertion being quantified. In addition, `condition` or other qualifiers may be used as described in Section 4.3.4.8 Quantifiers (class quantifier) 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;<!--FOR ALL--></mo>
 <mi>x</mi>
 <mo>.</mo>
 <mfenced>
  <mrow>
   <mrow><mi>x</mi><mo>&#x2212;<!--MINUS SIGN--></mo><mi>x</mi></mrow>
   <mo>=</mo>
   <mn>0</mn>
  </mrow>
 </mfenced>
</mrow>
```

$$\forall x\,.\,(x - x = 0)$$

Mapping to Strict Markup

When the `forall` element is used with a `condition` qualifier the strict equivalent is constructed with the help of logical implication by the rule Rewrite: quantifier. Thus

```
<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;<!--FOR ALL--></mo>
 <mrow>
  <mrow><mi>p</mi><mo>&#x2208;<!--ELEMENT OF--></mo><mi
```

```
                                  mathvariant="double-struck">Q</mi></mrow>
  <mo>&#x2227;<!--LOGICAL AND--></mo>
  <mrow><mi>q</mi><mo>&#x2208;<!--ELEMENT OF--></mo><mi
                                  mathvariant="double-struck">Q</mi></mrow>
  <mo>&#x2227;<!--LOGICAL AND--></mo>
  <mrow><mo>(</mo><mi>p</mi><mo>&lt;</mo><mi>q</mi><mo>)</mo></mrow>
 </mrow>
 <mo>.</mo>
 <mfenced>
  <mrow><mi>p</mi><mo>&lt;</mo><msup><mi>q</mi><mn>2</mn></msup></mrow>
 </mfenced>
</mrow>
```

$$\forall p \in \mathbb{Q} \wedge q \in \mathbb{Q} \wedge (p < q) \,.\, (p < q^2)$$

```
<mrow>
 <mo>&#x2200;<!--FOR ALL--></mo>
 <mrow><mi>p</mi><mo>,</mo><mi>q</mi></mrow>
 <mo>.</mo>
 <mfenced>
  <mrow>
   <mrow>
    <mo>(</mo>
    <mrow>
     <mi>p</mi><mo>&#x2208;<!--ELEMENT OF--></mo><mi
                                         mathvariant="double-struck">Q</mi>
    </mrow>
    <mo>&#x2227;<!--LOGICAL AND--></mo>
    <mrow>
     <mi>q</mi><mo>&#x2208;<!--ELEMENT OF--></mo><mi
                                         mathvariant="double-struck">Q</mi>
    </mrow>
    <mo>&#x2227;<!--LOGICAL AND--></mo>
    <mrow><mo>(</mo><mi>p</mi><mo>&lt;</mo><mi>q</mi><mo>)</mo></mrow>
    <mo>)</mo>
   </mrow>
   <mo>&#x21d2;<!--RIGHTWARDS DOUBLE ARROW--></mo>
   <mrow>
    <mo>(</mo>
    <mi>p</mi>
    <mo>&lt;</mo>
    <msup><mi>q</mi><mn>2</mn></msup>
    <mo>)</mo>
   </mrow>
  </mrow>
 </mfenced>
</mrow>
```

$$\forall p, q \,.\, ((p \in \mathbb{Q} \wedge q \in \mathbb{Q} \wedge (p < q)) \Rightarrow (p < q^2))$$

4.4.2.19 Existential quantifier `<exists/>`

Class	quantifier
Attributes	CommonAtt, DefEncAtt
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 Quantifiers (class quantifier) 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;<!--THERE EXISTS--></mo>
 <mi>x</mi>
 <mo>.</mo>
 <mfenced>
  <mrow>
   <mrow><mi>f</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--THERE EXISTS--></mo>
 <mi>x</mi>
 <mo>.</mo>
 <mfenced separators="">
  <mrow><mi>x</mi><mo>&#x2208;<!--ELEMENT OF--></mo><mi
                                   mathvariant="double-struck">Z</mi></mrow>
  <mo>&#x2227;<!--LOGICAL AND--></mo>
  <mrow>
   <mrow><mi>f</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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, DefEncAtt
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, DefEncAtt
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;<!--DOUBLE-STRUCK ITALIC SMALL I--></cn><ci>y</ci>
                                                                  </apply>
  </apply>
</apply>
```

Sample Presentation

```
<mover>
 <mrow>
  <mi>x</mi>
  <mo>+</mo>
  <mrow><mn>&#x2148;<!--DOUBLE-STRUCK ITALIC SMALL I--></mn><mo>&#x2062;<!--INVISIBLE TIMES-->
                                                       </mo><mi>y</mi></mrow>
 </mrow>
 <mo>&#xaf;<!--MACRON--></mo>
</mover>
```

$\overline{x + ⅈ y}$

4.4.2.22 Argument `<arg/>`

Class	unary-arith
Attributes	CommonAtt, DefEncAtt
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;<!--FUNCTION APPLICATION--></mo>
 <mfenced>
  <mrow>
   <mi>x</mi>
   <mo>+</mo>
   <mrow><mi>i</mi><mo>&#x2062;<!--INVISIBLE TIMES--></mo><mi>y</mi></mrow>
  </mrow>
 </mfenced>
</mrow>
```

$$\arg(x + iy)$$

4.4.2.23 Real part `<real/>`

Class	unary-arith
Attributes	CommonAtt, DefEncAtt
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;<!--SCRIPT CAPITAL R--></mo>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mfenced>
  <mrow>
   <mi>x</mi>
   <mo>+</mo>
```

```
    <mrow><mi>i</mi><mo>&#x2062;<!--INVISIBLE TIMES--></mo><mi>y</mi></mrow>
   </mrow>
  </mfenced>
 </mrow>
```

$\mathcal{R}\left(x + iy\right)$

4.4.2.24 Imaginary part `<imaginary/>`

Class	unary-arith
Attributes	CommonAtt, DefEncAtt
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;<!--BLACK-LETTER CAPITAL I--></mo>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mfenced>
  <mrow>
   <mi>x</mi>
   <mo>+</mo>
   <mrow><mi>i</mi><mo>&#x2062;<!--INVISIBLE TIMES--></mo><mi>y</mi></mrow>
  </mrow>
 </mfenced>
</mrow>
```

$\mathfrak{I}\left(x + iy\right)$

4.4.2.25 Lowest common multiple `<lcm/>`

Class	nary-arith
Attributes	CommonAtt, DefEncAtt
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;<!--FUNCTION APPLICATION--></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, DefEncAtt
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;<!--LEFT FLOOR--></mo><mi>a</mi><mo>&#x230b;<!--RIGHT FLOOR--></mo>
                                                                              </mrow>
```

$\lfloor a \rfloor$

4.4.2.27 Ceiling `<ceiling/>`

Class	unary-arith
Attributes	CommonAtt, DefEncAtt
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;<!--LEFT CEILING--></mo><mi>a</mi><mo>&#x2309;<!--RIGHT CEILING--></mo>
                                                                                  </mrow>
```

$\lceil a \rceil$

4.4.3 Relations

4.4.3.1 Equals `<eq/>`

Class	nary-reln
Attributes	CommonAtt, DefEncAtt
Content	Empty
Qualifiers	BvarQ, DomainQ
OM Symbols	eq

The `eq` elements represents the equality relation. While equality is a binary relation, `eq` may be used with more than two arguments, denoting a chain of equalities, as described in Section 4.3.4.3 N-ary Relations (classes nary-reln, nary-set-reln).

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, DefEncAtt
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;<!--NOT EQUAL TO--></mo><mn>4</mn></mrow>
```

$3 \neq 4$

4.4.3.3 Greater than `<gt/>`

Class	nary-reln
Attributes	CommonAtt, DefEncAtt
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 N-ary Relations (classes nary-reln, nary-set-reln).

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, DefEncAtt
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 N-ary Relations (classes nary-reln, nary-set-reln).

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, DefEncAtt
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 N-ary Relations (classes nary-reln, nary-set-reln).

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;<!--GREATER-THAN OR EQUAL TO--></mo><mn>3</mn>
                   <mo>&#x2265;<!--GREATER-THAN OR EQUAL TO--></mo><mn>3</mn></mrow>
```

$$4 \geq 3 \geq 3$$

4.4.3.6 Less Than or Equal `<leq/>`

Class	nary-reln
Attributes	CommonAtt, DefEncAtt
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 N-ary Relations (classes nary-reln, nary-set-reln).

Content MathML

```
<apply><leq/><cn>3</cn><cn>3</cn><cn>4</cn></apply>
```

Sample Presentation

```
<mrow><mn>3</mn><mo>&#x2264;<!--LESS-THAN OR EQUAL TO--></mo><mn>3</mn>
                                  <mo>&#x2264;<!--LESS-THAN OR EQUAL TO--></mo><mn>4</mn></mrow>
```

$$3 \leq 3 \leq 4$$

4.4.3.7 Equivalent `<equivalent/>`

Class	binary-logical
Attributes	CommonAtt, DefEncAtt
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;<!--IDENTICAL TO--></mo>
 <mrow><mo>&#xac;<!--NOT SIGN--></mo><mrow><mo>&#xac;<!--NOT SIGN--></mo><mi>a</mi>
                                                                     </mrow></mrow>
</mrow>
```

$$a \equiv \neg\neg a$$

4.4.3.8 Approximately `<approx/>`

Class	binary-reln
Attributes	CommonAtt, DefEncAtt
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;<!--GREEK SMALL LETTER PI--></mi>
 <mo>&#x2243;<!--ASYMPTOTICALLY EQUAL TO--></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, DefEncAtt
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, DefEncAtt
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;<!--INTEGRAL--></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;<!--INTEGRAL--></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;<!--INTEGRAL--></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.

When no bound variables are present, the translation of an indefinite integral to Strict Content Markup is straight forward. When bound variables are present, the following rule should be used.

Rewrite: int

Translate an indefinite integral, where *<ci>expression-in-x</ci>* is an arbitrary expression involving the bound variable(s) *<ci>x</ci>*

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <ci>expression-in-x</ci>
</apply>
```

to the expression

```
<apply>
  <apply><csymbol cd="calculus1">int</csymbol>
    <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>x</ci></bvar>
    <ci>expression-in-x</ci>
    </bind>
  </apply>
  <ci>x</ci>
</apply>
```

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 int.

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>
```

For a definite integral without bound variables, the translation is also straightforward.

For instance, the integral of a differential form f over an arbitrary domain C represented as

```
<apply><int/>
  <domainofapplication><ci>C</ci></domainofapplication>
  <ci>f</ci>
</apply>
```

is equivalent to the Strict Content MathML:

```
<apply><csymbol cd="calculus1">defint</csymbol><ci>C</ci><ci>f</ci></apply>
```

Note, however, the additional remarks on the translations of other kinds of qualifiers that may be used to specify a domain of integration in the rules for definite integrals following.

When bound variables are present, the situation is more complicated in general, and the following rules are used.

Rewrite: defint

Translate a definite integral, where *<ci>expression-in-x</ci>* is an arbitrary expression involving the bound variable(s) *<ci>x</ci>*

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <domainofapplication><ci>D</ci></domainofapplication>
  <ci>expression-in-x</ci>
</apply>
```

to the expression

```
<apply><csymbol cd="calculus1">defint</csymbol>
  <ci>D</ci>
  <bind><csymbol cd="fns1">lambda</csymbol>
  <bvar><ci>x</ci></bvar>
  <ci>expression-in-x</ci>
  </bind>
</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 accommodate this, use the following special translation rule:

Rewrite: defint limits

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <lowlimit><ci>a</ci></lowlimit>
  <uplimit><ci>b</ci></uplimit>
  <ci>expression-in-x</ci>
</apply>
```

where <ci>expression-in-x</ci> is an expression in the variable x is translated to to the expression:

```
<apply><csymbol cd="calculus1">defint</csymbol>
  <apply><csymbol cd="interval1">oriented_interval</csymbol>
    <ci>a</ci>  <ci>b</ci>
  </apply>
  <bind><csymbol cd="fns1">lambda</csymbol>
  <bvar><ci>x</ci></bvar>
  <ci>expression-in-x</ci>
  </bind>
</apply>
```

The oriented_interval symbol is also used when translating the `interval` qualifier, when it is used to specify the domain of integration. Integration is assumed to proceed from the left endpoint to the right endpoint.

The case for multiple integrands is treated analogously.

Note that use of the `condition` qualifier also requires special treatment. In particular, it 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

```
<apply><csymbol cd="calculus1">defint</csymbol>
 <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>
  <bind><csymbol cd="fns11">lambda</csymbol>
   <bvar><ci>x</ci></bvar>
   <bvar><ci>y</ci></bvar>
   <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>
  </bind>
 </apply>
</apply>
```

4.4.4.2 Differentiation `<diff/>`

Class	Differential-Operator
Attributes	CommonAtt, DefEncAtt
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;<!--PRIME--></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;<!--FUNCTION APPLICATION--></mo>
                                                   <mi>x</mi></mrow></mrow>
  <mrow><mi>d</mi><mi>x</mi></mrow>
 </mfrac>
 <mo>=</mo>
 <mrow><mi>cos</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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>
```

$$\frac{d^2x^4}{dx^2}$$

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><ci>x</ci></bvar>
  <ci>expression-in-x</ci>
</apply>
```

where *<ci>expression-in-x</ci>* is an expression in the variable x to the expression

```
<apply>
  <apply><csymbol cd="calculus1">diff</csymbol>
    <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>x</ci></bvar>
    <ci>E</ci>
    </bind>
  </apply>
  <ci>x</ci>
</apply>
```

Note that 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><ci>x</ci><degree><ci>n</ci></degree></bvar>
  <ci>expression-in-x</ci>
</apply>
```

where *<ci>expression-in-x</ci>* is an is an expression in the variable x is translated to to the expression:

```
<apply>
  <apply><csymbol cd="calculus1">nthdiff</csymbol>
    <ci>n</ci>
    <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>x</ci></bvar>
    <ci>expression-in-x</ci>
    </bind>
  </apply>
  <ci>x</ci>
</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, DefEncAtt
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>z</ci></bvar>
   <apply><ci>f</ci><ci>x</ci><ci>y</ci><ci>z</ci></apply>
  </lambda>
</apply>
```

Sample Presentation

```
<mfrac>
 <mrow>
  <msup><mo>&#x2202;<!--PARTIAL DIFFERENTIAL--></mo><mn>3</mn></msup>
  <mrow>
   <mi>f</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></mo><mfenced><mi>x</mi>
                                             <mi>y</mi><mi>z</mi></mfenced>
 </mrow>
 </mrow>
 <mrow>
  <mrow><mo>&#x2202;<!--PARTIAL DIFFERENTIAL--></mo><msup><mi>x</mi><mn>2</mn>
                                                             </msup></mrow>
  <mrow><mo>&#x2202;<!--PARTIAL DIFFERENTIAL--></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;<!--PARTIAL DIFFERENTIAL--></mo><mn>2</mn></msup>
  <mrow>
   <mi>f</mi>
   <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
   <mfenced><mi>x</mi><mi>y</mi></mfenced>
  </mrow>
 </mrow>
 <mrow>
  <mrow><mo>&#x2202;<!--PARTIAL DIFFERENTIAL--></mo><mi>x</mi></mrow>
  <mrow><mo>&#x2202;<!--PARTIAL DIFFERENTIAL--></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;<!--PARTIAL DIFFERENTIAL--></mo><mi>k</mi></msup>
  <mrow>
   <mi>f</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></mo><mfenced><mi>x</mi>
                                                           <mi>y</mi></mfenced>
  </mrow>
 </mrow>
 <mrow>
  <mrow><mo>&#x2202;<!--PARTIAL DIFFERENTIAL--></mo><msup><mi>x</mi><mi>m</mi>
                                                                  </msup></mrow>
  <mrow><mo>&#x2202;<!--PARTIAL DIFFERENTIAL--></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><ci>x1</ci><degree><ci>n1</ci></degree></bvar>
  <bvar><ci>xk</ci><degree><ci>nk</ci></degree></bvar>
  <degree><ci>total-n1-nk</ci></degree>
  <ci>expression-in-x1-xk</ci>
</apply>
```

<ci>expression-in-x1-xk</ci> is an arbitrary expression involving the bound variables.

```
<apply>
  <apply><csymbol cd="calculus1">partialdiffdegree</csymbol>
    <apply><csymbol cd="list1">list</csymbol>
      <ci>n1</ci>  <ci>nk</ci>
    </apply>
    <ci>total-n1-nk</ci>
    <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>x1</ci></bvar>
    <bvar><ci>xk</ci></bvar>
    <ci>expression-in-x1-xk</ci>
   </bind>
  </apply>
  <ci>x1</ci>
  <ci>xk</ci>
</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>
  <ci>n1</ci>  <ci>nk</ci>
</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, DefEncAtt
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;<!--FUNCTION APPLICATION--></mo><mfenced><mi>a</mi>
                                                              </mfenced></mrow>
```

$$\mathrm{div}(a)$$

Content MathML

```
<apply><divergence/>
  <ci type="vector">E</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>div</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></mo><mfenced><mi>E</mi>
                                                              </mfenced></mrow>
```

$$\mathrm{div}(E)$$

```
<mrow><mo>&#x2207;<!--NABLA--></mo><mo>&#x22c5;<!--DOT OPERATOR--></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;<!--FUNCTION APPLICATION--></mo>
 <mo>(</mo>
 <mtable>
  <mtr><mtd>
   <mi>x</mi>
   <mo>&#x21a6;<!--RIGHTWARDS ARROW FROM BAR--></mo>
   <mrow><mi>x</mi><mo>+</mo><mi>y</mi></mrow>
  </mtd></mtr>
  <mtr><mtd>
   <mi>y</mi>
   <mo>&#x21a6;<!--RIGHTWARDS ARROW FROM BAR--></mo>
   <mrow><mi>x</mi><mo>+</mo><mi>z</mi></mrow>
  </mtd></mtr>
  <mtr><mtd>
   <mi>z</mi>
   <mo>&#x21a6;<!--RIGHTWARDS ARROW FROM BAR--></mo>
   <mrow><mi>z</mi><mo>+</mo><mi>y</mi></mrow>
  </mtd></mtr>
 </mtable>
 <mo>)</mo>
</mrow>
```

$$\mathrm{div}\begin{pmatrix} x \mapsto x+y \\ y \mapsto x+z \\ z \mapsto z+y \end{pmatrix}$$

4.4.4.5 Gradient `<grad/>`

Class	unary-veccalc
Attributes	CommonAtt, DefEncAtt
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;<!--FUNCTION APPLICATION--></mo><mfenced><mi>f</mi>
                                                                  </mfenced></mrow>
```

$$\mathrm{grad}(f)$$

```
<mrow><mo>&#x2207;<!--NABLA--></mo><mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--FUNCTION APPLICATION--></mo>
 <mrow>
  <mo>(</mo>
  <mfenced><mi>x</mi><mi>y</mi><mi>z</mi></mfenced>
  <mo>&#x21a6;<!--RIGHTWARDS ARROW FROM BAR--></mo>
  <mrow>
   <mi>x</mi><mo>&#x2062;<!--INVISIBLE TIMES--></mo><mi>y</mi><mo>&#x2062;<!--INVISIBLE
                                                              TIMES--></mo><mi>z</mi>
  </mrow>
  <mo>)</mo>
  </mrow>
</mrow>
```

$$\mathrm{grad}((x,y,z)\mapsto xyz)$$

4.4.4.6 Curl `<curl/>`

Class	unary-veccalc
Attributes	CommonAtt, DefEncAtt
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;<!--FUNCTION APPLICATION--></mo><mfenced><mi>a</mi>
                                                                  </mfenced></mrow>
```

$$\mathrm{curl}(a)$$

```
<mrow><mo>&#x2207;<!--NABLA--></mo><mo>&#xd7;<!--MULTIPLICATION SIGN--></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, DefEncAtt
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;<!--NABLA--></mo><mn>2</mn></msup>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--NABLA--></mo><mn>2</mn></msup>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mrow>
  <mo>(</mo>
  <mfenced><mi>x</mi><mi>y</mi><mi>z</mi></mfenced>
  <mo>&#x21a6;<!--RIGHTWARDS ARROW FROM BAR--></mo>
  <mrow>
   <mi>f</mi>
   <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
   <mfenced><mi>x</mi><mi>y</mi></mfenced>
  </mrow>
  <mo>)</mo>
 </mrow>
</mrow>
```

$$\nabla^2((x,y,z)\mapsto f(x,y))$$

4.4.5 Theory of Sets

4.4.5.1 Set `<set>`

Class	nary-setlist-constructor
Attributes	CommonAtt, DefEncAtt, type?
`type` Attribute Values	"set" | "multiset" | text
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 Container Markup for Constructor Symbols. 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>
```

$$\{x \mid x < 5\}$$

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;<!--ELEMENT OF--></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;<!--LOGICAL AND--></mo>
  <mrow>
   <mi>x</mi><mo>&#x2208;<!--ELEMENT OF--></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, DefEncAtt, order
`order` Attribute Values	"numeric" | "lexicographic"
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>
```

$$(x \mid x < 5)$$

4.4.5.3 Union `<union/>`

Class	nary-set
Attributes	CommonAtt, DefEncAtt
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;<!--UNION--></mo><mi>B</mi></mrow>
```

$$A \cup B$$

Arguments may also be specified using qualifier elements as described in Section 4.3.4.1 N-ary Operators (classes nary-arith, nary-functional, nary-logical, nary-linalg, nary-set, nary-constructor). 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;<!--N-ARY UNION--></mo><mi>L</mi></munder><mi>S</mi>
                                                                       </mrow>
```

$$\bigcup_{L} S$$

4.4.5.4 Intersect `<intersect/>`

Class	nary-set
Attributes	CommonAtt, DefEncAtt
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 qualifier elements as described in Section 4.3.4.1 N-ary Operators (classes nary-arith, nary-functional, nary-logical, nary-linalg, nary-set, nary-constructor).

Content MathML

```
<apply><intersect/>
  <ci type="set"> A </ci>
  <ci type="set"> B </ci>
</apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#x2229;<!--INTERSECTION--></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;<!--N-ARY INTERSECTION--></mo><mi>L</mi></munder>
                                                            <mi>S</mi></mrow>
```

$$\bigcap_{L} S$$

4.4.5.5 Set inclusion `<in/>`

Class	binary-set
Attributes	CommonAtt, DefEncAtt
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;<!--ELEMENT OF--></mo><mi>A</mi></mrow>
```

$$a \in A$$

When translating to Strict Content Markup, if the `type` has value "multiset", then the in symbol from multiset1 should be used instead.

4.4.5.6 Set exclusion `<notin/>`

Class	binary-set
Attributes	CommonAtt, DefEncAtt
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;<!--NOT AN ELEMENT OF--></mo><mi>A</mi></mrow>
```

$$a \notin A$$

When translating to Strict Content Markup, if the `type` has value "multiset", then the in symbol from multiset1 should be used instead.

4.4.5.7 Subset `<subset/>`

Class	nary-set-reln
Attributes	CommonAtt, DefEncAtt
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 N-ary Relations (classes nary-reln, nary-set-reln), 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;<!--SUBSET OF OR EQUAL TO--></mo><mi>B</mi></mrow>
```

$$A \subseteq B$$

4.4.5.8 Proper Subset `<prsubset/>`

Class	nary-set-reln
Attributes	CommonAtt, DefEncAtt
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 N-ary Relations (classes nary-reln, nary-set-reln), 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;<!--SUBSET OF--></mo><mi>B</mi></mrow>
```

$$A \subset B$$

4.4.5.9 Not Subset `<notsubset/>`

Class	binary-set
Attributes	CommonAtt, DefEncAtt
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;<!--NEITHER A SUBSET OF NOR EQUAL TO--></mo><mi>B</mi>
                                                                          </mrow>
```

$A \nsubseteq B$

When translating to Strict Content Markup, if the type has value "multiset", then the in symbol from multiset1 should be used instead.

4.4.5.10 Not Proper Subset `<notprsubset/>`

Class	binary-set
Attributes	CommonAtt, DefEncAtt
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;<!--NOT A SUBSET OF--></mo><mi>B</mi></mrow>
```

$A \not\subset B$

When translating to Strict Content Markup, if the type has value "multiset", then the in symbol from multiset1 should be used instead.

4.4.5.11 Set Difference `<setdiff/>`

Class	binary-set
Attributes	CommonAtt, DefEncAtt
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;<!--SET MINUS--></mo><mi>B</mi></mrow>
```

$A \setminus B$

When translating to Strict Content Markup, if the `type` has value "multiset", then the in symbol from multiset1 should be used instead.

4.4.5.12 Cardinality `<card/>`

Class	unary-set
Attributes	CommonAtt, DefEncAtt
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 symbol from multiset1 should be used instead.

4.4.5.13 Cartesian product `<cartesianproduct/>`

Class	nary-set
Attributes	CommonAtt, DefEncAtt
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 qualifier elements as described in Section 4.3.4.1 N-ary Operators (classes nary-arith, nary-functional, nary-logical, nary-linalg, nary-set, nary-constructor).

Content MathML

```
<apply><cartesianproduct/><ci>A</ci><ci>B</ci></apply>
```

Sample Presentation

```
<mrow><mi>A</mi><mo>&#xd7;<!--MULTIPLICATION SIGN--></mo><mi>B</mi></mrow>
```

$A \times B$

4.4.6 Sequences and Series

4.4.6.1 Sum `<sum/>`

Class	sum
Attributes	CommonAtt, DefEncAtt
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;<!--N-ARY SUMMATION--></mo>
  <mrow><mi>x</mi><mo>=</mo><mi>a</mi></mrow>
  <mi>b</mi>
 </munderover>
 <mrow><mi>f</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--N-ARY SUMMATION--></mo>
  <mrow><mi>x</mi><mo>&#x2208;<!--ELEMENT OF--></mo><mi>B</mi></mrow>
 </munder>
 <mrow><mi>f</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--N-ARY SUMMATION--></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, DefEncAtt
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;<!--N-ARY PRODUCT--></mo>
  <mrow><mi>x</mi><mo>=</mo><mi>a</mi></mrow>
  <mi>b</mi>
 </munderover>
 <mrow><mi>f</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--N-ARY PRODUCT--></mo>
  <mrow><mi>x</mi><mo>&#x2208;<!--ELEMENT OF--></mo><mi>B</mi></mrow>
 </munder>
 <mrow><mi>f</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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, DefEncAtt
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;<!--RIGHTWARDS ARROW--></mo><mn>0</mn></mrow>
 </munder>
 <mrow><mi>sin</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--RIGHTWARDS ARROW--></mo><mn>0</mn></mrow>
 </munder>
 <mrow><mi>sin</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--RIGHTWARDS ARROW--></mo><msup><mi>a</mi>
                                                 <mo>+</mo></msup></mrow>
 </munder>
 <mrow><mi>sin</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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><ci>x</ci></bvar>
  <condition>
    <apply><tendsto/><ci>x</ci><cn>0</cn></apply>
  </condition>
  <ci>expression-in-x</ci>
</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><ci>x</ci></bvar>
    <ci>expression-in-x</ci>
  </bind>
</apply>
```

where *<ci>expression-in-x</ci>* is an arbitrary expression involving the bound variable(s), and the choice of symbol, null depends on the `type` attribute of the `tendsto` element as described above.

4.4.6.4 Tends To `<tendsto/>`

Class	binary-reln
Attributes	CommonAtt, DefEncAtt, type?
`type` Attribute Values	string
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;<!--RIGHTWARDS ARROW--></mo>
 <msup><msup><mi>a</mi><mn>2</mn></msup><mo>+</mo></msup>
</mrow>
```

$$x^2 \longrightarrow {a^2}^+$$

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;<!--RIGHTWARDS ARROW--></mo>
<mfenced><mtable>
  <mtr><mtd>
    <mi>f</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></mo><mfenced><mi>x</mi>
                                                    <mi>y</mi></mfenced>
  </mtd></mtr>
  <mtr><mtd>
    <mi>g</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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 encoding="MathML-Content">
  <tendsto/>
 </annotation-xml>
</semantics>
```

4.4.7 Elementary classical functions

4.4.7.1 Common trigonometric functions `<sin/>`, `<cos/>`, `<tan/>`, `<sec/>`, `<csc/>`, `<cot/>`

Class	unary-elementary
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	sin

These operator elements denote the standard trigonometric functions. Since their standard interpretations are widely known, they are discussed as a group.

Content MathML

```
<apply><sin/><ci>x</ci></apply>
```

Sample Presentation

```
<mrow><mi>sin</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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;<!--FUNCTION APPLICATION--></mo>
 <mrow>
  <mo>(</mo>
  <mrow><mi>cos</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></mo><mi>x</mi></mrow>
  <mo>+</mo>
  <msup><mi>x</mi><mn>3</mn></msup>
  <mo>)</mo>
 </mrow>
</mrow>
```

$$\sin(\cos x + x^3)$$

4.4.7.2 Common inverses of trigonometric functions `<arcsin/>`, `<arccos/>`, `<arctan/>`, `<arcsec/>`, `<arccsc/>`, `<arccot/>`

Class	unary-elementary
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	arcsin

These operator elements denote the inverses of standard trigonometric functions. Differing definitions are in use so for maximum interoperability applications evaluating such expressions should follow the definitions in [Abramowitz1977].

Content MathML

```
<apply><arcsin/><ci>x</ci></apply>
```

Sample Presentations

```
<mrow>
 <mi>arcsin</mi>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mi>x</mi>
</mrow>
```

$$\arcsin x$$

```
<mrow>
 <msup><mi>sin</mi><mrow><mo>-</mo><mn>1</mn></mrow></msup>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mi>x</mi>
</mrow>
```

$$\sin^{-1} x$$

4.4.7.3 Common hyperbolic functions `<sinh/>`, `<cosh/>`, `<tanh/>`, `<sech/>`, `<csch/>`, `<coth/>`

Class	unary-elementary
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	sinh

These operator elements denote the standard hyperbolic functions. Since their standard interpretations are widely known, they are discussed as a group.

Content MathML

```
<apply><sinh/><ci>x</ci></apply>
```

Sample Presentation

```
<mrow><mi>sinh</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></mo><mi>x</mi></mrow>
```

$$\sinh x$$

4.4.7.4 Common inverses of hyperbolic functions <arcsinh/>, <arccosh/>, <arctanh/>, <arcsech/>, <arccsch/>, <arccoth/>

Class	unary-elementary
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	arcsinh

These operator elements denote the inverses of standard hyperbolic functions. Differing definitions are in use so for maximum interoperability applications evaluating such expressions should follow the definitions in [Abramowitz1977].

Content MathML

```
<apply><arcsinh/><ci>x</ci></apply>
```

Sample Presentations

```
<mrow>
 <mi>arcsinh</mi>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mi>x</mi>
</mrow>
```

$$\operatorname{arcsinh} x$$

```
<mrow>
 <msup><mi>sinh</mi><mrow><mo>-</mo><mn>1</mn></mrow></msup>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mi>x</mi>
</mrow>
```

$$\sinh^{-1} x$$

4.4.7.5 Exponential <exp/>

Class	unary-arith
Attributes	CommonAtt, DefEncAtt
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

4.4.7.6 Natural Logarithm `<ln/>`

Class	unary-functional
Attributes	CommonAtt, DefEncAtt
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;<!--FUNCTION APPLICATION--></mo><mi>a</mi></mrow>
```

$\ln a$

4.4.7.7 Logarithm `<log/>`, `<logbase>`

Class	unary-functional
Attributes	CommonAtt, DefEncAtt
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;<!--FUNCTION APPLICATION-->
                                                      </mo><mi>x</mi></mrow>
```

$\log_3 x$

Content MathML

```
<apply><log/><ci>x</ci></apply>
```

Sample Presentation

```
<mrow><mi>log</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></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 its 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>
    <cn>2</cn>
    <ci>x</ci>
  </apply>
  <apply>
    <csymbol cd="transc1">log</csymbol>
    <cn>10</cn>
    <ci>y</ci>
  </apply>
</apply>
```

4.4.8 Statistics

4.4.8.1 Mean `<mean/>`

Class	nary-stats
Attributes	CommonAtt, DefEncAtt
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.

Content MathML

```
<apply><mean/>
  <cn>3</cn><cn>4</cn><cn>3</cn><cn>7</cn><cn>4</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mo>&#x27E8;<!--MATHEMATICAL LEFT ANGLE BRACKET--></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>&#x27E9;<!--MATHEMATICAL RIGHT ANGLE BRACKET--></mo>
</mrow>
```

$$\langle 3,4,3,7,4 \rangle$$

Content MathML

```
<apply><mean/><ci>X</ci></apply>
```

Sample Presentation

```
<mrow><mo>&#x27E8;<!--MATHEMATICAL LEFT ANGLE BRACKET--></mo><mi>X</mi>
                       <mo>&#x27E9;<!--MATHEMATICAL RIGHT ANGLE BRACKET--></mo></mrow>
```

$$\langle X \rangle$$

```
<mover><mi>X</mi><mo>&#xaf;<!--MACRON--></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, DefEncAtt
Content	Empty
Qualifiers	BvarQ, DomainQ
OM Symbols	sdev, sdev

The `sdev` element is used to denote the standard deviation function for a data set or random variable. Standard deviation is a statistical measure of dispersion given by the square root of the variance.

Content MathML

```
<apply><sdev/>
  <cn>3</cn><cn>4</cn><cn>2</cn><cn>2</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <mo>&#x3c3;<!--GREEK SMALL LETTER SIGMA--></mo>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mfenced><mn>3</mn><mn>4</mn><mn>2</mn><mn>2</mn></mfenced>
</mrow>
```

$$\sigma\left(3,4,2,2\right)$$

Content MathML

```
<apply><sdev/>
  <ci type="discrete_random_variable">X</ci>
</apply>
```

Sample Presentation

```
<mrow><mo>&#x3c3;<!--GREEK SMALL LETTER SIGMA--></mo><mo>&#x2061;<!--FUNCTION APPLICATION-->
                                  </mo><mfenced><mi>X</mi></mfenced></mrow>
```

$$\sigma\left(X\right)$$

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, DefEncAtt
Content	Empty
Qualifiers	BvarQ, DomainQ
OM Symbols	variance, variance

The `variance` element represents the variance of a data set or random variable. Variance is a statistical measure of dispersion, averaging the squares of the deviations of possible values from their mean.

Content MathML

```
<apply><variance/>
  <cn>3</cn><cn>4</cn><cn>2</cn><cn>2</cn>
</apply>
```

Sample Presentation

```
<mrow>
 <msup>
  <mo>&#x3c3;<!--GREEK SMALL LETTER SIGMA--></mo>
  <mn>2</mn>
 </msup>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mfenced><mn>3</mn><mn>4</mn><mn>2</mn><mn>2</mn></mfenced>
</mrow>
```

$$\sigma^2\left(3, 4, 2, 2\right)$$

Content MathML

```
<apply><variance/>
  <ci type="discrete_random_variable"> X</ci>
</apply>
```

Sample Presentation

```
<mrow>
 <msup><mo>&#x3c3;<!--GREEK SMALL LETTER SIGMA--></mo><mn>2</mn></msup>
 <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
 <mfenced><mi>X</mi></mfenced>
</mrow>
```

$$\sigma^2\left(X\right)$$

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, DefEncAtt
Content	Empty
Qualifiers	BvarQ, DomainQ
OM Symbols	median

The `median` element represents an operator returning the median of its arguments. The median is a number separating the lower half of the sample values from the upper half.

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;<!--FUNCTION APPLICATION--></mo>
 <mfenced><mn>3</mn><mn>4</mn><mn>2</mn><mn>2</mn></mfenced>
</mrow>
```

$$\mathrm{median}(3, 4, 2, 2)$$

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, DefEncAtt
Content	Empty
Qualifiers	BvarQ, DomainQ
OM Symbols	mode

The `mode` element is used to denote the mode of its arguments. The mode 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;<!--FUNCTION APPLICATION--></mo>
 <mfenced><mn>3</mn><mn>4</mn><mn>2</mn><mn>2</mn></mfenced>
</mrow>
```

$$\mathrm{mode}(3,4,2,2)$$

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, DefEncAtt
Content	Empty
Qualifiers	degree, momentabout
OM Symbols	moment, moment

The `moment` element is used to denote the ith 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>&#x27E8;<!--MATHEMATICAL LEFT ANGLE BRACKET--></mo>
  <msup>
   <mfenced><mn>6</mn><mn>4</mn><mn>2</mn><mn>2</mn><mn>5</mn></mfenced>
   <mn>3</mn>
  </msup>
  <mo>&#x27E9;<!--MATHEMATICAL RIGHT ANGLE BRACKET--></mo>
 </mrow>
 <mi>mean</mi>
</msub>
```

$$\left\langle (6,4,2,2,5)^{3} \right\rangle_{\mathrm{mean}}$$

Content MathML

```
<apply><moment/>
  <degree><cn>3</cn></degree>
  <momentabout><ci>p</ci></momentabout>
  <ci>X</ci>
</apply>
```

Sample Presentation

```
<msub>
 <mrow>
  <mo>&#x27E8;<!--MATHEMATICAL LEFT ANGLE BRACKET--></mo><msup><mi>X</mi><mn>3</mn>
                          </msup><mo>&#x27E9;<!--MATHEMATICAL RIGHT ANGLE BRACKET--></mo>
 </mrow>
 <mi>p</mi>
</msub>
```

$$\langle X^3 \rangle_p$$

Mapping to Strict Markup

When rewriting to Strict Markup, the moment symbol from the s_data1 content dictionary is used when the `moment` element is applied to an explicit list of arguments. When it is applied to a distribution, then the moment symbol from the s_dist1 content dictionary should be used. Both operators take the degree as the first argument, the point as the second, followed by the data set or random variable respectively.

```
<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, DefEncAtt
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.

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 as a matrix consisting of a single row.

The components of a `vector` may be given explicitly as child elements, or specified by rule as described in Section 4.3.1.1 Container Markup for Constructor Symbols.

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>
```

$$(x+y, 3, 7)$$

4.4.9.2 Matrix `<matrix>`

Class	nary-constructor
Attributes	CommonAtt, DefEncAtt
Qualifiers	BvarQ, DomainQ
Content	ContExp*
OM Symbol	matrix

A matrix is regarded as made up of matrix rows, each of which can be thought of as a special type of vector.

Note that the behavior of the `matrix` and `matrixrow` elements is substantially different from the `mtable` and `mtr` presentation elements.

The `matrix` element is a *constructor* element, so the entries may be given explicitly as child elements, or specified by rule as described in Section 4.3.1.1 Container Markup for Constructor Symbols. In the latter case, the entries are specified 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.

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;<!--ELEMENT OF--></mo>
   <mfenced open="[" close="]"><mi>1</mi><mi>5</mi></mfenced>
  </mrow>
  <mo>&#x2227;<!--LOGICAL AND--></mo>
  <mrow>
   <mi>j</mi>
   <mo>&#x2208;<!--ELEMENT OF--></mo>
   <mfenced open="[" close="]"><mi>5</mi><mi>9</mi></mfenced>
  </mrow>
 </mrow>
 <mo>]</mo>
</mrow>
```

$$\left[m_{i,j} \middle| m_{i,j} = i^{j} ; i \in [1,5] \wedge j \in [5,9]\right]$$

4.4.9.3 Matrix row `<matrixrow>`

Class	nary-constructor
Attributes	CommonAtt, DefEncAtt
Qualifiers	BvarQ, DomainQ
Content	ContExp*
OM Symbol	matrixrow

This element is an n-ary constructor used to represent rows of matrices.

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, DefEncAtt
Content	Empty
OM Symbols	determinant

This element is used for the unary function which returns the determinant of its argument, which should be a square matrix.

Content MathML

```
<apply><determinant/>
  <ci type="matrix">A</ci>
</apply>
```

Sample Presentation

```
<mrow><mi>det</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></mo><mi>A</mi></mrow>
```

$\det A$

4.4.9.5 Transpose `<transpose/>`

Class	unary-linalg
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	transpose

This element represents a unary function that signifies 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, DefEncAtt
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; so the resulting list is of matrix rows given entry by entry. That is, for a matrix $\left(a_{i,j}\right)$, where the indices denote row and column, respectively, the ordering would be $a_{1,1}, a_{1,2}, \cdots a_{2,1}, a_{2,2} \cdots$ etc.

When two arguments are given, and the first is a vector or list, the second argument specifies the index of an entry in the list or vector. If the first argument is a matrix then the second argument specifies the index of a matrix row.

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 indices of the selected element.

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>
  <matrix>
    <matrixrow><cn>1</cn><cn>2</cn></matrixrow>
  </matrix>
</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>
 <mrow>
  <mo>(</mo>
  <mtable><mtr><mtd><mn>1</mn></mtd><mtd><mn>2</mn></mtd></mtr></mtable>
  <mo>)</mo>
 </mrow>
</mrow>
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}_1 = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

4.4.9.7 Vector product `<vectorproduct/>`

Class	binary-linalg
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	vectorproduct

This element represents the vector product. It takes two three-dimensional vector arguments and represents as value 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;<!--GREEK SMALL LETTER THETA--></ci></apply>
    <ci type="vector"> N </ci>
  </apply>
</apply>
```

Sample Presentation

```
<mrow>
 <mrow><mi>A</mi><mo>&#xd7;<!--MULTIPLICATION SIGN--></mo><mi>B</mi></mrow>
 <mo>=</mo>
 <mrow>
```

```
  <mi>a</mi>
  <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
  <mi>b</mi>
  <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
  <mrow><mi>sin</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></mo><mi>&#x3b8;<!--GREEK
                                                        SMALL LETTER THETA--></mi></mrow>
  <mo>&#x2062;<!--INVISIBLE TIMES--></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, DefEncAtt
Content	Empty
OM Symbols	scalarproduct

This element 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;<!--GREEK SMALL LETTER THETA--></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;<!--INVISIBLE TIMES--></mo>
  <mi>b</mi>
  <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
  <mrow><mi>cos</mi><mo>&#x2061;<!--FUNCTION APPLICATION--></mo><mi>&#x3b8;<!--GREEK
                                                        SMALL LETTER THETA--></mi></mrow>
 </mrow>
</mrow>
```

$$A\,.\,B = ab\cos\theta$$

4.4.9.9 Outer product `<outerproduct/>`

Class	binary-linalg
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	outerproduct

This element represents the outer product function. It takes two vector arguments and returns as value 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;<!--CIRCLED TIMES--></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, DefEncAtt
Content	Empty
OM Symbols	Z

This element 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;<!--ELEMENT OF--></mo><mi
                                  mathvariant="double-struck">Z</mi></mrow>
```

$$42 \in \mathbb{Z}$$

4.4.10.2 reals `<reals/>`

Class	constant-set
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	R

This element 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;<!--ELEMENT OF--></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, DefEncAtt
Content	Empty
OM Symbols	Q

This element 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;<!--ELEMENT OF--></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, DefEncAtt
Content	Empty
OM Symbols	N

This element 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;<!--ELEMENT OF--></mo><mi
                                        mathvariant="double-struck">N</mi>
</mrow>
```

$1729 \in \mathbb{N}$

4.4.10.5 complexes <complexes/>

Class	constant-set
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	C

This element 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;<!--INVISIBLE TIMES--></mo>
                                                         <mi>i</mi></mrow>
 <mo>&#x2208;<!--ELEMENT OF--></mo>
 <mi mathvariant="double-struck">C</mi>
</mrow>
```

$17 + 29i \in \mathbb{C}$

4.4.10.6 primes `<primes/>`

Class	constant-set
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	P

This element 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;<!--ELEMENT OF--></mo><mi
                                  mathvariant="double-struck">P</mi></mrow>
```

$$17 \in \mathbb{P}$$

4.4.10.7 Exponential e `<exponentiale/>`

Class	constant-arith
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	e

This element 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;<!--FUNCTION APPLICATION--></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, DefEncAtt
Content	Empty
OM Symbols	i

This element 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, DefEncAtt
Content	Empty
OM Symbols	NaN

This element represents the notion of not-a-number, i.e. the result of an ill-posed floating computation. See [IEEE754].

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>
```

$$0/0 = \mathrm{NaN}$$

4.4.10.10 True `<true/>`

Class	constant-arith
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	true

This element 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;<!--LOGICAL OR--></mo><mi>P</mi></mrow>
 <mo>=</mo>
 <mi>true</mi>
</mrow>
```

$$\mathrm{true} \vee P = \mathrm{true}$$

4.4.10.11 False `<false/>`

Class	constant-arith
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	false

This element 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;<!--LOGICAL AND--></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, DefEncAtt
Content	Empty
OM Symbols	emptyset, emptyset

This element is used to represent the empty set, that is the set which contains no members.

Content MathML

```
<apply><neq/>
  <integers/>
  <emptyset/>
</apply>
```

Sample Presentation

```
<mrow>
 <mi mathvariant="double-struck">Z</mi><mo>&#x2260;<!--NOT EQUAL TO--></mo>
                                              <mi>&#x2205;<!--EMPTY SET--></mi>
</mrow>
```

$\mathbb{Z} \neq \varnothing$

Mapping to Strict Markup

In some situations, it may be clear from context that `emptyset` corresponds to the emptyset However, as there is no method other than annotation for an author to explicitly indicate this, it is always acceptable to translate to the emptyset symbol from the set1 CD.

4.4.10.13 pi `<pi/>`

Class	constant-arith
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	pi

This element represents pi, approximately 3.142, which is 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;<!--GREEK SMALL LETTER PI--></mi>
 <mo>&#x2243;<!--ASYMPTOTICALLY EQUAL TO--></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, DefEncAtt
Content	Empty
OM Symbols	gamma

This element denotes the gamma constant, approximately 0.5772.

Content MathML

```
<apply><approx/>
  <eulergamma/>
  <cn>0.5772156649</cn>
</apply>
```

Sample Presentation

```
<mrow><mi>&#x3b3;<!--GREEK SMALL LETTER GAMMA--></mi><mo>&#x2243;<!--ASYMPTOTICALLY EQUAL TO--></mo><mn>0.5772156649</mn></mrow>
```

$$\gamma \simeq 0.5772156649$$

4.4.10.15 infinity `<infinity/>`

Class	constant-arith
Attributes	CommonAtt, DefEncAtt
Content	Empty
OM Symbols	infinity

This element represents the notion of infinity.

Content MathML

```
<infinity/>
```

Sample Presentation

```
<mi>&#x221e;<!--INFINITY--></mi>
```

$$\infty$$

4.5 Deprecated Content Elements

4.5.1 Declare `<declare>`

Attributes	CommonAtt, type, scope, occurrence, definitionURL, encoding
`type` Attribute	defines the MathML element type of the identifier declared.
`scope` Attribute	defines the scope of application of the declaration.
`nargs` Attribute	number of arguments for function declarations.
`occurrence` Attribute values	"prefix" | "infix" | "function-model"
`definitionURL` Attribute	URI pointing to detailed semantics of the function.
`encoding` Attribute	syntax of the detailed semantics of the function.
Content	ContExp, ContExp?

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 Structure Sharing `<share>` for details).

4.5.2 Relation `<reln>`

Content	ContExp*

MathML1 provided the `reln` element to construct an equation or relation. This usage was deprecated in MathML 2.0 in favor of the more generally usable `apply`.

4.5.3 Relation `<fn>`

Content	ContExp

MathML1 provided the `fn` element to extend the collection of known mathematical functions. This usage was deprecated in MathML 2.0 in favor of the more generally applicable `csymbol`.

4.6 The Strict Content MathML Transformation

MathML 3 assigns semantics to content markup by defining a mapping to Strict Content MathML. Strict MathML, in turn, is in one-to-one correspondence with OpenMath, and the subset of OpenMath expressions obtained from content MathML expressions in this fashion all have well-defined semantics via the standard OpenMath Content Dictionary set. Consequently, the mapping of arbitrary content MathML expressions to equivalent Strict Content MathML plays a key role in underpinning the meaning of content MathML.

The mapping of arbitrary content MathML into Strict content MathML is defined algorithmically. The algorithm is described below as a collection of rewrite rules applying to specific non-Strict constructions. The individual rewrite transformations have been described in detail in context above. The goal of this section is to outline the complete algorithm in one place.

The algorithm is a sequence of nine steps. Each step is applied repeatedly to rewrite the input until no further application is possible. Note that in many programming languages, such as XSLT, the natural implementation is as a recursive algorithm, rather than the multi-pass implementation suggested by the description below. The translation to XSL is straightforward and produces the same eventual Strict Content MathML. However, because the overall structure of the multi-pass algorithm is clearer, that is the formulation given here.

To transform an arbitrary content MathML expression into Strict Content MathML, apply each of the following rules in turn to the input expression until all instances of the target constructs have been eliminated:

1. *Rewrite non-strict `bind` and elminate deprecated elements*: 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 later rules will change the `apply` elements introduced in this step back to `bind` elements. Also in this step, deprecated `reln` elements are rewritten to `apply`, and `fn` elements are replaced by the child expressions they enclose.
2. *Apply special case rules for idiomatic uses of qualifiers*:
    a. Rewrite derivatives with rules Rewrite: diff, Rewrite: nthdiff, and Rewrite: partialdiffdegree to explicate the binding status of the variables involved.
    b. Rewrite integrals with the rules Rewrite: int, Rewrite: defint and Rewrite: defint limits to disambiguate the status of bound and free variables and of the orientation of the range of integration if it is given as a `lowlimit`/`uplimit` pair.
    c. Rewrite limits as described in Rewrite: tendsto and Rewrite: limits condition.
    d. Rewrite sums and products as described in Section 4.4.6.1 Sum `<sum/>` and Section 4.4.6.2 Product `<product/>`.
    e. Rewrite roots as described in Section 4.4.2.11 Root `<root/>`.
    f. Rewrite logarithms as described in Section 4.4.7.7 Logarithm `<log/>`, `<logbase>` .
    g. Rewrite moments as described in Section 4.4.8.6 Moment `<moment/>`, `<momentabout>`.
3. *Rewrite Qualifiers to `domainofapplication`*: These rules rewrite all `apply` constructions using `bvar` and qualifiers to those using only the general `domainofapplication` qualifier.
    a. *Intervals*: Rewrite qualifiers given as `interval` and `lowlimit`/`uplimit` to intervals of integers via Rewrite: interval qualifier.
    b. *Multiple `condition`s*: Rewrite multiple `condition` qualifiers to a single one by taking their conjunction. The resulting compound `condition` is then rewritten to `domainofapplication` according to rule Rewrite: condition.
    c. *Multiple `domainofapplication`s*: Rewrite multiple `domainofapplication` qualifiers to a single one by taking the intersection of the specified domains.
4. *Normalize Container Markup*:
    a. Rewrite sets and lists by the rule Rewrite: n-ary setlist domainofapplication.
    b. Rewrite interval, vectors, matrices, and matrix rows as described in Section 4.4.1.1 Interval `<interval>`, Section 4.4.9.1 Vector `<vector>`, Section 4.4.9.2 Matrix `<matrix>` and Section 4.4.9.3 Matrix row `<matrixrow>`. Note any qualifiers will have been rewritten to `domainofapplication` and will be further rewritten in Step 6.
    c. Rewrite lambda expressions by the rules Rewrite: lambda and Rewrite: lambda domainofapplication
    d. Rewrite piecewise functions as described in Section 4.4.1.9 Piecewise declaration `<piecewise>`, `<piece>`, `<otherwise>`.
5. *Apply Special Case Rules for Operators using `domainofapplication` Qualifiers*: This step deals with the special cases for the operators introduced in Section 4.4 Content MathML for Specific Operators and Constants. There are different classes of special cases to be taken into account:
    a. Rewrite `min`, `max`, `mean` and similar n-ary/unary operators by the rules Rewrite: n-ary unary set, Rewrite: n-ary unary domainofapplication and Rewrite: n-ary unary single.
    b. Rewrite the quantifiers `forall` and `exists` used with `domainofapplication` to expressions using implication and conjunction by the rule Rewrite: quantifier.
    c. Rewrite integrals used with a `domainofapplication` element (with or without a `bvar`) according to the rules Rewrite: int and Rewrite: defint.

   d. Rewrite sums and products used with a `domainofapplication` element (with or without a `bvar`) as described in Section 4.4.6.1 Sum `<sum/>` and Section 4.4.6.2 Product `<product/>`.

6. *Eliminate `domainofapplication`*: At this stage, any `apply` has at most one `domainofapplication` child and special cases have been addressed. 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 the predicate_on_list symbol 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 an `apply` with bound variables via the Rewrite: apply bvar domainofapplication rule.

7. *Rewrite non-strict token elements*:

   a. Rewrite numbers represented as `cn` elements where the `type` attribute is one of "e-notation", "rational", "complex-cartesian", "complex-polar", "constant" as strict `cn` via rules Rewrite: cn sep, Rewrite: cn based_integer and Rewrite: cn constant.

   b. Rewrite any `ci`, `csymbol` or `cn` containing presentation MathML to `semantics` elements with rules Rewrite: cn presentation mathml and Rewrite: ci presentation mathml and the analogous rule for `csymbol`.

8. *Rewrite operators*: Rewrite any remaining operator defined in Section 4.4 Content MathML for Specific Operators and Constants to a `csymbol` referencing the symbol identified in the syntax table by the rule Rewrite: element. As noted in the descriptions of each operator element, some require special case rules to determine the proper choice of symbol. Some cases of particular note are:

   a. The order of the arguments for the `selector` operator must be rewritten, and the symbol depends on the type of the arguments.

   b. The choice of symbol for the `minus` operator depends on the number of the arguments.

   c. The choice of symbol for some set operators depends on the values of the `type` of the arguments.

   d. The choice of symbol for some statistical operators depends on the values of the types of the arguments.

9. *Rewrite non-strict attributes*:

   a. *Rewrite the `type` attribute*: At this point, all elements that accept the `type`, other than `ci` and `csymbol`, should have been rewritten into Strict Content Markup equivalents without `type` attributes, where type information is reflected in the choice of operator symbol. Now rewrite remaining `ci` and `csymbol` elements with a `type` attribute to a strict expression with `semantics` according to rules Rewrite: ci type annotation and Rewrite: csymbol type annotation.

   b. *Rewrite `definitionURL` and `encoding` attributes*: 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 Non-Strict uses of `<csymbol>` for details), then rewrite to reference the content dictionary by the `cd` attribute instead.

   c. *Rewrite attributes*: Rewrite any element with attributes that are not allowed in strict markup to a `semantics` construction with the element without these attributes as the first child and the attributes in `annotation` elements by rule Rewrite: attributes.

5 Mixing Markup Languages for Mathematical Expressions

MathML markup can be combined with other markup languages, and these mixing constructions are realized by the semantic annotation elements. The semantic annotation elements provide an important tool for making associations between alternate representations of an expression, and for associating semantic properties 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 Annotation Framework

An important concern of MathML is to represent associations between presentation and content markup forms for an expression. Representing associations between MathML expressions and data of other kinds is also important in many contexts. For this reason, MathML provides a general framework for annotation. A MathML expression may be decorated with a sequence of pairs made up of a symbol that indicates the kind of annotation, known as the *annotation key*, and associated data, known as the *annotation value*.

5.1.1 Annotation elements

The `semantics`, `annotation`, and `annotation-xml` elements are used together to represent annotations in MathML. The `semantics` element provides the container for a expression and its annotations. The `annotation` element is the container for text annotations, and the `annotation-xml` element is used for structured annotations. The `semantics` element contains the expression being annotated as its first child, followed by a sequence of zero or more `annotation` and/or `annotation-xml` elements.

```
<semantics>
  <mrow>
    <mrow>
      <mi>sin</mi>
      <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
      <mfenced><mi>x</mi></mfenced>
    </mrow>
    <mo>+</mo>
    <mn>5</mn>
  </mrow>
  <annotation encoding="application/x-tex">
    \sin x + 5
  </annotation>
  <annotation-xml encoding="application/openmath+xml">
    <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>
```

Note that this example makes use of the namespace extensibility that is only available in the XML syntax of MathML. If this example is included in an HTML document then it would be considered *invalid* and the OpenMath elements would be parsed as elements un the MathML namespace. See Section 5.2.3.3 Using `annotation-xml` in HTML documents for details.

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 keys

An *annotation key* specifies the relationship between an expression and an annotation. Many kinds of relationships are possible. Examples include alternate representations, specification or clarification of semantics, type information, rendering hints, and private data intended for specific processors. The annotation key is the primary means by which a processor determines whether or not to process an annotation.

The logical relationship between an expression and an annotation can have a significant impact on the proper processing of the expression. For example, a particular annotation form, called *semantic attributions*, cannot be ignored without altering the meaning of the annotated expression, at least in some processing contexts. On the other hand, alternate representations do not alter the meaning of an expression, but may alter the presentation of the expression as they are frequently used to provide rendering hints. Still other annotations carry private data or metadata that are useful in a specific context, but do not alter either the semantics or the presentation of the expression.

In MathML 3, annotation keys are defined as symbols in Content Dictionaries, and are specified using of the `cd` and `name` attributes on the `annotation` and `annotation-xml` elements. For backward compatibility with MathML 2, an annotation key may also be referenced using the `definitionURL` attribute as an alternative to the `cd` and `name` attributes. Further details on referencing symbols in Content Dictionaries are discussed in Section 4.2.3 Content Symbols `<csymbol>`. The symbol definition in a Content Dictionary for an annotation key may have a `role` property. Two particular roles are relevant for annotations: a role of "attribution" identifies a generic annotation that can be ignored without altering the meaning of the annotated term, and a role of "semantic-attribution" indicates that the annotation is a semantic annotation, that is, the annotation cannot be ignored without potentially altering the meaning of the expression.

MathML 3 provides two predefined annotation keys for the most common kinds of annotations: alternate-representation and contentequiv defined in the mathmlkeys content dictionary. The alternate-representation annotation key specifies that the annotation value provides an alternate representation for an expression in some other markup language, and the contentequiv annotation key specifies that the annotation value provides a semantically equivalent alternative for the annotated expression. Further details about the use of these keys is given in the sections below.

The default annotation key is alternate-representation when no annotation key is explicitly specified on an `annotation` or `annotation-xml` element.

Typically, annotation keys specify only the logical nature of the relationship between an expression and an annotation. The data format for an annotation is indicated with the `encoding` attribute. In MathML 2, the `encoding` attribute was the primary information that a processor could use to determine whether or not it could understand an annotation. For backward compatibility, processors are encouraged to examine both the annotation key and `encoding` attribute. In particular, MathML 2 specified the predefined encoding values `MathML`, `MathML-Content`, and `MathML-Presentation`. The `MathML` encoding value is used to indicate an `annotation-xml` element contains a MathML expression. The use of the other values is more specific, as discussed in following sections.

While the predefined alternate-representation and contentequiv keys cover many common use cases, user communities are encouraged to define and standardize additional content dictionaries as necessary. Annotation keys in user-defined, public Content Dictionaries are preferred over private encoding attribute value conventions, since content dictionaries are more expressive, more open and more maintainable than private encoding values. However, for backward compatibility with MathML 2, the encoding attribute may also be used.

5.1.3 Alternate representations

Alternate representation annotations are most often used to provide renderings for an expression, or to provide an equivalent representation in another markup language. In general, alternate representation annotations do not alter the meaning of the annotated expression, but may alter its presentation.

A particularly important case is the use of a presentation MathML expression to indicate a preferred rendering for a content MathML expression. This case may be represented by labeling the annotation with the `application/mathml-presentation+xml` value for the `encoding` attribute. For backward compatibility with MathML 2.0, this case can also be represented with the equivalent `MathML-Presentation` value for the `encoding` attribute. Note that when a presentation MathML annotation is present in a `semantics` element, it may be used as the default rendering of the `semantics` element, instead of the default rendering of the first child.

In the example below, the `semantics` element binds together various alternate representations for a content MathML expression. The presentation MathML annotation may be used as the default rendering, while the other annotations give representations in other markup languages. Since no attribution keys are explicitly specified, the default annotation key alternate-representation applies to each of the annotations.

```
<semantics>
  <apply>
    <plus/>
    <apply><sin/><ci>x</ci></apply>
    <cn>5</cn>
  </apply>
  <annotation-xml encoding="MathML-Presentation">
    <mrow>
      <mrow>
        <mi>sin</mi>
        <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
        <mfenced open="(" close=")"><mi>x</mi></mfenced>
      </mrow>
      <mo>+</mo>
      <mn>5</mn>
    </mrow>
  </annotation-xml>
  <annotation encoding="application/x-maple">
    sin(x) + 5
  </annotation>
  <annotation encoding="application/vnd.wolfram.mathematica">
    Sin[x] + 5
  </annotation>
  <annotation encoding="application/x-tex">
    \sin x + 5
  </annotation>
  <annotation-xml encoding="application/openmath+xml">
    <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>
```

Note that this example makes use of the namespace extensibility that is only available in the XML syntax of MathML. If this example is included in an HTML document then it would be considered *invalid* and the OpenMath elements would be parsed as elements un the MathML namespace. See Section 5.2.3.3 Using `annotation-xml` in HTML documents for details.

5.1.4 Content equivalents

Content equivalent annotations provide additional computational information about an expression. Annotations with the contentequiv key cannot be ignored without potentially changing the behavior of an expression.

An important case arises when a content MathML annotation is used to disambiguate the meaning of a presentation MathML expression. This case may be represented by labeling the annotation with the `application/mathml-content+xml` value for the `encoding` attribute. In MathML 2, this type of annotation was represented with the equivalent `MathML-Content` value for the `encoding` attribute, so processors are urged to support this usage for backward compatibility. A content MathML annotation, whether in MathML 2 or 3, may be used for other purposes as well, such as for other kinds of semantic assertions. Consequently, in MathML 3, the contentequiv annotation key should be used to make an explicit assertion that the annotation provides a definitive content markup equivalent for an expression.

In the example below, an ambiguous presentation MathML expression is annotated with a `MathML-Content` annotation clarifying its precise meaning.

```
<semantics>
  <mrow>
    <mrow>
      <mi>a</mi>
      <mfenced open="(" close=")">
        <mrow><mi>x</mi><mo>+</mo><mn>5</mn></mrow>
      </mfenced>
    </mrow>
  </mrow>
  <annotation-xml cd="mathmlkeys" name="contentequiv"
                                                  encoding="MathML-Content">
    <apply>
      <ci>a</ci>
      <apply><plus/><ci>x</ci><cn>5</cn></apply>
    </apply>
  </annotation-xml>
</semantics>
```

5.1.5 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 `src` 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" src="333/formula56.png"/>
  <annotation encoding="application/x-maple" src="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.

5.2 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

5.2.1.1 Description

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. Any MathML expression may appear as the first child of the `semantics` element. Subsequent `annotation` and `annotation-xml` children enclose the annotations. An annotation represented in XML is enclosed in an `annotation-xml` element. An annotation represented in character data is enclosed in an `annotation` element.

As noted above, the `semantics` element is considered to be both a presentation element and a content element, since it can act as either, depending on its content. Consequently, all MathML processors should process the `semantics` element, even if they process only presentation markup or only content markup.

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.

```
<semantics>
  <mrow>
    <mrow>
      <mi>sin</mi>
      <mo>&#x2061;<!--FUNCTION APPLICATION--></mo>
      <mfenced><mi>x</mi></mfenced>
    </mrow>
    <mo>+</mo>
    <mn>5</mn>
  </mrow>
  <annotation-xml cd="mathmlkeys" name="contentequiv"
                                                  encoding="MathML-Content">
    <apply>
      <plus/>
      <apply><sin/><ci>x</ci></apply>
      <cn>5</cn>
    </apply>
  </annotation-xml>
  <annotation encoding="application/x-tex">
    \sin x + 5
  </annotation>
</semantics>
```

5.2.1.2 Attributes

Name	values	default
`definitionURL`	*URI*	*none*
	The location of an external source for semantic information	
`encoding`	*string*	*none*
	The encoding of the external semantic information	

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 on the `semantics` element is deprecated in MathML3.

5.2.2 The `<annotation>` element

5.2.2.1 Description

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 `&`, `<` then these characters must be encoded using an entity reference or (in the XML syntax) an XML `CDATA` section.

5.2.2.2 Attributes

Name	values	default
`definitionURL`	*URI*	*none*
	The location of the annotation key symbol	
`encoding`	*string*	*none*
	The encoding of the semantic information in the annotation	
`cd`	*string*	mathmlkeys
	The content dictionary that contains the annotation key symbol	
`name`	*string*	alternate-representation
	The name of the annotation key symbol	
`src`	*URI*	*none*
	The location of an external source for semantic information	

Taken together, the `cd` and `name` attributes specify the annotation key symbol, which identifies the relationship between the annotated element and the annotation, as described in Section 5.1.1 Annotation elements. The `definitionURL` attribute provides an alternate way to reference the annotation key symbol as a single attribute. If none of these attributes are present, the annotation key symbol is the symbol alternate-representation from the mathmlkeys content dictionary.

The `encoding` attribute 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.

The `src` attribute provides a mechanism to attach external entities as annotations on MathML expressions.

```
<annotation encoding="image/png" src="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.

5.2.3 The <annotation-xml> element

5.2.3.1 Description

The annotation-xml element is the container element for a semantic annotation whose representation is structured markup. The annotation-xml element should contain the markup elements, attributes, and character data for the annotation.

5.2.3.2 Attributes

Name	values	default
definitionURL	*URI*	*none*
	The location of the annotation key symbol	
encoding	*string*	*none*
	The encoding of the semantic information in the annotation	
cd	*string*	mathmlkeys
	The content dictionary that contains the annotation key symbol	
name	*string*	alternate-representation
	The name of the annotation key symbol	
src	*URI*	*none*
	The location of an external source for semantic information	

Taken together, the cd and name attributes specify the annotation key symbol, which identifies the relationship between the annotated element and the annotation, as described in Section 5.1.1 Annotation elements. The definitionURL attribute provides an alternate way to reference the annotation key symbol as a single attribute. If none of these attributes are present, the annotation key symbol is the symbol alternate-representation from the mathmlkeys content dictionary.

The encoding attribute 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. In particular, as described above and in Section 6.2.4 Names of MathML Encodings, MathML specifies MathML, MathML-Presentation, and MathML-Content as predefined values for the encoding attribute. Finally, The src attribute 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 encoding="application/openmath+xml">
  <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 MathML is being parsed as XML and 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.

5.2.3.3 Using `annotation-xml` in HTML documents

Note that the Namespace extensibility used in the above examples may not be available if the MathML is not being treated as an XML document. In particular HTML parsers treat `xmlns` attributes as ordinary attributes, so the OpenMath example would be classified as invalid by an HTML validator. The OpenMath elements would still be parsed as children of the `annotation-xml` element, however they would be placed in the *MathML* namespace. The above examples are not rendered in the HTML version of this specification, to ensure that that document is a valid HTML5 document.

The details of the HTML parser handling of `annotation-xml` is specified in [HTML5] and summarized in Section 6.4.3 Mixing MathML and HTML, however the main differences from the behavior of an XML parser that affect MathML annotations are that the HTML parser does not treat `xmlns` attributes, nor `:` in element names as special and has built-in rules determining whether the three "known" namespaces, HTML, SVG or MathML are used.

- If the `annotation-xml` has an `encoding` attribute that is (ignoring case differences) "text/html" or "annotation/xhtml+xml" then the content is parsed as HTML and placed (initially) in the HTML namespace.
- Otherwise it is parsed as *foreign content* and parsed in a more XML-like manner (like MathML itself in HTML) in which `/>` signifies an empty element. Content will be placed in the MathML namespace.

  If any recognised HTML element appears in this foreign content annotation the HTML parser will effectively termnate the math expression, closing all open elements until the `math` element is closed, and then process the nested HTML as if it were not inside the math context. Any following MathML elements will then not render correctly as they are not in a math context, or in the MathML namespace.

These issues mean that the following example is valid whether parsed by an XML or HTML parser:

```
<math>
 <semantics>
  <mi>a</mi>
  <annotation-xml encoding="text/html">
```

```
   <span>xxx</span>
  </annotation-xml>
 </semantics>
 <mo>+</mo>
 <mi>b</mi>
</math>
```

However the if the `encoding` attribute is omitted then the expression is only valid if parsed as XML:

```
<math>
 <semantics>
  <mi>a</mi>
  <annotation-xml>
   <span>xxx</span>
  </annotation-xml>
 </semantics>
 <mo>+</mo>
 <mi>b</mi>
</math>
```

If the above is parsed by an HTML parser it produces a result equivalent to the following invalid input, where the `span` element has caused all MathML elements to be prematurely closed. The remaining MathML elements following the `span` are no longer contained within `<math>` so will be parsed as unknown HTML elements and render incorrectly.

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
 <semantics>
  <mi>a</mi>
  <annotation-xml>
  </annotation-xml>
 </semantics>
</math>
<span xmlns="http://www.w3.org/1999/xhtml">xxx</span>
<mo xmlns="http://www.w3.org/1999/xhtml">+</mo>
<mi xmlns="http://www.w3.org/1999/xhtml">b</mi>
```

Note here that the HTML `span` element has caused all open MathML elements to be prematurely closed, resulting in the following MathML elements being treated as unknown HTML elements as they are no longer descendents of `math`. See Section 6.4.3 Mixing MathML and HTML for more details of the parsing of MathML in HTML.

Any use of elements in other vocabularies (such as the OpenMath examples above) is considered *invalid* in HTML. If validity is not a strict requirement it is possible to use such elements but they will be parsed as elements on the *MathML* namespace. Documents SHOULD NOT use namespace prefixes and element names containing colon (`:`) as the element nodes produced by the HTML parser with have local names containing a colon, which can not be constructed by a namespace aware XML parser. Rather than use such foreign annotations, when using an HTML parser it is better to encode the annotation using the existing vocabulary. For example as shown in Chapter 4 Content Markup OpenMath may be encoded faithfuly as *Strict Content MathML*. Similarly RDF annotations could be encoded using RDFa in `text/html` annotation or (say) N3 notation in `annotation` rather than using RDF/XML encoding in an `annotation-xml` 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 Characters, Entities and Fonts) 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>&#x2062;<!--INVISIBLE TIMES--></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 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">&#x2062;<!--INVISIBLE TIMES--></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.

For parallel markup, all of the `id` attributes referenced by any `xref` attribute should 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="EE">
    <mrow id="EE.1">
      <mo id="EE.1.1">(</mo>
      <mi id="EE.1.2">a</mi>
      <mo id="EE.1.3">+</mo>
      <mi id="EE.1.4">b</mi>
      <mo id="EE.1.5">)</mo>
    </mrow>
    <mo id="EE.2">&#x2062;<!--INVISIBLE TIMES--></mo>
    <mrow id="EE.3">
      <mo id="EE.3.1">(</mo>
      <mi id="EE.3.2">c</mi>
      <mo id="EE.3.3">+</mo>
      <mi id="EE.3.4">d</mi>
      <mo id="EE.3.5">)</mo>
    </mrow>
  </mrow>

  <annotation-xml encoding="MathML-Content">
    <apply xref="EE">
      <and xref="EE.2"/>
      <apply xref="EE.1">
        <xor xref="EE.1.3"/><ci xref="EE.1.2">a</ci><ci xref="EE.1.4">b</ci>
      </apply>
      <apply xref="EE.3">
        <xor xref="EE.3.3"/><ci xref="EE.3.2">c</ci><ci xref="EE.3.4">d</ci>
      </apply>
    </apply>
  </annotation-xml>

  <annotation-xml encoding="application/openmath+xml">
    <om:OMA xmlns:om="http://www.openmath.org/OpenMath" href="EE">
      <om:OMS name="and" cd="logic1" href="EE.2"/>

      <om:OMA href="EE.1">
        <om:OMS name="xor" cd="logic1" href="EE.1.3"/>
        <om:OMV name="a" href="EE.1.2"/>
        <om:OMV name="b" href="EE.1.4"/>
      </om:OMA>

      <om:OMA href="EE.3">
        <om:OMS name="xor" cd="logic1" href="EE.3.3"/>
        <om:OMV name="c" href="EE.3.2"/>
        <om:OMV name="d" href="EE.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. As noted above, the use of namespaces other than MathML, SVG or HTML within `annotation-xml` is not considered valid in the HTML syntax. Use of colons and namespace-prefixed element names should be avoided as the HTML parser will generate nodes with *local* name `om:OMA` (for example), and such nodes can not be constructed by a namespace-aware XML parser.

6 Interactions 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 relate to embedding MathML in [HTML5], 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 markup must be recognized as valid embedded XML content, and not as an error. This issue could be seen primarily as a question of managing namespaces in XML [Namespaces].

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 JavaScript/ECMAScript 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 [MathMLforCSS] that is tailored for use with CSS rendering engines that support CSS 2.1 [CSS21]. This restricted profile of MathML3 does not offer the full expressiveness of MathML3, but it 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 made to ensure that MathML can easily be generated by user-friendly conversion and authoring tools, and that these tools work together in a dependable, platform-independent, and vendor-independent way.

This chapter applies to both content and presentation markup, and describes a particular processing model for the `semantics`, `annotation` and `annotation-xml` elements described in Section 5.1 Annotation Framework.

6.2 Invoking MathML Processors

6.2.1 Recognizing MathML in XML

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 MathML namespace by the use of the MathML namespace URI `http://www.w3.org/1998/Math/MathML`.

The MathML namespace URI is the recommended method to embed MathML within [XHTML] documents. However, some user-agents may require supplementary information to be available to allow them to invoke specific extensions to process the MathML markup.

Markup-language specifications that wish to embed MathML may require special conditions to recognize MathML markup that are independent of this recommendation. The conditions should be similar to those expressed in this recommendation, and the local names of the MathML elements should remain the same as those defined in this recommendation.

6.2.2 Recognizing MathML in HTML

HTML does not allow arbitrary namespaces, but has built in knowledge of the MathML namespace. The `math` element and its descendants will be placed in the `http://www.w3.org/1998/Math/MathML` namespace by the HTML parser, and will appear to applications as if the input had been XHTML with the namespace declared as in the previous section. See Section 6.4.3 Mixing MathML and HTML for detailed rules of the HTML parser's handling of MathML.

6.2.3 Resource Types for MathML Documents

Although rendering MathML expressions often takes 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 by which MathML fragments should be identified.

Outside of those environments where XML namespaces are recognized, media types [RFC2045], [RFC2046] should be used if possible to ensure the invocation of a MathML processor. For those environments where media types are not appropriate, such as clipboard formats on some platforms, the encoding names described in the next section should be used.

6.2.4 Names of MathML Encodings

MathML contains two distinct vocabularies: one for encoding visual presentation, defined in Chapter 3 Presentation Markup, and one for encoding computational structure, defined in Chapter 4 Content Markup. Some MathML applications may import and export only one of these two vocabularies, while others may produce and consume each in a different way, and still others may process both without any distinction between the two. The following encoding names may be used to distinguish between content and presentation MathML markup when needed.

- *MathML-Presentation*: The instance contains presentation MathML markup only.
    - Media Type: `application/mathml-presentation+xml`
    - Windows Clipboard Flavor: `MathML Presentation`
    - Universal Type Identifier: `public.mathml.presentation`
- *MathML-Content*: The instance contains content MathML markup only.
    - Media Type: `application/mathml-content+xml`
    - Windows Clipboard Flavor: `MathML Content`
    - Universal Type Identifier: `public.mathml.content`
- *MathML* (generic): The instance may contain presentation MathML markup, content MathML markup, or a mixture of the two.
    - File name extension: `.mml`
    - Media Type: `application/mathml+xml`
    - Windows Clipboard Flavor: `MathML`
    - Universal Type Identifier: `public.mathml`

See Appendix B Media Types Registrations for more details about each of these encoding names.

MathML 2 specified the predefined encoding values `MathML`, `MathML-Content`, and `MathML-Presentation` for the `encoding` attribute on the `annotation-xml` element. These values may be used as an alternative to the media type for backward compatibility. See Section 5.1.3 Alternate representations and Section 5.1.4

Content equivalents for details. Moreover, MathML 1.0 suggested the media-type `text/mathml`, which has been superseded 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 are often stored in files or exchanged over the HTTP protocol. This section provides recommended ways to process MathML during these transfers.

The transfers of MathML fragments described in this section occur between the contexts of two applications by making the MathML data available in several flavors, often called *media types*, *clipboard formats*, or *data flavors*. These flavors are typically ordered by preference by the producing application, and are typically examined in preference order by the consuming application. The copy-and-paste paradigm allows an application to *place* content in a central *clipboard*, with one data stream per *clipboard format*; a consuming application negotiates by choosing to read the data of the format it prefers. The drag-and-drop paradigm allows an application to *offer* content by declaring the available formats; a potential recipient accepts or rejects a drop based on the list of available formats, and the drop action allows the receiving application to request the delivery of the data in one of the indicated formats. An HTTP GET transfer, as in [HTTP11], allows a client to submit a list of acceptable media types; the server then delivers the data using the one of the indicated media types. An HTTP POST transfer, as in [HTTP11], allows a client to submit data labelled with a media type that is acceptable to the server application.

Current desktop platforms offer copy-and-paste and drag-and-drop transfers using similar architectures, but with varying naming schemes depending on the platform. HTTP transfers are all based on media types. This section specifies what transfer types applications should provide, how they should be named, and how they should handle the special `semantics`, `annotation`, and `annotation-xml` elements.

To summarize the three negotiation mechanisms, the following paragraphs will describe transfer *flavors*, each with a *name* (a character string) and *content* (a stream of binary data), which are *offered*, *accepted*, and/or *exported*.

6.3.1 Basic Transfer Flavor Names and Contents

The names listed in Section 6.2.4 Names of MathML Encodings are the exact strings that should be used to identify the transfer flavors that correspond to the MathML encodings. On operating systems that allow such, an application should register their support for these flavor names (e.g. on Windows, a call to RegisterClipboardFormat, or, on the Macintosh platform, declaration of support for the universal type identifier in the application descriptor).

When transferring MathML, an application MUST ensure the content of the data transfer is a well-formed XML instance of a MathML document type. Specifically:

1. The instance MAY begin with an XML declaration, e.g. <?xml version="1.0">
2. The instance MUST contain exactly one root `math` element.
3. The instance MUST declare the MathML namespace on the root `math` element.
4. The instance MAY use a `schemaLocation` attribute on the `math` element to indicate the location of the MathML schema that describes the MathML document type to which the instance conforms. The presence of the `schemaLocation` attribute does not require a consumer of the MathML instance to obtain or use the referenced schema.
5. The instance SHOULD use numeric character references (e.g. &#x03b1;) rather than character entity names (e.g. &alpha;) for greater interoperability.

6. The instance MUST specify the character encoding, if it uses an encoding other than UTF-8, either in the XML declaration, or by the use of a byte-order mark (BOM) for UTF-16-encoded data.

6.3.2 Recommended Behaviors when Transferring

An application that transfers MathML markup SHOULD adhere to the following conventions:

1. An application that supports pure presentation markup and/or pure content markup SHOULD offer as many of these flavors as it has available.
2. An application that only exports one MathML flavor SHOULD name it `MathML` if it is unable to determine a more specific flavor.
3. If an application is able to determine a more specific flavor, it SHOULD offer both the generic and specific transfer flavors, but it SHOULD only deliver the specific flavor if it knows that the recipient supports it. For an HTTP GET transfer, for example, the specific transfer types for content and presentation markup should only be returned if they are included in the HTTP `Accept` header sent by the client.
4. An application that exports the two specific transfer flavors SHOULD export both the content and presentation transfer flavors, as well as the generic flavor, which SHOULD combine the other two flavors using a top-level MathML `semantics` element (see Section 5.4.1 Top-level Parallel Markup).
5. When an application exports a MathML fragment whose only child of the root element is a `semantics` element, it SHOULD offer, after the above flavors, a transfer flavor for each `annotation` or `annotation-xml` element, provided the transfer flavor can be recognized and named based on the `encoding` attribute value, and provided the annotation key is (the default) alternate-representation. The transfer content for each annotation should contain the character data in the specified encoding (for an `annotation` element), or a well-formed XML fragment (for an `annotation-xml` element), or the data that results by requesting the URL given by the `src` attribute (for an annotation reference).
6. As a final fallback, an application MAY export a version of the data in a plain-text flavor (such as `text/plain`, `CF_UNICODETEXT`, `UnicodeText`, or `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 processors expect MathML instances transferred as plain text to begin with a `math` element, the text version SHOULD generally omit the XML declaration, DOCTYPE declaration, and other XML prolog material that would appear before the `math` element. 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

To determine whether a MathML instance is pure content markup or pure presentation markup, the `math`, `semantics`, `annotation` and `annotation-xml` elements should be regarded as belonging to both the presentation and content markup vocabularies. The `math` element is treated in this way because it is required as the root element in any MathML transfer. 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, an application that consumes MathML should always process these four elements, even if it only implements one of the two vocabularies.

It is worth noting that the above recommendations allow agents that produce MathML to provide binary data for the clipboard, for example in an image or other application-specific format. The sole method to do so is to reference the binary data using the `src` attribute of an annotation, since XML character data does not allow for the transfer of arbitrary byte-stream data.

While the above recommendations are intended to improve interoperability between MathML-aware applications that use these transfer paradigms, it should be noted that they do not guarantee interoperability. For example, references to external resources (e.g. stylesheets, etc.) in MathML data can cause interoperability problems if the consumer of the data is unable to locate them, as can happen when cutting and pasting HTML or other data types. An application that makes use of references to external resources is encouraged to make users aware of potential problems and provide alternate ways to obtain the referenced resources. In general, consumers of MathML data that contains references they cannot resolve or do not understand should ignore the external references.

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`	`<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 on the Windows platform 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`	`<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 on the Mac OS X platform 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 schemata, markup is stored with a namespace prefix. The system therefore can transfer the following data:

Flavor Name	Flavor Content

`public.mathml.presentation`	`<math xmlns="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">`
	`<mrow>`
	`...`
	`<mrow>`
	`</math>`

Flavor Name	Flavor Content
public.mathml.content	<math xmlns="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"> <apply> ... <apply> </math>
public.mathml	<math xmlns="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"> <mrow> <apply> ... content markup within presentation markup ... </apply> ... </mrow> </math>
public.plain-text.tex	{x \over x-1}
public.plain-text	<math xmlns="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"> <mrow> ... <mrow> </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 MathML-Presentation, MathML-Content, TeX and pictures in TIFF format. In web-pages being browsed, it could produce a MathML fragment such as the following:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <semantics>
    <mrow>...</mrow>
    <annotation-xml encoding="MathML-Content">...</annotation-xml>
    <annotation encoding="TeX">{1 \over x}</annotation>
    <annotation encoding="image/tiff" src="formula3848.tiff"/>
  </semantics>
</math>
```

A web-browser on the Windows platform 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	<math xmlns="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">
	  <mrow>
	  ...
	  <mrow>
	</math>

MathML Content	<math xmlns="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">
	  <apply>
	  ...
	  <apply>
	</math>

MathML	<math xmlns="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">
	  <mrow>
	    <apply>
	       ... content markup within presentation markup ...
	    </apply>
	    ...
	  </mrow>
	</math>

TeX	{x \over x-1}
CF_TIFF	(the content of the picture file, requested from formula3848.tiff)

CF_UNICODETEXT	<math xmlns="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">
	  <mrow>
	  ...
	  <mrow>
	</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 HTML or DocBook. It is also common that other object-level markup languages are also included in a compound document format, such as MathML and SVG in HTML5. 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 it was broken off.

Most document markup languages have some concept of an inline equation, (or graphic, object, etc.) so there is a typically a natural 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 position. 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, the default MathML schema does not allow foreign markup elements to be included within MathML instances.

In the standard schema, elements from other namespaces are not allowed, but attributes from other namespaces are permitted. MathML processors that encounter unknown XML markup should behave as follows:

1. An attribute from a non-MathML namespace should be silently ignored.
2. An element from a non-MathML namespace should be treated as an error, except in an `annotation-xml` element. If the element is a child of a presentation element, it should be handled as described in Section 3.3.5 Error Message `<merror>`. If the element is a child of a content element, it should be handled as described in Section 4.2.9 Error Markup `<cerror>`.

For example, if the second child of an `mfrac` element is an unknown element, the fraction should be rendered with a denominator that indicates the error.

When designing a compound document format in which MathML is included in a larger document type, the designer may extend the content model of MathML to allow additional elements. For example, a common extension is to extend the MathML schema such that elements from non-MathML namespaces are allowed in token elements, but not in other elements. MathML processors that encounter unknown markup should behave as follows:

1. An unrecognized XML attribute should be silently ignored.
2. An unrecognized element in a MathML token element should be silently ignored.
3. An element from a non-MathML namespace should be treated as an error, except in an `annotation-xml` element. If the element is a child of a presentation element, it should be handled as described in Section 3.3.5 Error Message `<merror>`. If the element is a child of a content element, it should be handled as described in Section 4.2.9 Error Markup `<cerror>`.

Extending the schema in this way is easily achieved using the Relax NG schema described in Appendix A Parsing MathML, it may be as simple as including the MathML schema whilst overriding the content model of `mtext`:

```
default namespace m = "http://www.w3.org/1998/Math/MathML"

include "mathml3.rnc" {
mtext = element mtext {mtext.attributes, (token.content|anyElement)*}
}
```

The definition given here would allow any well formed XML that is not in the MathML namespace as a child of `mtext`. In practice this may be too lax. For example, an XHTML+MathML Schema may just want to allow inline XHTML elements as additional children of `mtext`. This may be achieved by replacing `anyElement` by a suitable production from the schema for the host document type, see Section 6.4.1 Mixing MathML and XHTML.

Considerations about mixing markup vocabularies in compound documents arise when a compound document type is first designed. But once the document type 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 store additional information within a MathML instance. Since MathML is most often generated by authoring tools, a particularly common and important case is where an authoring tool needs to store information about its internal state along with a MathML expression, so an author can resume editing from a previous state. For example, placeholders may be used to indicate incomplete parts of an expression, or a insertion point within an expression may need to be stored.

An application that needs to persist private data within a MathML expression 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 requirement, regardless of what may be allowed by the content model of a particular compound document format, MathML permits the storage of private data via the following strategies:

1. In a format that permits the use of XML Namespaces, for small amounts of data, attributes from other namespaces are allowed on all MathML elements.
2. For larger amounts of data, applications may use the `semantics` element, as described in Section 5.1 Annotation Framework.
3. For authoring tools and other applications that need 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, as described in Section 3.7.1 Bind Action to Sub-Expression `<maction>`.

6.4.1 Mixing MathML and XHTML

To fully integrate MathML into XHTML, it should be possible not only to embed MathML in XHTML, but also to embed XHTML in MathML. The schema used for the W3C HTML5 validator extends `mtext` to allow all inline (phrasing) HTML elements (including `svg`) to be used within the content of `mtext`. See the example in Section 3.2.2.2 Embedding HTML in MathML. As noted above, MathML fragments using XHTML elements within `mtext` will not be valid MathML if extracted from the document and used in isolation. Editing tools may offer support for removing any HTML markup from within `mtext` and replacing it by a text alternative.

In most cases, XHTML elements (headings, paragraphs, lists, etc.) either do not apply in mathematical contexts, or MathML already provides equivalent or improved functionality specifically tailored to mathematical content (tables, mathematics style changes, etc.).

Consult the W3C Math Working Group home page for compatibility and implementation suggestions for current browsers and other MathML-aware tools.

6.4.2 Mixing MathML and non-XML contexts

There may be non-XML vocabularies which require markup for mathematical expressions, where it makes sense to reference this specification. HTML is an important example discussed in the next section, however other examples exist. It is possible to use a TeX-like syntax such as `\frac{a}{b}` rather than explicitly using `<mfrac>` and `<mi>`. If a system parses a specified syntax and produces a tree that may be validated against the MathML schema then it may be viewed as as a MathML application. Note however that documents using such a system are not valid MathML. Implementations of such a syntax should, if possible, offer a facility to output any mathematical expressions as MathML in the XML syntax defined here. Such an application would then be a MathML-output-conformant processor as described in Section 2.3.1 MathML Conformance.

6.4.3 Mixing MathML and HTML

An important example of a non-XML based system is defined in [HTML5]. When considering MathML in HTML there are two separate issues to consider. Firstly the schema is extended to allow HTML in `mtext` as described above in the context of XHTML. Secondly an HTML parser is used rather than an XML parser. The parsing of MathML by an HTML parser is normatively defined in [HTML5]. The description there is aimed at parser implementers and written in terms of the state transitions of the parser as it parses each character of the input. The *non-normative* description below aims to give a higher level description and examples.

XML parsing is completely regular, any XML document may be parsed without reference to the particular vocabulary being used. HTML parsing differs in that it is a custom parser for the HTML vocabulary with specific rules for each element. Similarly to XML though, the HTML parser distinguishes parsing from validation; some input, even if it renders correctly, is classed as a *parse error* which may be reported by validators (but typically is not reported by rendering systems).

The main differences that affect MathML usage may be summarized as:

- Attribute values in most cases do not need to be quoted: `<mfenced open=( close=)>` would parse correctly.
- End tags may in many cases be omitted.
- HTML does not support namespaces other than the three built in ones for HTML, MathML and SVG, and does not support namespace prefixes. Thus you can not use a prefix form like `<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">` and while you may use `<math xmlns="http://www.w3.org/1998/Math/MathML">`, the namespace declaration is essentially ignored and the input is treated as `<math>`. In either case the `math` element and its descendants are placed in the MathML namespace. As noted in Chapter 5 Mixing Markup Languages for Mathematical Expressions the lack of namespace support limits some of the possibilities for annotating MathML with markup from other vocabularies when used in HTML.
- Unlike the XML parser, the HTML parser is defined to accept *any* input string and produce a defined result (which may be classified as non-conforming. The extreme example `<math></<><z =5>` for example would be flagged as a parse error by validators but would return a tree corresponding to a math element containing a comment `<` and an element `z` with an attribute that could not be expressed in XML with name `=5` and value `""`.
- Unless inside the token elements `<mtext>`, `<mo>`, `<mn>`, `<mi>`, `<ms>`, or inside an `<annotation-xml>` with `encoding` attribute "text/html" or "annotation/xhtml+xml", the presence of an HTML element will *terminate* the math expression by closing all open MathML elements, so that the HTML element is interpreted as being in the outer HTML context. Any following MathML elements are then not contained in `<math>` so will be parsed as invalid HTML elements and not rendered as MathML. See for example the example given in Section 5.2.3.3 Using `annotation-xml` in HTML documents.

In the interests of compatibility with existing MathML applications authors and editing systems *should* use MathML fragments that are well formed XML, even when embedded in an HTML document. Also as noted above, although applications accepting MathML in HTML documents must accept MathML making use of these HTML parser features, they should offer a way to export MathML in a portable XML syntax.

6.4.4 Linking

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, most user agents have no way to implement nested links or links on elements with no visible rendering; such links may have no effect.

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`

MathML elements that should not be linking elements	
malignmark	maligngroup

For compound document formats that support linking mechanisms, the id attribute should be used to specify the location for a link into a MathML expression. The id attribute is allowed on all MathML elements, and its value must be unique within a document, making it ideal for this purpose.

Note that MathML 2 has no direct support for linking; it refers to the W3C Recommendation "XML Linking Language" [XLink] in defining links in compound document contexts by using an xlink:href attribute. As mentioned above, MathML 3 adds an href attribute for linking so that xlink:href is no longer needed. However, xlink:href is still allowed because MathML permits the use of attributes from non-MathML namespaces. It is recommended that new compound document formats use the MathML 3 href attribute for linking. When user agents encounter MathML elements with both href and xlink:href attributes, the href attribute should take precedence. To support backward compatibility, user agents that implement XML Linking in compound documents containing MathML 2 should continue to support the use of the xlink:href attribute in addition to supporting the href attribute.

6.4.5 MathML and Graphical Markup

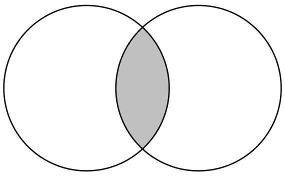
Apart from the introduction of new glyphs, many of the situations where one might be inclined to use an image amount to displaying labeled diagrams. For example, knot diagrams, Venn diagrams, Dynkin diagrams, Feynman diagrams and commutative diagrams all fall into this category. As such, their content 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="image/svg+xml">
    <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 MathML-Content 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 MathML-Content 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

When MathML is rendered in an environment that supports CSS [CSS21], controlling mathematics style properties with a CSS style sheet is desirable, but not as simple as it might first appear, because the formatting of MathML layout schemata is quite different from the CSS visual formatting model and many of the style parameters that affect mathematics layout have no direct textual analogs. Even in cases where there are analogous properties, the sensible values for these properties may not correspond. Because of this difference, applications that support MathML natively may choose to restrict the CSS properties applicable to MathML layout schemata to those properties that do not affect layout.

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 property 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 Mathematics style attributes common to token elements, and more generally throughout Chapter 3 Presentation Markup.

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, stylesheets should be written in a way such that changes to document-wide typographic styles do not affect embedded MathML expressions.

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.

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. The order is defined by [CSS21] cascading order taking into account precedence of non-CSS presentational hints.

7 Characters, Entities and Fonts

7.1 Introduction

Notation and symbols have proved very important for mathematics. Mathematics has grown in part because its notation continually changes toward being succinct and suggestive. Many new signs have been developed for use in mathematical notation, and many have been adopted that were 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 the job of specifying part of the mechanism needed to proceed from notation to final presentation, and has collaborated with the Unicode Technical Committee (UTC) and the STIX Fonts Project in undertaking specification of the rest.

This chapter contains discussion of characters for use within MathML, recommendations for their use, and warnings concerning the correct form of the corresponding code points given in the Universal Multiple-Octet Coded Character Set (UCS) [ISO10646] as codified in Unicode [Unicode]. For simplicity we refer to this character set by the short name Unicode. Unless otherwise stated, the mappings discussed in this chapter and elsewhere in the MathML 3.0 recommendation are based on Unicode 5.2. Conformant MathML processors (see Section 2.3 Conformance) are free to use characters defined in Unicode 5.2 or later.

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. For the latest character tables and font information, see the [Entities] and the Unicode Home Page, notably Unicode Work in Progress and Unicode Technical Report #25 “Unicode Support for Mathematics”.

A MathML token element (see Section 3.2 Token Elements, Section 4.2.1 Numbers `<cn>`, Section 4.2.2 Content Identifiers `<ci>`, Section 4.2.3 Content Symbols `<csymbol>`) takes as content a sequence of MathML characters or `mglyph` elements. The latter are used to represent characters that do not have a Unicode encoding, as described in Section 3.2.1.2 Using images to represent symbols `<mglyph/>`. The need for `mglyph` should be rare because Unicode 3.1 provided approximately one thousand alphabetic characters for mathematics, and Unicode 3.2 added over 900 more special mathematical symbols.

7.2 Unicode Character Data

Any character allowed by XML may be used in MathML. More precisely, the legal Unicode 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+10000..U+10FFFF). 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 in an XML document.

- Using characters directly: For example, the 'é' (character U+00E9 [LATIN SMALL LETTER E WITH ACUTE]) may have been inserted. This option is only useful if the character encoding specified for the XML document includes the character intended. 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; for instance character U+00E9 (eacute) is, but character U+03B1 (alpha) is not.

- Using numeric XML character references: For example, 'é' may be represented as &#233; (decimal) or &#xE9; (hex), or &#0233; (decimal) or &#x00E9;. Note that the numbers in the character references 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.
- Using entity references: The MathML DTD defines internal entities that expand to character data. Thus for example the entity reference &eacute; may be used rather than the character reference &#xE9;. 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 can be useful for small illustrative examples.

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 an ancillary document, XML Entity Definitions for Characters [Entities]. The tables there list the entity names and the corresponding Unicode character references. That document describes several entity sets; not all of them are used in the MathML DTD. The MathML DTD references the combined HTML MathML entity set defined in [Entities].

7.4 Special Characters Not in Unicode

For special purposes, one may need a symbol which does not have a Unicode representation. 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-Unicode encoding. All MathML token elements accept characters in their content and also accept an `mglyph` there. 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 a useful alternative representation in its `alt` attribute.

7.5 Mathematical Alphanumeric Symbols

In mathematical and scientific writing, single letters often 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 generally *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 as of a certain mathematical type by specialists in a given area: 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.

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 Mathematics style attributes common to token elements, MathML offers an alternative mechanism to specify mathematical alphanumeric characters. This alternative mechanism spans the gap between the specification of the mathematical alphanumeric symbols as Unicode code points, and the deployment of software and fonts that support them. Namely, one uses the `mathvariant` attribute on a token element such as `mi` to indicate that the character data in the token element selects a mathematical alphanumeric symbol.

In principle, any mathvariant value may be used with any character data to define a specific symbolic token. In practice, only certain combinations of character data and mathvariant values will be visually distinguished by a given renderer. In this section we explain the correspondence between certain characters in Plane 0 that, when modified by the `mathvariant` attribute, are considered equivalent to mathematical alphanumeric symbol characters.

The mathematical alphanumeric symbol characters in Plane 1 include alphabets for Latin upper-case and lower-case letters, including dotless i and j, Greek upper-case and lower-case letters, Greek symbols (also known as variants), including upper-case and lower-case digamma, and Latin digits. These alphabets provide Plane 1 Unicode code points that differ from corresponding Plane 0 characters only by a variation in font that carries mathematical semantics when used in a formula.

The `mathvariant` attribute uses exactly this correspondence to provide an alternate markup encoding that selects these Plane 1 characters. For example, the Mathematical Italic alphabet runs from U+1D434 ("A") to U+1D467 ("z"). Thus, a typical example of an identifier for a variable, marked up as

```
<mi>a</mi>
```

and rendered in a mathematical italic font (as described in Section 3.2.3 Identifier `<mi>`) could equivalently be marked up as

```
<mi>&#x1D44E;<!--MATHEMATICAL ITALIC SMALL A--></mi>
```

which invokes the Mathematical Italic lower-case a explicitly.

An important use of the mathematical alphanumeric symbols in Plane 1 is for identifiers normally printed in special mathematical fonts, such as Fraktur, Greek, Boldface, or Script. As another example, the Mathematical Fraktur alphabet runs from U+1D504 ("A") to U+1D537 ("z"). Thus, an identifier for a variable that uses Fraktur characters could be marked up as

```
<mi>&#x1D504;<!--MATHEMATICAL FRAKTUR CAPITAL A--></mi>
```

An alternative, equivalent markup for this example is to use the common upper-case A, modified by using the `mathvariant` attribute:

```
<mi mathvariant="fraktur">A</mi>
```

A MathML processor must treat a mathematical alphanumeric character (when it appears) as identical to the corresponding combination of the unstyled character and mathvariant attribute value. It is important to note that the mathvariant attribute specifies a semantic class of characters, each of which has a specific appearance that should be protected from document-wide style changes, so the intended meaning of the character may be preserved. The use of a mathematical alphanumeric character is also intended to preserve this specific appearance, and so these characters are also not to be affected by surrounding style changes.

Not all combinations of character data and mathvariant values have assigned Unicode code points. For example, sans-serif Greek alphabets are omitted, while bold sans-serif Greek alphabets are included, and bold digits are included, while bold-italic digits are excluded. A renderer should visually distinguish those combinations of character data and mathvariant attribute values that it can subject to the availability of font characters. It is intended that renderers distinghish at least those combinations that have equivalent Unicode code points, and renderers are free to ignore those combinations that have no assigned Unicode code point or for which adequate font support is unavailable.

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 in Plane 0 are not in the 'expected' sequence in Plane 1. The table below shows the Plane 0 mathematical alphanumeric symbols, listing for each character its Unicode code point, its Unicode character name, its corresponding unstyled alphabetic character, and the code point in Plane 1 where one might naturally have sought this character.

Unicode code point	Unicode name	BMP code	Plane-1 code
U+210E	PLANCK CONSTANT	U+0068	U+1D455
U+212C	SCRIPT CAPITAL B	U+0042	U+1D49D
U+2130	SCRIPT CAPITAL E	U+0045	U+1D4A0
U+2131	SCRIPT CAPITAL F	U+0046	U+1D4A1
U+210B	SCRIPT CAPITAL H	U+0048	U+1D4A3
U+2110	SCRIPT CAPITAL I	U+0049	U+1D4A4
U+2112	SCRIPT CAPITAL L	U+004C	U+1D4A7
U+2133	SCRIPT CAPITAL M	U+004D	U+1D4A8
U+211B	SCRIPT CAPITAL R	U+0052	U+1D4AD
U+212F	SCRIPT SMALL E	U+0065	U+1D4BA
U+210A	SCRIPT SMALL G	U+0067	U+1D4BC
U+2134	SCRIPT SMALL O	U+006F	U+1D4C4
U+212D	BLACK-LETTER CAPITAL C	U+0043	U+1D506
U+210C	BLACK-LETTER CAPITAL H	U+0048	U+1D50B
U+2111	BLACK-LETTER CAPITAL I	U+0049	U+1D50C
U+211C	BLACK-LETTER CAPITAL R	U+0052	U+1D515
U+2128	BLACK-LETTER CAPITAL Z	U+005A	U+1D51D
U+2102	DOUBLE-STRUCK CAPITAL C	U+0043	U+1D53A
U+210D	DOUBLE-STRUCK CAPITAL H	U+0048	U+1D53F
U+2115	DOUBLE-STRUCK CAPITAL N	U+004E	U+1D545
U+2119	DOUBLE-STRUCK CAPITAL P	U+0050	U+1D547
U+211A	DOUBLE-STRUCK CAPITAL Q	U+0051	U+1D548
U+211D	DOUBLE-STRUCK CAPITAL R	U+0052	U+1D549
U+2124	DOUBLE-STRUCK CAPITAL Z	U+005A	U+1D551

Mathematical Alphanumeric Symbol characters should not be used for styled prose. For example, Mathematical Fraktur A must not be used to just select a blackletter font for an uppercase A as it 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 Presentation Markup and Chapter 4 Content Markup.

In MathML, control of page composition, such as line-breaking, is effected by the use of the proper attributes on the `mo` and `mspace` elements.

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 code point	Unicode name	Description
U+2061	FUNCTION APPLICATION	character showing function application in presentation tagging (Section 3.2.5 Operator, Fence, Separator or Accent `<mo>`)
U+2062	INVISIBLE TIMES	marks multiplication when it is understood without a mark (Section 3.2.5 Operator, Fence, Separator or Accent `<mo>`)
U+2063	INVISIBLE SEPARATOR	used as a separator, e.g., in indices (Section 3.2.5 Operator, Fence, Separator or Accent `<mo>`)
U+2064	INVISIBLE PLUS	marks addition, especially in constructs such as 1½ (Section 3.2.5 Operator, Fence, Separator or Accent `<mo>`)

7.7 Anomalous Mathematical Characters

Some characters which occur fairly often in mathematical texts, and have special significance there, are frequently confused with other similar characters in the UCS. In some cases, common keyboard characters have become entrenched as alternatives to the more appropriate mathematical characters. In others, characters have legitimate uses in both formulas and text, but conflicting rendering and font conventions. All these characters are called here anomalous characters.

7.7.1 Keyboard Characters

Typical Latin-1-based keyboards contain several characters that are visually similar to important mathematical characters. Consequently, these characters are frequently substituted, intentionally or unintentionally, for their more correct mathematical counterparts.

7.7.1.1 Minus

The most common ordinary text character which enjoys a special mathematical use is U+002D [HYPHEN-MINUS]. As its Unicode name suggests, it is used as a hyphen in prose contexts, and as a minus or negative sign in formulas. For text use, there is a specific code point U+2010 [HYPHEN] which is intended for prose contexts, and which should render as a hyphen or short dash. For mathematical use, there is another code point U+2212 [MINUS SIGN] which is intended for mathematical formulas, and which should render as a longer minus or negative sign. MathML renderers should treat U+002D [HYPHEN-MINUS] as equivalent to U+2212 [MINUS SIGN] in formula contexts such as `mo`, and as equivalent to U+2010 [HYPHEN] in text contexts such as `mtext`.

7.7.1.2 Apostrophes, Quotes and Primes

On a typical European keyboard there is a key available which is viewed as an apostrophe or a single quotation mark (an upright or right quotation mark). Thus one key is doing double duty for prose input to enter U+0027 [APOSTROPHE] and U+2019 [RIGHT SINGLE QUOTATION MARK]. In mathematical contexts it is also commonly used for the prime, which should be U+2032 [PRIME]. Unicode recognizes the overloading of this symbol and remarks that it can also signify the units of minutes or feet. In the unstructured printed text of normal prose the characters are placed next to one another. The U+0027 [APOSTROPHE] and U+2019 [RIGHT SINGLE QUOTATION MARK] are marked with glyphs that are small and raised with respect to the center line of the text. The fonts used provide small raised glyphs in the appropriate places indexed by the Unicode codes. The U+2032 [PRIME] of mathematics is similarly treated in fuller Unicode fonts.

MathML renderers are encouraged to treat U+0027 [APOSTROPHE] as U+2032 [PRIME] when appropriate in formula contexts.

A final remark is that a ‘prime’ is often used in transliteration of the Cyrillic character U+044C [CYRILLIC SMALL LETTER SOFT SIGN]. This different use of primes is not part of considerations for mathematical formulas.

7.7.1.3 Other Keyboard Substitutions

While the minus and prime characters are the most common and important keyboard characters with more precise mathematical counterparts, there are a number of other keyboard character substitutions that are sometime used. For example some may expect

```
<mo>''</mo>
```

to be treated as U+2033 [DOUBLE PRIME], and analogous substitutions could perhaps be made for U+2034 [TRIPLE PRIME] and U+2057 [QUADRUPLE PRIME]. Similarly, sometimes U+007C [VERTICAL LINE] is used for U+2223 [DIVIDES]. MathML regards these as application-specific authoring conventions, and recommends that authoring tools generate markup using the more precise mathematical characters for better interoperability.

7.7.2 Pseudo-scripts

There are a number of characters in the UCS that traditionally have been taken to have a natural ‘script’ aspect. The visual presentation of these characters is similar to a script, that is, raised from the baseline, and smaller than the base font size. The degree symbol and prime characters are examples. For use in text, such characters occur in sequence with the identifier they follow, and are typically rendered using the same font. These characters are called pseudo-scripts here.

In almost all mathematical contexts, pseudo-script characters should be associated with a base expression using explicit script markup in MathML. For example, the preferred encoding of "x prime" is

```
<msup><mi>x</mi><mo>&#x2032;<!--PRIME--></mo></msup>
```

and not

```
<mi>x'</mi>
```

or any other variants not using an explicit script construct. Note, however, that within text contexts such as `mtext`, pseudo-scripts may be used in sequence with other character data.

There are two reasons why explicit markup is preferable in mathematical contexts. First, a problem arises with typesetting, when pseudo-scripts are used with subscripted identifiers. Traditionally, subscripting of x' would be rendered stacked under the prime. This is easily accomplished with script markup, for example:

```
<mrow><msubsup><mi>x</mi><mn>0</mn><mo>&#x2032;<!--PRIME--></mo></msubsup></mrow>
```

By contrast,

```
<mrow><msub><mi>x'</mi><mn>0</mn></msub></mrow>
```

will render with staggered scripts.

Note this means that a renderer of MathML will have to treat pseudo-scripts differently from most other character codes it finds in a superscript position; in most fonts, the glyphs for pseudo-scripts are already shrunk and raised from the baseline.

The second reason that explicit script markup is preferrable to juxtaposition of characters is that it generally better reflects the intended mathematical structure. For example,

```
<msup>
  <mrow><mo>(</mo><mrow><mi>f</mi><mo>+</mo><mi>g</mi></mrow><mo>)</mo>
                                                                      </mrow>
  <mo>&#x2032;<!--PRIME--></mo>
</msup>
```

accurately reflects that the prime here is operating on an entire expression, and does not suggest that the prime is acting on the final right parenthesis.

However, the data model for all MathML token elements is Unicode text, so one cannot rule out the possibility of valid MathML markup containing constructions such as

```
<mrow><mi>x'</mi></mrow>
```

and

```
<mrow><mi>x</mi><mo>'</mo></mrow>
```

While the first form may, in some rare situations, legitmately be used to distinguish a multi-character identifer named x' from the derivative of a function x, such forms should generally be avoided. Authoring and validation tools are encouraged to generate the recommended script markup:

```
<mrow><msup><mi>x</mi><mo>&#x2032;<!--PRIME--></mo></msup></mrow>
```

The U+2032 [PRIME] character is perhaps the most common pseudo-script, but there are many others, as listed below:

Pseudo-script Characters

U+0022	QUOTATION MARK
U+0027	APOSTROPHE
U+002A	ASTERISK
U+0060	GRAVE ACCENT
U+00AA	FEMININE ORDINAL INDICATOR
U+00B0	DEGREE SIGN
U+00B2	SUPERSCRIPT TWO
U+00B3	SUPERSCRIPT THREE
U+00B4	ACUTE ACCENT
U+00B9	SUPERSCRIPT ONE
U+00BA	MASCULINE ORDINAL INDICATOR

Pseudo-script Characters	
U+2018	LEFT SINGLE QUOTATION MARK
U+2019	RIGHT SINGLE QUOTATION MARK
U+201A	SINGLE LOW-9 QUOTATION MARK
U+201B	SINGLE HIGH-REVERSED-9 QUOTATION MARK
U+201C	LEFT DOUBLE QUOTATION MARK
U+201D	RIGHT DOUBLE QUOTATION MARK
U+201E	DOUBLE LOW-9 QUOTATION MARK
U+201F	DOUBLE HIGH-REVERSED-9 QUOTATION MARK
U+2032	PRIME
U+2033	DOUBLE PRIME
U+2034	TRIPLE PRIME
U+2035	REVERSED PRIME
U+2036	REVERSED DOUBLE PRIME
U+2037	REVERSED TRIPLE PRIME
U+2057	QUADRUPLE PRIME

In addition, the characters in the Unicode Superscript and Subscript block (beginning at U+2070) should be treated as pseudo-scripts when they appear in mathematical formulas.

Note that several of these characters are common on keyboards, including U+002A [ASTERISK], U+00B0 [DEGREE SIGN], U+2033 [DOUBLE PRIME], and U+2035 [REVERSED PRIME] also known as a back prime.

7.7.3 Combining Characters

In the UCS there are many combining characters that are intended to be used for the many accents of numerous different natural languages. Some of them may seem to provide markup needed for mathematical accents. They should not be used in mathematical markup. Superscript, subscript, underscript, and overscript constructions as just discussed above should be used for this purpose. Of course, combining characters may be used in multi-character identifiers as they are needed, or in text contexts.

There is one more case where combining characters turn up naturally in mathematical markup. Some relations have associated negations, such as U+226F [NOT GREATER-THAN] for the negation of U+003E [GREATER-THAN SIGN]. The glyph for U+226F [NOT GREATER-THAN] is usually just that for U+003E [GREATER-THAN SIGN] with a slash through it. Thus it could also be expressed by U+003E-0338 making use of the combining slash U+0338 [COMBINING LONG SOLIDUS OVERLAY]. That is true of 25 other characters in common enough mathematical use to merit their own Unicode code points. In the other direction there are 31 character entity names listed in [Entities] which are to be expressed using U+0338 [COMBINING LONG SOLIDUS OVERLAY].

In a similar way there are mathematical characters which have negations given by a vertical bar overlay U+20D2 [COMBINING LONG VERTICAL LINE OVERLAY]. Some are available in pre-composed forms, and some named character entities are given explicitly as combinations. In addition there are examples using U+0333 [COMBINING DOUBLE LOW LINE] and U+20E5 [COMBINING REVERSE SOLIDUS OVERLAY], and variants specified by use of the U+FE00 [VARIATION SELECTOR-1]. For fuller listing of these cases see the listings in [Entities].

The general rule is that a base character followed by a string of combining characters should be treated just as though it were the pre-composed character that results from the combination, if such a character exists.

A Parsing MathML

A.1 Use of MathML as Well-Formed XML

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 advantageous *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 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 &#x222B; rather than &int;

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 Using the RelaxNG Schema for MathML3, discusses the RelaxNG Schema for MathML3 [RELAX-NG], which forms a normative part of the specification. Following that, Section A.4 Using the MathML XML Schema, and Section A.3 Using the MathML DTD 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 schema for MathML3 in five parts:

- The grammar for full MathML
- The grammar for elements common to Content and Presentation
- The grammar for Presentation MathML
- The grammar for Strict Content MathML
- The grammar for Content MathML3

A.2.1 Full MathML

The RelaxNG schema for full MathML builds on the schema describing the various parts of the 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 1998-2010 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


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"
```

A.2.2 Elements Common to Presentation and Content 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 1998-2014 W3C (MIT, ERCIM, Keio, Beihang)
#
#     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


default namespace m = "http://www.w3.org/1998/Math/MathML"
namespace local = ""

start = math

math = element math {math.attributes, MathExpression*}
MathExpression = semantics

NonMathMLAtt = attribute (* - (local:*|m:*)) {xsd:string}

CommonDeprecatedAtt = attribute other {text}?

CommonAtt = attribute id {xsd:ID}?,
            attribute xref {text}?,
            attribute class {xsd:NMTOKENS}?,
            attribute style {xsd:string}?,
            attribute href {xsd:anyURI}?,
            CommonDeprecatedAtt,
            NonMathMLAtt*
```

```
math.attributes = CommonAtt,
      attribute display {"block" | "inline"}?,
      attribute maxwidth {length}?,
      attribute overflow {"linebreak" | "scroll" | "elide" | "truncate" | "scale"}?,
      attribute altimg {xsd:anyURI}?,
      attribute altimg-width {length}?,
      attribute altimg-height {length}?,
      attribute altimg-valign {length | "top" | "middle" | "bottom"}?,
      attribute alttext {text}?,
      attribute cdgroup {xsd:anyURI}?,
      math.deprecatedattributes

# the mathml3-presentation schema  adds additional attributes
# to the math element, all those valid on mstyle

math.deprecatedattributes = attribute mode {xsd:string}?,
                            attribute macros {xsd:string}?


name = attribute name {xsd:NCName}
cd = attribute cd {xsd:NCName}

src = attribute src {xsd:anyURI}?

annotation = element annotation {annotation.attributes,text}

annotation-xml.model = (MathExpression|anyElement)*

anyElement =  element (* - m:*) {(attribute * {text}|text| anyElement)*}

annotation-xml = element annotation-xml {annotation.attributes,
                                          annotation-xml.model}
annotation.attributes = CommonAtt,
                        cd?,
                        name?,
                        DefEncAtt,
                        src?

DefEncAtt = attribute encoding {xsd:string}?,
            attribute definitionURL {xsd:anyURI}?

semantics = element semantics {semantics.attributes,
                               MathExpression,
                              (annotation|annotation-xml)*}
semantics.attributes = CommonAtt, DefEncAtt, cd?, name?



length
# wrapped for display
 = xsd:string {
  pattern = '\s*((-?[0-9]*([0-9]\.?|\.[0-9])[0-9]*(e[mx]|in|cm|mm|p[xtc]|%)?)|
             (negative)?((very){0,2}thi(n|ck)|medium)mathspace)\s*'
}
```

A.2.3 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 1998-2014 W3C (MIT, ERCIM, Keio, Beihang)
#
#     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

default namespace m = "http://www.w3.org/1998/Math/MathML"

MathExpression |= PresentationExpression

ImpliedMrow = MathExpression*

TableRowExpression = mtr|mlabeledtr

TableCellExpression = mtd

MstackExpression = MathExpression|mscarries|msline|msrow|msgroup

MsrowExpression = MathExpression|none

MultiScriptExpression = (MathExpression|none),(MathExpression|none)

mpadded-length
# wrapped for display
 = xsd:string {
  pattern = '\s*([\+\-]?[0-9]*([0-9]\.?|\.[0-9])[0-9]*\s*((%?\s*(height|depth|
             width)?)|e[mx]|in|cm|mm|p[xtc]|((negative)?((very){0,2}thi(n|ck)|
             medium)mathspace))?)\s*'}

linestyle = "none" | "solid" | "dashed"

verticalalign =
      "top" |
      "bottom" |
      "center" |
      "baseline" |
      "axis"

columnalignstyle = "left" | "center" | "right"

notationstyle =
     "longdiv" |
     "actuarial" |
     "radical" |
     "box" |
     "roundedbox" |
     "circle" |
     "left" |
     "right" |
     "top" |
     "bottom" |
     "updiagonalstrike" |
     "downdiagonalstrike" |
     "verticalstrike" |
     "horizontalstrike" |
     "madruwb"

idref = text
unsigned-integer = xsd:unsignedLong
integer = xsd:integer
```

```
number = xsd:decimal

character = xsd:string {
  pattern = '\s*\S\s*'}

color
# wrapped for display
 =  xsd:string {
  pattern = '\s*((#[0-9a-fA-F]{3}([0-9a-fA-F]{3})?)|[aA][qQ][uU][aA]|
             [bB][lL][aA][cC][kK]|[bB][lL][uU][eE]|[fF][uU][cC][hH][sS][iI][aA]|
             [gG][rR][aA][yY]|[gG][rR][eE][eE][nN]|[lL][iI][mM][eE]|
             [mM][aA][rR][oO][oO][nN]|[nN][aA][vV][yY]|[oO][lL][iI][vV][eE]|
             [pP][uU][rR][pP][lL][eE]|[rR][eE][dD]|[sS][iI][lL][vV][eE][rR]|
             [tT][eE][aA][lL]|[wW][hH][iI][tT][eE]|[yY][eE][lL][lL][oO][wW])\s*'}


group-alignment = "left" | "center" | "right" | "decimalpoint"
group-alignment-list = list {group-alignment+}
group-alignment-list-list
# wrapped for display
 = xsd:string {
  pattern = '(\s*\{\s*(left|center|right|decimalpoint)
             (\s+(left|center|right|decimalpoint))*\})*\s*'}
positive-integer = xsd:positiveInteger


TokenExpression = mi|mn|mo|mtext|mspace|ms

token.content = mglyph|malignmark|text

mi = element mi {mi.attributes, token.content*}
mi.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt


mn = element mn {mn.attributes, token.content*}
mn.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt


mo = element mo {mo.attributes, token.content*}
mo.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt,
  attribute form {"prefix" | "infix" | "postfix"}?,
  attribute fence {"true" | "false"}?,
  attribute separator {"true" | "false"}?,
  attribute lspace {length}?,
  attribute rspace {length}?,
  attribute stretchy {"true" | "false"}?,
  attribute symmetric {"true" | "false"}?,
  attribute maxsize {length | "infinity"}?,
  attribute minsize {length}?,
  attribute largeop {"true" | "false"}?,
  attribute movablelimits {"true" | "false"}?,
  attribute accent {"true" | "false"}?,
  attribute linebreak {"auto" | "newline" | "nobreak" | "goodbreak" | "badbreak"}?,
  attribute lineleading {length}?,
  attribute linebreakstyle {"before" | "after" | "duplicate" | "infixlinebreakstyle"}?,
  attribute linebreakmultchar {text}?,
  attribute indentalign {"left" | "center" | "right" | "auto" | "id"}?,
  attribute indentshift {length}?,
  attribute indenttarget {idref}?,
  attribute indentalignfirst {"left" | "center" | "right" | "auto" | "id" |
                                                                 "indentalign"}?,
  attribute indentshiftfirst {length | "indentshift"}?,
  attribute indentalignlast {"left" | "center" | "right" | "auto" | "id" |
                                                                 "indentalign"}?,
  attribute indentshiftlast {length | "indentshift"}?


mtext = element mtext {mtext.attributes, token.content*}
mtext.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt


mspace = element mspace {mspace.attributes, empty}
mspace.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt,
  attribute width {length}?,
  attribute height {length}?,
  attribute depth {length}?,
  attribute linebreak {"auto" | "newline" | "nobreak" | "goodbreak" | "badbreak" |
                                                                 "indentingnewline"}?,
  attribute indentalign {"left" | "center" | "right" | "auto" | "id"}?,
  attribute indentshift {length}?,
  attribute indenttarget {idref}?,
  attribute indentalignfirst {"left" | "center" | "right" | "auto" | "id" |
                                                                 "indentalign"}?,
  attribute indentshiftfirst {length | "indentshift"}?,
  attribute indentalignlast {"left" | "center" | "right" | "auto" | "id" |
                                                                 "indentalign"}?,
  attribute indentshiftlast {length | "indentshift"}?


ms = element ms {ms.attributes, token.content*}
ms.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt,
  attribute lquote {text}?,
  attribute rquote {text}?


mglyph = element mglyph {mglyph.attributes, mglyph.deprecatedattributes,empty}
mglyph.attributes =
  CommonAtt,  CommonPresAtt,
  attribute src {xsd:anyURI}?,
  attribute width {length}?,
  attribute height {length}?,
  attribute valign {length}?,
  attribute alt {text}?
mglyph.deprecatedattributes =
  attribute index {integer}?,
  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 {"small" | "normal" | "big" | length}?,
  DeprecatedTokenAtt

msline = element msline {msline.attributes,empty}
msline.attributes =
  CommonAtt,  CommonPresAtt,
  attribute position {integer}?,
  attribute length {unsigned-integer}?,
  attribute leftoverhang {length}?,
  attribute rightoverhang {length}?,
  attribute mslinethickness {length | "thin" | "medium" | "thick"}?

none = element none {none.attributes,empty}
none.attributes =
  CommonAtt,
  CommonPresAtt

mprescripts = element mprescripts {mprescripts.attributes,empty}
mprescripts.attributes =
  CommonAtt,
  CommonPresAtt


CommonPresAtt =
  attribute mathcolor {color}?,
  attribute mathbackground {color | "transparent"}?

TokenAtt =
  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 {"small" | "normal" | "big" | length}?,
  attribute dir {"ltr" | "rtl"}?,
  DeprecatedTokenAtt

DeprecatedTokenAtt =
  attribute fontfamily {text}?,
  attribute fontweight {"normal" | "bold"}?,
  attribute fontstyle {"normal" | "italic"}?,
  attribute fontsize {length}?,
  attribute color {color}?,
  attribute background {color | "transparent"}?

MalignExpression = maligngroup|malignmark

malignmark = element malignmark {malignmark.attributes, empty}
malignmark.attributes =
  CommonAtt, CommonPresAtt,
  attribute edge {"left" | "right"}?


maligngroup = element maligngroup {maligngroup.attributes, empty}
maligngroup.attributes =
  CommonAtt, CommonPresAtt,
  attribute groupalign {"left" | "center" | "right" | "decimalpoint"}?


PresentationExpression =
  TokenExpression | MalignExpression | mrow | mfrac | msqrt | mroot | mstyle |
  merror | mpadded | mphantom | mfenced | menclose | msub | msup | msubsup |
  munder | mover | munderover | mmultiscripts | mtable | mstack | mlongdiv |
  maction



mrow = element mrow {mrow.attributes, MathExpression*}
mrow.attributes =
  CommonAtt, CommonPresAtt,
  attribute dir {"ltr" | "rtl"}?


mfrac = element mfrac {mfrac.attributes, MathExpression, MathExpression}
mfrac.attributes =
  CommonAtt, CommonPresAtt,
  attribute linethickness {length | "thin" | "medium" | "thick"}?,
  attribute numalign {"left" | "center" | "right"}?,
  attribute denomalign {"left" | "center" | "right"}?,
  attribute bevelled {"true" | "false"}?


msqrt = element msqrt {msqrt.attributes, ImpliedMrow}
msqrt.attributes =
  CommonAtt, CommonPresAtt


mroot = element mroot {mroot.attributes, MathExpression, MathExpression}
mroot.attributes =
  CommonAtt, CommonPresAtt


mstyle = element mstyle {mstyle.attributes, ImpliedMrow}
mstyle.attributes =
  CommonAtt, CommonPresAtt,
  mstyle.specificattributes,
  mstyle.generalattributes,
  mstyle.deprecatedattributes

mstyle.specificattributes =
  attribute scriptlevel {integer}?,
  attribute displaystyle {"true" | "false"}?,
  attribute scriptsizemultiplier {number}?,
  attribute scriptminsize {length}?,
  attribute infixlinebreakstyle {"before" | "after" | "duplicate"}?,
  attribute decimalpoint {character}?

mstyle.generalattributes =
  attribute accent {"true" | "false"}?,
  attribute accentunder {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?,
  attribute alignmentscope {list {("true" | "false") +}}?,
  attribute bevelled {"true" | "false"}?,
  attribute charalign {"left" | "center" | "right"}?,
  attribute charspacing {length | "loose" | "medium" | "tight"}?,
  attribute close {text}?,
  attribute columnalign {list {columnalignstyle+}}?,
  attribute columnlines {list {linestyle +}}?,
  attribute columnspacing {list {(length) +}}?,
  attribute columnspan {positive-integer}?,
  attribute columnwidth {list {("auto" | length | "fit") +}}?,
  attribute crossout {list {("none" | "updiagonalstrike" | "downdiagonalstrike" |
                                           "verticalstrike" | "horizontalstrike")*}}?,
  attribute denomalign {"left" | "center" | "right"}?,
  attribute depth {length}?,
  attribute dir {"ltr" | "rtl"}?,
  attribute edge {"left" | "right"}?,
  attribute equalcolumns {"true" | "false"}?,
  attribute equalrows {"true" | "false"}?,
  attribute fence {"true" | "false"}?,
  attribute form {"prefix" | "infix" | "postfix"}?,
  attribute frame {linestyle}?,
  attribute framespacing {list {length, length}}?,
  attribute groupalign {group-alignment-list-list}?,
  attribute height {length}?,
  attribute indentalign {"left" | "center" | "right" | "auto" | "id"}?,
  attribute indentalignfirst {"left" | "center" | "right" | "auto" | "id" |
                                                                      "indentalign"}?,
  attribute indentalignlast {"left" | "center" | "right" | "auto" | "id" |
                                                                      "indentalign"}?,
  attribute indentshift {length}?,
  attribute indentshiftfirst {length | "indentshift"}?,
  attribute indentshiftlast {length | "indentshift"}?,
  attribute indenttarget {idref}?,
  attribute largeop {"true" | "false"}?,
  attribute leftoverhang {length}?,
  attribute length {unsigned-integer}?,
  attribute linebreak {"auto" | "newline" | "nobreak" | "goodbreak" | "badbreak"}?,
  attribute linebreakmultchar {text}?,
  attribute linebreakstyle {"before" | "after" | "duplicate" | "infixlinebreakstyle"}?,
  attribute lineleading {length}?,
  attribute linethickness {length | "thin" | "medium" | "thick"}?,
  attribute location {"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"}?,
  attribute longdivstyle {"lefttop" | "stackedrightright" |
          "mediumstackedrightright" | "shortstackedrightright" | "righttop" |
          "left/\right" | "left)(right" | ":right=right" | "stackedleftleft" |
          "stackedleftlinetop"}?,
  attribute lquote {text}?,
  attribute lspace {length}?,
  attribute mathsize {"small" | "normal" | "big" | length}?,
  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 maxsize {length | "infinity"}?,
  attribute minlabelspacing {length}?,
  attribute minsize {length}?,
  attribute movablelimits {"true" | "false"}?,
  attribute mslinethickness {length | "thin" | "medium" | "thick"}?,
  attribute notation {text}?,
  attribute numalign {"left" | "center" | "right"}?,
  attribute open {text}?,
  attribute position {integer}?,
  attribute rightoverhang {length}?,
  attribute rowalign {list {verticalalign+}}?,
  attribute rowlines {list {linestyle +}}?,
  attribute rowspacing {list {(length) +}}?,
  attribute rowspan {positive-integer}?,
  attribute rquote {text}?,
  attribute rspace {length}?,
  attribute selection {positive-integer}?,
  attribute separator {"true" | "false"}?,
  attribute separators {text}?,
  attribute shift {integer}?,
  attribute side {"left" | "right" | "leftoverlap" | "rightoverlap"}?,
  attribute stackalign {"left" | "center" | "right" | "decimalpoint"}?,
  attribute stretchy {"true" | "false"}?,
  attribute subscriptshift {length}?,
  attribute superscriptshift {length}?,
  attribute symmetric {"true" | "false"}?,
  attribute valign {length}?,
  attribute width {length}?

mstyle.deprecatedattributes =
  DeprecatedTokenAtt,
  attribute veryverythinmathspace {length}?,
  attribute verythinmathspace {length}?,
  attribute thinmathspace {length}?,
  attribute mediummathspace {length}?,
  attribute thickmathspace {length}?,
  attribute verythickmathspace {length}?,
  attribute veryverythickmathspace {length}?

math.attributes &= CommonPresAtt
math.attributes &= mstyle.specificattributes
math.attributes &= mstyle.generalattributes


merror = element merror {merror.attributes, ImpliedMrow}
merror.attributes =
  CommonAtt, CommonPresAtt


mpadded = element mpadded {mpadded.attributes, ImpliedMrow}
mpadded.attributes =
  CommonAtt, CommonPresAtt,
  attribute height {mpadded-length}?,
  attribute depth {mpadded-length}?,
  attribute width {mpadded-length}?,
  attribute lspace {mpadded-length}?,
  attribute voffset {mpadded-length}?


mphantom = element mphantom {mphantom.attributes, ImpliedMrow}
mphantom.attributes =
  CommonAtt, CommonPresAtt


mfenced = element mfenced {mfenced.attributes, MathExpression*}
mfenced.attributes =
  CommonAtt, CommonPresAtt,
  attribute open {text}?,
  attribute close {text}?,
  attribute separators {text}?


menclose = element menclose {menclose.attributes, ImpliedMrow}
```

```
menclose.attributes =
  CommonAtt, CommonPresAtt,
  attribute notation {text}?


msub = element msub {msub.attributes, MathExpression, MathExpression}
msub.attributes =
  CommonAtt, CommonPresAtt,
  attribute subscriptshift {length}?


msup = element msup {msup.attributes, MathExpression, MathExpression}
msup.attributes =
  CommonAtt, CommonPresAtt,
  attribute superscriptshift {length}?


msubsup = element msubsup {msubsup.attributes, MathExpression, MathExpression,
                                                                       MathExpression}
msubsup.attributes =
  CommonAtt, CommonPresAtt,
  attribute subscriptshift {length}?,
  attribute superscriptshift {length}?


munder = element munder {munder.attributes, MathExpression, MathExpression}
munder.attributes =
  CommonAtt, CommonPresAtt,
  attribute accentunder {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?


mover = element mover {mover.attributes, MathExpression, MathExpression}
mover.attributes =
  CommonAtt, CommonPresAtt,
  attribute accent {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?


munderover = element munderover {munderover.attributes, MathExpression,
                                                       MathExpression, MathExpression}
munderover.attributes =
  CommonAtt, CommonPresAtt,
  attribute accent {"true" | "false"}?,
  attribute accentunder {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?


mmultiscripts = element mmultiscripts {mmultiscripts.attributes, MathExpression,
                        MultiScriptExpression*,(mprescripts, MultiScriptExpression*)?}
mmultiscripts.attributes =
  msubsup.attributes


mtable = element mtable {mtable.attributes, TableRowExpression*}
mtable.attributes =
  CommonAtt, CommonPresAtt,
  attribute align {xsd:string {
    pattern ='\s*(top|bottom|center|baseline|axis)(\s+-?[0-9]+)?\s*'}}?,
  attribute rowalign {list {verticalalign+}}?,
  attribute columnalign {list {columnalignstyle+}}?,
  attribute groupalign {group-alignment-list-list}?,
  attribute alignmentscope {list {("true" | "false") +}}?,
  attribute columnwidth {list {("auto" | length | "fit") +}}?,
  attribute width {"auto" | length}?,
  attribute rowspacing {list {(length) +}}?,
  attribute columnspacing {list {(length) +}}?,
  attribute rowlines {list {linestyle +}}?,
  attribute columnlines {list {linestyle +}}?,
  attribute frame {linestyle}?,
  attribute framespacing {list {length, length}}?,
  attribute equalrows {"true" | "false"}?,
  attribute equalcolumns {"true" | "false"}?,
  attribute displaystyle {"true" | "false"}?,
  attribute side {"left" | "right" | "leftoverlap" | "rightoverlap"}?,
  attribute minlabelspacing {length}?


mlabeledtr = element mlabeledtr {mlabeledtr.attributes, TableCellExpression+}
mlabeledtr.attributes =
  mtr.attributes


mtr = element mtr {mtr.attributes, TableCellExpression*}
mtr.attributes =
  CommonAtt, CommonPresAtt,
  attribute rowalign {"top" | "bottom" | "center" | "baseline" | "axis"}?,
  attribute columnalign {list {columnalignstyle+}}?,
  attribute groupalign {group-alignment-list-list}?


mtd = element mtd {mtd.attributes, ImpliedMrow}
mtd.attributes =
  CommonAtt, CommonPresAtt,
  attribute rowspan {positive-integer}?,
  attribute columnspan {positive-integer}?,
  attribute rowalign {"top" | "bottom" | "center" | "baseline" | "axis"}?,
  attribute columnalign {columnalignstyle}?,
  attribute groupalign {group-alignment-list}?


mstack = element mstack {mstack.attributes, MstackExpression*}
mstack.attributes =
  CommonAtt, CommonPresAtt,
  attribute align {xsd:string {
    pattern ='\s*(top|bottom|center|baseline|axis)(\s+-?[0-9]+)?\s*'}}?,
  attribute stackalign {"left" | "center" | "right" | "decimalpoint"}?,
  attribute charalign {"left" | "center" | "right"}?,
  attribute charspacing {length | "loose" | "medium" | "tight"}?


mlongdiv = element mlongdiv {mlongdiv.attributes, MstackExpression, MstackExpression,
                                                                  MstackExpression+}
mlongdiv.attributes =
  msgroup.attributes,
  attribute longdivstyle {"lefttop" | "stackedrightright" |
          "mediumstackedrightright" | "shortstackedrightright" | "righttop" |
          "left/\right" | "left)(right" | ":right=right" | "stackedleftleft" |
          "stackedleftlinetop"}?


msgroup = element msgroup {msgroup.attributes, MstackExpression*}
```

```
msgroup.attributes =
  CommonAtt, CommonPresAtt,
  attribute position {integer}?,
  attribute shift {integer}?


msrow = element msrow {msrow.attributes, MsrowExpression*}
msrow.attributes =
  CommonAtt, CommonPresAtt,
  attribute position {integer}?


mscarries = element mscarries {mscarries.attributes, (MsrowExpression|mscarry)*}
mscarries.attributes =
  CommonAtt, CommonPresAtt,
  attribute position {integer}?,
  attribute location {"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"}?,
  attribute crossout {list {("none" | "updiagonalstrike" | "downdiagonalstrike" |
                                        "verticalstrike" | "horizontalstrike")*}}?,
  attribute scriptsizemultiplier {number}?


mscarry = element mscarry {mscarry.attributes, MsrowExpression*}
mscarry.attributes =
  CommonAtt, CommonPresAtt,
  attribute location {"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"}?,
  attribute crossout {list {("none" | "updiagonalstrike" | "downdiagonalstrike" |
                                         "verticalstrike" | "horizontalstrike")*}}?


maction = element maction {maction.attributes, MathExpression+}
maction.attributes =
  CommonAtt, CommonPresAtt,
  attribute actiontype {text},
  attribute selection {positive-integer}?
```

A.2.4 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-content.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 1998-2014 W3C (MIT, ERCIM, Keio, Beihang)
#
#     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


default namespace m = "http://www.w3.org/1998/Math/MathML"

ContExp = semantics-contexp | cn | ci | csymbol | apply | bind | share | cerror |
                                                                    cbytes | cs

cn = element cn {cn.attributes, cn.content}
cn.content = text
cn.attributes = CommonAtt, attribute type {"integer" | "real" | "double" |
                                                            "hexdouble"}

semantics-ci = element semantics {semantics.attributes,(ci|semantics-ci),
  (annotation|annotation-xml)*}

semantics-contexp = element semantics {semantics.attributes, ContExp,
  (annotation|annotation-xml)*}

ci = element ci {ci.attributes, ci.content}
ci.attributes = CommonAtt, ci.type?
ci.type = attribute type {"integer" | "rational" | "real" | "complex" |
          "complex-polar" | "complex-cartesian" | "constant" | "function" |
          "vector" | "list" | "set" | "matrix"}
ci.content = text


csymbol = element csymbol {csymbol.attributes, csymbol.content}

SymbolName = xsd:NCName
csymbol.attributes = CommonAtt, cd
csymbol.content = SymbolName

BvarQ = bvar*
bvar = element bvar {CommonAtt, (ci | semantics-ci)}

apply = element apply {CommonAtt, apply.content}
apply.content = ContExp+


bind = element bind {CommonAtt, bind.content}
bind.content = ContExp, bvar*, ContExp

share = element share {CommonAtt, src, empty}

cerror = element cerror {cerror.attributes, csymbol, ContExp*}
cerror.attributes = CommonAtt

cbytes = element cbytes {cbytes.attributes, base64}
cbytes.attributes = CommonAtt
base64 = xsd:base64Binary

cs = element cs {cs.attributes, text}
cs.attributes = CommonAtt

MathExpression |= ContExp
```

A.2.5 The Grammar for Content MathML

The grammar for Content MathML3 builds on the grammar for the Strict Content MathML subset, and can be found at http://www.w3.org/Math/RelaxNG/mathml3/mathml3-content.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 1998-2014 W3C (MIT, ERCIM, Keio, Beihang)
#
#     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
```

```
include "mathml3-strict-content.rnc"{
  cn.content = (text | mglyph | sep | PresentationExpression)*
  cn.attributes = CommonAtt, DefEncAtt, attribute type {text}?, base?

  ci.attributes = CommonAtt, DefEncAtt, ci.type?
  ci.type = attribute type {text}
  ci.content = (text | mglyph | PresentationExpression)*

  csymbol.attributes = CommonAtt, DefEncAtt, attribute type {text}?, cd?
  csymbol.content = (text | mglyph | PresentationExpression)*

  bvar = element bvar {CommonAtt, ((ci | semantics-ci) & degree?)}

  cbytes.attributes = CommonAtt, DefEncAtt

  cs.attributes = CommonAtt, DefEncAtt

  apply.content = ContExp+ | (ContExp, BvarQ, Qualifier*, ContExp*)

  bind.content = apply.content
}

base = attribute base {text}


sep = element sep {empty}
PresentationExpression |= notAllowed


DomainQ = (domainofapplication|condition|interval|(lowlimit, uplimit?))*
domainofapplication = element domainofapplication {ContExp}
condition = element condition {ContExp}
uplimit = element uplimit {ContExp}
lowlimit = element lowlimit {ContExp}

Qualifier = DomainQ|degree|momentabout|logbase
degree = element degree {ContExp}
momentabout = element momentabout {ContExp}
logbase = element logbase {ContExp}

type = attribute type {text}
order = attribute order {"numeric" | "lexicographic"}
closure = attribute closure {text}


ContExp |= piecewise


piecewise = element piecewise {CommonAtt, DefEncAtt,(piece* & otherwise?)}

piece = element piece {CommonAtt, DefEncAtt, ContExp, ContExp}

otherwise = element otherwise {CommonAtt, DefEncAtt, ContExp}


DeprecatedContExp = reln | fn | declare
ContExp |= DeprecatedContExp

reln = element reln {ContExp*}
fn = element fn {ContExp}
```

```
declare = element declare {attribute type {xsd:string}?,
                           attribute scope {xsd:string}?,
                           attribute nargs {xsd:nonNegativeInteger}?,
                           attribute occurrence {"prefix"|"infix"|"function-model"}?,
                           DefEncAtt,
                           ContExp+}


interval.class = interval
ContExp |= interval.class


interval = element interval { CommonAtt, DefEncAtt, closure?, ContExp, ContExp}

unary-functional.class = inverse | ident | domain | codomain | image | ln | log |
                                                                                moment
ContExp |= unary-functional.class


inverse = element inverse { CommonAtt, DefEncAtt, empty}
ident = element ident { CommonAtt, DefEncAtt, empty}
domain = element domain { CommonAtt, DefEncAtt, empty}
codomain = element codomain { CommonAtt, DefEncAtt, empty}
image = element image { CommonAtt, DefEncAtt, empty}
ln = element ln { CommonAtt, DefEncAtt, empty}
log = element log { CommonAtt, DefEncAtt, empty}
moment = element moment { CommonAtt, DefEncAtt, empty}

lambda.class = lambda
ContExp |= lambda.class


lambda = element lambda { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp}

nary-functional.class = compose
ContExp |= nary-functional.class


compose = element compose { CommonAtt, DefEncAtt, empty}

binary-arith.class = quotient | divide | minus | power | rem | root
ContExp |= binary-arith.class


quotient = element quotient { CommonAtt, DefEncAtt, empty}
divide = element divide { CommonAtt, DefEncAtt, empty}
minus = element minus { CommonAtt, DefEncAtt, empty}
power = element power { CommonAtt, DefEncAtt, empty}
rem = element rem { CommonAtt, DefEncAtt, empty}
root = element root { CommonAtt, DefEncAtt, empty}

unary-arith.class = factorial | minus | root | abs | conjugate | arg | real |
                                                     imaginary | floor | ceiling | exp
ContExp |= unary-arith.class


factorial = element factorial { CommonAtt, DefEncAtt, empty}
abs = element abs { CommonAtt, DefEncAtt, empty}
conjugate = element conjugate { CommonAtt, DefEncAtt, empty}
arg = element arg { CommonAtt, DefEncAtt, empty}
real = element real { CommonAtt, DefEncAtt, empty}
```

```
imaginary = element imaginary { CommonAtt, DefEncAtt, empty}
floor = element floor { CommonAtt, DefEncAtt, empty}
ceiling = element ceiling { CommonAtt, DefEncAtt, empty}
exp = element exp { CommonAtt, DefEncAtt, empty}

nary-minmax.class = max | min
ContExp |= nary-minmax.class


max = element max { CommonAtt, DefEncAtt, empty}
min = element min { CommonAtt, DefEncAtt, empty}

nary-arith.class = plus | times | gcd | lcm
ContExp |= nary-arith.class


plus = element plus { CommonAtt, DefEncAtt, empty}
times = element times { CommonAtt, DefEncAtt, empty}
gcd = element gcd { CommonAtt, DefEncAtt, empty}
lcm = element lcm { CommonAtt, DefEncAtt, empty}

nary-logical.class = and | or | xor
ContExp |= nary-logical.class


and = element and { CommonAtt, DefEncAtt, empty}
or = element or { CommonAtt, DefEncAtt, empty}
xor = element xor { CommonAtt, DefEncAtt, empty}

unary-logical.class = not
ContExp |= unary-logical.class


not = element not { CommonAtt, DefEncAtt, empty}

binary-logical.class = implies | equivalent
ContExp |= binary-logical.class


implies = element implies { CommonAtt, DefEncAtt, empty}
equivalent = element equivalent { CommonAtt, DefEncAtt, empty}

quantifier.class = forall | exists
ContExp |= quantifier.class


forall = element forall { CommonAtt, DefEncAtt, empty}
exists = element exists { CommonAtt, DefEncAtt, empty}

nary-reln.class = eq | gt | lt | geq | leq
ContExp |= nary-reln.class


eq = element eq { CommonAtt, DefEncAtt, empty}
gt = element gt { CommonAtt, DefEncAtt, empty}
lt = element lt { CommonAtt, DefEncAtt, empty}
geq = element geq { CommonAtt, DefEncAtt, empty}
leq = element leq { CommonAtt, DefEncAtt, empty}

binary-reln.class = neq | approx | factorof | tendsto
ContExp |= binary-reln.class
```

```
neq = element neq { CommonAtt, DefEncAtt, empty}
approx = element approx { CommonAtt, DefEncAtt, empty}
factorof = element factorof { CommonAtt, DefEncAtt, empty}
tendsto = element tendsto { CommonAtt, DefEncAtt, type?, empty}

int.class = int
ContExp |= int.class


int = element int { CommonAtt, DefEncAtt, empty}

Differential-Operator.class = diff
ContExp |= Differential-Operator.class


diff = element diff { CommonAtt, DefEncAtt, empty}

partialdiff.class = partialdiff
ContExp |= partialdiff.class


partialdiff = element partialdiff { CommonAtt, DefEncAtt, empty}

unary-veccalc.class = divergence | grad | curl | laplacian
ContExp |= unary-veccalc.class


divergence = element divergence { CommonAtt, DefEncAtt, empty}
grad = element grad { CommonAtt, DefEncAtt, empty}
curl = element curl { CommonAtt, DefEncAtt, empty}
laplacian = element laplacian { CommonAtt, DefEncAtt, empty}

nary-setlist-constructor.class = set | \list
ContExp |= nary-setlist-constructor.class


set = element set { CommonAtt, DefEncAtt, type?, BvarQ*, DomainQ*, ContExp*}
\list = element \list { CommonAtt, DefEncAtt, order?, BvarQ*, DomainQ*, ContExp*}

nary-set.class = union | intersect | cartesianproduct
ContExp |= nary-set.class


union = element union { CommonAtt, DefEncAtt, empty}
intersect = element intersect { CommonAtt, DefEncAtt, empty}
cartesianproduct = element cartesianproduct { CommonAtt, DefEncAtt, empty}

binary-set.class = in | notin | notsubset | notprsubset | setdiff
ContExp |= binary-set.class


in = element in { CommonAtt, DefEncAtt, empty}
notin = element notin { CommonAtt, DefEncAtt, empty}
notsubset = element notsubset { CommonAtt, DefEncAtt, empty}
notprsubset = element notprsubset { CommonAtt, DefEncAtt, empty}
setdiff = element setdiff { CommonAtt, DefEncAtt, empty}

nary-set-reln.class = subset | prsubset
ContExp |= nary-set-reln.class
```

```
subset = element subset { CommonAtt, DefEncAtt, empty}
prsubset = element prsubset { CommonAtt, DefEncAtt, empty}

unary-set.class = card
ContExp |= unary-set.class


card = element card { CommonAtt, DefEncAtt, empty}

sum.class = sum
ContExp |= sum.class


sum = element sum { CommonAtt, DefEncAtt, empty}

product.class = product
ContExp |= product.class


product = element product { CommonAtt, DefEncAtt, empty}

limit.class = limit
ContExp |= limit.class


limit = element limit { CommonAtt, DefEncAtt, empty}

unary-elementary.class = sin | cos | tan | sec | csc | cot | sinh | cosh | tanh |
        sech | csch | coth | arcsin | arccos | arctan | arccosh | arccot |
        arccoth | arccsc | arccsch | arcsec | arcsech | arcsinh | arctanh
ContExp |= unary-elementary.class


sin = element sin { CommonAtt, DefEncAtt, empty}
cos = element cos { CommonAtt, DefEncAtt, empty}
tan = element tan { CommonAtt, DefEncAtt, empty}
sec = element sec { CommonAtt, DefEncAtt, empty}
csc = element csc { CommonAtt, DefEncAtt, empty}
cot = element cot { CommonAtt, DefEncAtt, empty}
sinh = element sinh { CommonAtt, DefEncAtt, empty}
cosh = element cosh { CommonAtt, DefEncAtt, empty}
tanh = element tanh { CommonAtt, DefEncAtt, empty}
sech = element sech { CommonAtt, DefEncAtt, empty}
csch = element csch { CommonAtt, DefEncAtt, empty}
coth = element coth { CommonAtt, DefEncAtt, empty}
arcsin = element arcsin { CommonAtt, DefEncAtt, empty}
arccos = element arccos { CommonAtt, DefEncAtt, empty}
arctan = element arctan { CommonAtt, DefEncAtt, empty}
arccosh = element arccosh { CommonAtt, DefEncAtt, empty}
arccot = element arccot { CommonAtt, DefEncAtt, empty}
arccoth = element arccoth { CommonAtt, DefEncAtt, empty}
arccsc = element arccsc { CommonAtt, DefEncAtt, empty}
arccsch = element arccsch { CommonAtt, DefEncAtt, empty}
arcsec = element arcsec { CommonAtt, DefEncAtt, empty}
arcsech = element arcsech { CommonAtt, DefEncAtt, empty}
arcsinh = element arcsinh { CommonAtt, DefEncAtt, empty}
arctanh = element arctanh { CommonAtt, DefEncAtt, empty}

nary-stats.class = mean | sdev | variance | median | mode
```

```
ContExp |= nary-stats.class


mean = element mean { CommonAtt, DefEncAtt, empty}
sdev = element sdev { CommonAtt, DefEncAtt, empty}
variance = element variance { CommonAtt, DefEncAtt, empty}
median = element median { CommonAtt, DefEncAtt, empty}
mode = element mode { CommonAtt, DefEncAtt, empty}

nary-constructor.class = vector | matrix | matrixrow
ContExp |= nary-constructor.class


vector = element vector { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp*}
matrix = element matrix { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp*}
matrixrow = element matrixrow { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp*}

unary-linalg.class = determinant | transpose
ContExp |= unary-linalg.class


determinant = element determinant { CommonAtt, DefEncAtt, empty}
transpose = element transpose { CommonAtt, DefEncAtt, empty}

nary-linalg.class = selector
ContExp |= nary-linalg.class


selector = element selector { CommonAtt, DefEncAtt, empty}

binary-linalg.class = vectorproduct | scalarproduct | outerproduct
ContExp |= binary-linalg.class


vectorproduct = element vectorproduct { CommonAtt, DefEncAtt, empty}
scalarproduct = element scalarproduct { CommonAtt, DefEncAtt, empty}
outerproduct = element outerproduct { CommonAtt, DefEncAtt, empty}

constant-set.class = integers | reals | rationals | naturalnumbers | complexes |
                                                                   primes | emptyset
ContExp |= constant-set.class


integers = element integers { CommonAtt, DefEncAtt, empty}
reals = element reals { CommonAtt, DefEncAtt, empty}
rationals = element rationals { CommonAtt, DefEncAtt, empty}
naturalnumbers = element naturalnumbers { CommonAtt, DefEncAtt, empty}
complexes = element complexes { CommonAtt, DefEncAtt, empty}
primes = element primes { CommonAtt, DefEncAtt, empty}
emptyset = element emptyset { CommonAtt, DefEncAtt, empty}

constant-arith.class = exponentiale | imaginaryi | notanumber | true | false | pi |
                                                                eulergamma | infinity
ContExp |= constant-arith.class


exponentiale = element exponentiale { CommonAtt, DefEncAtt, empty}
imaginaryi = element imaginaryi { CommonAtt, DefEncAtt, empty}
notanumber = element notanumber { CommonAtt, DefEncAtt, empty}
true = element true { CommonAtt, DefEncAtt, empty}
false = element false { CommonAtt, DefEncAtt, empty}
```

```
pi = element pi { CommonAtt, DefEncAtt, empty}
eulergamma = element eulergamma { CommonAtt, DefEncAtt, empty}
infinity = element infinity { CommonAtt, DefEncAtt, empty}
```

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 content models for inline and block-level elements.

Customizing the MathML3 schema so that we can restrict the content of `annotation-xml` elements is similarly simple, for example:

```
# A RelaxNG Schema for MathML with OpenMath3 annotations
omobj = external "openmath3.rnc"
include "mathml3.rnc" {anotation-xml.model = omobj}
```

The MathML3 schema is organized so that subsetting to one of the sublanguages specified here is easy. To include strict content MathML3 in a schema just include

```
include "mathml3-common.rnc"
include "mathml3-strict-content.rnc"
```

instead of `include mathml3.rnc`.

For details about RelaxNG grammars and modularization see [RELAX-NG] or [RelaxNGBook].

A.3 Using the MathML DTD

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 MathML DTD uses the strategy outlined in [Modularization] of making 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.

A.3.2 Attribute values in the MathML DTD

Note that the DTD is far more permissive than the Relax NG schema, many constraints on attribute values may not be enforced using DTD syntax. For example, many attributes expect numerical values or specific micro-syntax, but are declared as CDATA (arbitrary string) in the DTD. 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 Handling of Errors 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 Characters, Entities and Fonts first, then Chapter 3 Presentation Markup, Chapter 4 Content Markup, then Section 2.1 MathML Syntax and Grammar, and then other parts of the text.)

A.3.3 DOCTYPE declaration for MathML

MathML documents should be validated using the XML DTD for MathML, http://www.w3.org/Math/DTD/mathml3/mathml3.dtd.

Documents using this DTD should contain a DOCTYPE declaration of the form:

```
<!DOCTYPE math
    PUBLIC "-//W3C//DTD MathML 3.0//EN"
           "http://www.w3.org/Math/DTD/mathml3/mathml3.dtd">
```

The URI may be changed to that of a local copy of the DTD if required.

A.4 Using the MathML XML Schema

MathML fragments can be validated using the XML Schema for MathML, located at http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd. The provided schema has been mechanically generated from the Relax NG schema, it omits some constraints that can not be enforced using XSD syntax.

A.4.1 Associating the MathML schema with MathML fragments

Similarly to the DOCTYPE declaration used in documents, it is possible to link a MathML fragment to the XML Schema, as shown below.

```
<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/1998/Math/MathML
            http://www.w3.org/Math/XMLSchema/mathml3/mathml3.xsd">
...
</mml:math>
```

The `xsi:schemaLocation` attribute belongs to the XML Schema instance namespace, defined in [XMLSchemas]. The value of the attribute is a pair of URIs: the first is the MathML namespace URI and the second is the location of the schema for that namespace. The second URI may be changed to that of a local copy if required.

As the XML Schema specification indicates, the use of the `schemaLocation` attribute is only a hint for schema processors: validation by a MathML-aware processor can be performed without requiring that the `schemaLocation` attribute be declared in the instance. Moreover, a processor can even ignore or override the specified schema URI, if it is directed to.

A.5 Parsing MathML in XHTML

As noted in Section 6.4.1 Mixing MathML and XHTML The schema used for XHTML+MathML documents extends the content model of `mtext` to allow inline (phrasing) html elements. The extension used by the W3C validator uses the RelaxNG schema, however DTD and XSD versions of the grammar may be similarly extended.

A.6 Parsing MathML in HTML

See Section 6.4.3 Mixing MathML and HTML for more details of the parsing of MathML in HTML.

B Media Types Registrations

This normative appendix registers three media types for MathML, "application/mathml+xml", "application/mathml-presentation+xml" and "application/mathml-content+xml", in conformance with [RFC4288] and W3CRegMedia. The media-types have been approved by IESG for registration with IANA as announced by IETF and are visible on the list of application media-types.

B.1 Selection of Media Types for MathML Instances

MathML contains two distinct vocabularies. Presentation markup is for encoding visual presentation, and consists of the elements defined in Chapter 3 Presentation Markup. Content markup is for encoding mathematical meaning, and consists of the elements defined in Chapter 4 Content Markup. In addition, both the presentation and content vocabularies contain the `math`, `semantics`, `annotation` and `annotation-xml` elements. The MathML media types should be used as follows:

"application/mathml-presentation+xml"
: MathML instances that consist solely of presentation markup.

"application/mathml-content+xml"
: MathML instances that consist solely of content markup.

"application/mathml+xml"
: Any valid MathML instance. Must be used for MathML instances that are a mix of presentation and content markup, or where the composition of an instance is not known or cannot be guaranteed.

Some MathML applications may import and export only one of these two vocabularies, while others may produce and consume each in a different way, and still others may process both without any distinction between the two. Internally, many MathML processors favor one vocabulary, and support the other vocabulary via conversion if at all. For example, computational software typically favors content markup while typesetting software generally favors presentation markup. By using separate media types for MathML instances consisting solely of presentation or solely of content markup, such processors can conduct negotiation for MathML representations in the preferred vocabulary. For example, consider two web services offering mathematical computation services such as a spreadsheet and a computer algebra system. Internally both prefer content markup, but by default, both generate presentation markup for output. In the absence of media type negotiation, a likely scenario for an exchange between two systems involves two conversions, content to presentation and back again. With negotiation, the conversions are eliminated. Similarly, a client with a MathML instance in one of the vocabularies might seek a web service that preferred that vocabulary.

MathML is commonly used in compound document settings, e.g. within HTML, where content is drawn from a variety of sources, and processed with multiple tools. In these cases, the composition of MathML expressions generally is not known or at least cannot be guaranteed by a user agent. Consequently, the "application/mathml+xml" type should be used, as it may be applied to any valid MathML expression. Since most applications involve data from untrusted sources, "application/mathml+xml" will commonly be appropriate to use as a default type, and all MathML processors are encouraged to accept it as a fallback to the more specific formats.

The media types described here may be applied to instances of all versions of MathML up to and including MathML 3. MathML instances do not contain version numbers, so processors and producers must follow the normative backward compatibility behavior described in this specification.

B.2 Media type for Generic MathML

This registration has been submitted to community review and has been approved by IESG for registration with IANA.

Type name

`application`

Subtype name

`mathml+xml`

Required parameters

None

Optional parameters

Same as `charset` parameter of application/xml as specified in [RFC3023]

Encoding considerations

The encoding considerations of application/xml as specified in [RFC3023] apply.

Security considerations

As with other XML types and as noted in [RFC3023] section 10, repeated expansion of maliciously constructed XML entities can be used to consume large amounts of memory, which may cause XML processors in constrained environments to fail.

Several MathML elements may cause arbitrary URIs to be referenced. In this case, the security issues of [RFC3986], section 7, should be considered.

In common with HTML, MathML documents may reference external media such as images, style sheets, and scripting languages. Scripting languages are executable content. In this case, the security considerations in the Media Type registrations for those formats shall apply. Similarly, MathML annotation elements may contain content intended for execution or processing. In the case where the processor recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, additional security issues potentially arise. Since the normative semantics of this specification do not require processing of annotation elements, such issues fall outside the domain of this registration document.

MathML may be used to describe mathematical expressions intended for evaluation in computing systems. Because of the nature of mathematics, a seemingly innocuous expression may lead to a computation which does not terminate or is impractically large. This introduces the risk that computational processors in constrained environments may fail.

In addition, because of the extensibility features for MathML and of XML in general, it is possible that "application/mathml+xml" may describe content that has security implications beyond those described here. However, if the processor follows only the normative semantics of this specification, this content will be outside the MathML namespace and shall be ignored.

Interoperability considerations

This specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements and attributes, both in the MathML namespace and in other namespaces.

Because MathML is extensible, conformant "application/mathml+xml" processors must expect that content received is well-formed XML, but it cannot be guaranteed that the content is valid to a particular DTD or Schema or that the processor will recognize all of the elements and attributes in the document.

MathML instances do not contain version numbers, so processors and producers must follow the normative backward compatibility behavior described in this specification.

In computational contexts, the result of evaluating a MathML expression is system-specific, and is not guaranteed to be interoperable between systems.

Published specification

This media type registration is extracted from Appendix B of the Mathematical Markup Language (MathML) Version 3. specification.

Applications that use this media type

Web browsers, rendering engines, formula editors, typesetting software, search robots, computing systems.

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

Paul Libbrecht (member-math@w3.org). See the W3C Math Working Group home page for more information.

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 Presentation MathML

This registration has been submitted to community review and has been approved by IESG for registration with IANA.

Type name

`application`

Subtype name

`mathml-presentation+xml`

Required parameters

None

Optional parameters

Same as `charset` parameter of application/xml as specified in [RFC3023]

Encoding considerations

The considerations of application/xml as specified in [RFC3023] apply.

Security considerations

As with other XML types and as noted in [RFC3023] section 10, repeated expansion of maliciously constructed XML entities can be used to consume large amounts of memory, which may cause XML processors in constrained environments to fail.

Several MathML elements may cause arbitrary URIs to be referenced. In this case, the security issues of [RFC3986], section 7, should be considered.

In common with HTML, MathML documents may reference external media such as images, style sheets, and scripting languages. Scripting languages are executable content. In this case, the security considerations in the Media Type registrations for those formats shall apply. Similarly, MathML annotation elements may contain content intended for execution or processing. In the case where the processor recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, additional security issues potentially arise. Since the normative semantics of this specification do not require processing of annotation elements, such issues fall outside the domain of this registration document.

MathML may be used to describe mathematical expressions intended for evaluation in computing systems. Because of the nature of mathematics, a seemingly innocuous expression may lead to a computation which does not terminate or is impractically large. This introduces the risk that computational processors in constrained environments may fail.

In addition, because of the extensibility features for MathML and of XML in general, it is possible that "application/mathml-presentation+xml" may describe content that has security implications beyond those described here. However, if the processor follows only the normative semantics of this specification, this content will be outside the MathML namespace and shall be ignored.

Interoperability considerations

This specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements and attributes, both in the MathML namespace and in other namespaces.

Because MathML is extensible, conformant "application/mathml-presentation+xml" processors must expect that content received is well-formed XML, but it cannot be guaranteed that the content is valid to a particular DTD or Schema or that the processor will recognize all of the elements and attributes in the document.

MathML instances do not contain version numbers, so processors and producers must follow the normative backward compatibility behavior described in this specification.

In computational contexts, the result of evaluating a MathML expression is system-specific, and is not guaranteed to be interoperable between systems.

This specification does not record a file extension for the media type "application/mathml-presentation+xml" because we expect tools processing files with MathML inside to have sufficient information with the generic media-type (application/mathml+xml) while other content negotiation forms will take advantage of specific media-types.

Published specification

This media type registration is extracted from Appendix B of the Mathematical Markup Language (MathML) Version 3. specification.

Applications that use this media type

Web browsers, rendering engines, formula editors, typesetting software, search robots, computing systems.

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

Paul Libbrecht (member-math@w3.org). See the W3C Math Working Group home page for more information.

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.4 Media type for Content MathML

This registration has been submitted to community review and has been approved by IESG for registration with IANA.

Type name

`application`

Subtype name

`mathml-content+xml`

Required parameters

None

Optional parameters

Same as `charset` parameter of application/xml as specified in [RFC3023]

Encoding considerations

The encoding considerations of application/xml as specified in [RFC3023] apply.

Security considerations

As with other XML types and as noted in [RFC3023] section 10, repeated expansion of maliciously constructed XML entities can be used to consume large amounts of memory, which may cause XML processors in constrained environments to fail.

Several MathML elements may cause arbitrary URIs to be referenced. In this case, the security issues of [RFC3986], section 7, should be considered.

In common with HTML, MathML documents may reference external media such as images, style sheets, and scripting languages. Scripting languages are executable content. In this case, the security considerations in the Media Type registrations for those formats shall apply. Similarly, MathML annotation elements may contain content intended for execution or processing. In the case where the processor recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, additional security issues potentially arise. Since the normative semantics of this specification do not require processing of annotation elements, such issues fall outside the domain of this registration document.

MathML may be used to describe mathematical expressions intended for evaluation in computing systems. Because of the nature of mathematics, a seemingly innocuous expression may lead to a computation which does not terminate or is impractically large. This introduces the risk that computational processors in constrained environments may fail.

In addition, because of the extensibility features for MathML and of XML in general, it is possible that "application/mathml-content+xml" may describe content that has security implications beyond those described here. However, if the processor follows only the normative semantics of this specification, this content will be outside the MathML namespace and shall be ignored.

Interoperability considerations

This specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements and attributes, both in the MathML namespace and in other namespaces.

Because MathML is extensible, conformant "application/mathml-content+xml" processors must expect that content received is well-formed XML, but it cannot be guaranteed that the content is valid to a particular DTD or Schema or that the processor will recognize all of the elements and attributes in the document.

MathML instances do not contain version numbers, so processors and producers must follow the normative backward compatibility behavior described in this specification.

In computational contexts, the result of evaluating a MathML expression is system-specific, and is not guaranteed to be interoperable between systems.

This specification does not record a file extension for the media type "application/mathml-content+xml" because we expect tools processing files with MathML inside to have sufficient information with the generic media-type (application/mathml+xml) while other content negotiation forms will take advantage of specific media-types.

Published specification

This media type registration is extracted from Appendix B of the Mathematical Markup Language (MathML) Version 3. specification.

Applications that use this media type

Formula editors, search robots, computing systems.

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

Paul Libbrecht (member-math@w3.org). See the W3C Math Working Group home page for more information.

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.

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 3.2.5.7.2 Default value of the form attribute.

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 code point and Unicode Name) and the value of the form attribute. The fourth column gives the **priority** which as described in Section 3.3.1 Horizontally Group Sub-Expressions `<mrow>`), 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 attributes such as `lspace` are given explicitly in the following columns. Boolean valued attributes such as `stretchy` are specified together in the final **Properties** column by listing the attribute name if its value should be set to **true** by the dictionary. Finally some rarely used non-boolean properties are shown in the Properties column together with a value, for example `linebreakstyle`=after.

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 Operator, Fence, Separator or Accent `<mo>`.

(	(	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; Such whitespace will be ignored by a MathML system, as described in Section 2.1.7 Collapsing Whitespace in Input.

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 7, they are given numerically in order to save space in the table, the values should be taken as referring to the named mathematical spaces, as follows.

Table Entry	Named Space	Default Length
0		0 em
1	veryverythinmathspace	1/18 em
2	verythinmathspace	2/18 em
3	thinmathspace	3/18 em
4	mediummathspace	4/18 em
5	thickmathspace	5/18 em
6	verythickmathspace	6/18 em
7	veryverythickmathspace	7/18 em

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 Style Change `<mstyle>`), as is the case in TEX.

C.4 Operator dictionary entries

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2018;	‘	left single quotation mark	prefix	10	0	0	fence
&#x2019;	’	right single quotation mark	postfix	10	0	0	fence
&#x201C;	“	left double quotation mark	prefix	10	0	0	fence
&#x201D;	”	right double quotation mark	postfix	10	0	0	fence
(	(	left parenthesis	prefix	20	0	0	fence, stretchy, symmetric
)	)	right parenthesis	postfix	20	0	0	fence, stretchy, symmetric
[	[	left square bracket	prefix	20	0	0	fence, stretchy, symmetric
]	]	right square bracket	postfix	20	0	0	fence, stretchy, symmetric
{	{	left curly bracket	prefix	20	0	0	fence, stretchy, symmetric
|	|	vertical line	prefix	20	0	0	fence, stretchy, symmetric
|	|	vertical line	postfix	20	0	0	fence, stretchy, symmetric
||	||	multiple character operator: ||	prefix	20	0	0	fence, stretchy, symmetric
||	||	multiple character operator: ||	postfix	20	0	0	fence, stretchy, symmetric
|||	|||	multiple character operator: |||	prefix	20	0	0	fence, stretchy, symmetric
|||	|||	multiple character operator: |||	postfix	20	0	0	fence, stretchy, symmetric
}	}	right curly bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x2016;	‖	double vertical line	prefix	20	0	0	fence, stretchy

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2016;	‖	double vertical line	postfix	20	0	0	fence, stretchy
&#x2308;	⌈	left ceiling	prefix	20	0	0	fence, stretchy, symmetric
&#x2309;	⌉	right ceiling	postfix	20	0	0	fence, stretchy, symmetric
&#x230A;	⌊	left floor	prefix	20	0	0	fence, stretchy, symmetric
&#x230B;	⌋	right floor	postfix	20	0	0	fence, stretchy, symmetric
&#x2329;	〈	left-pointing angle bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x232A;	〉	right-pointing angle bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x2772;	❲	light left tortoise shell bracket ornament	prefix	20	0	0	fence, stretchy, symmetric
&#x2773;	❳	light right tortoise shell bracket ornament	postfix	20	0	0	fence, stretchy, symmetric
&#x27E6;	⟦	mathematical left white square bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x27E7;	⟧	mathematical right white square bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x27E8;	⟨	mathematical left angle bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x27E9;	⟩	mathematical right angle bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x27EA;	⟪	mathematical left double angle bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x27EB;	⟫	mathematical right double angle bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x27EC;	⟬	mathematical left white tortoise shell bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x27ED;	⟭	mathematical right white tortoise shell bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x27EE;	⟮	mathematical left flattened parenthesis	prefix	20	0	0	fence, stretchy, symmetric
&#x27EF;	⟯	mathematical right flattened parenthesis	postfix	20	0	0	fence, stretchy, symmetric
&#x2980;	⦀	triple vertical bar delimiter	prefix	20	0	0	fence, stretchy
&#x2980;	⦀	triple vertical bar delimiter	postfix	20	0	0	fence, stretchy
&#x2983;	⦃	left white curly bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x2984;	⦄	right white curly bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x2985;	⦅	left white parenthesis	prefix	20	0	0	fence, stretchy, symmetric
&#x2986;	⦆	right white parenthesis	postfix	20	0	0	fence, stretchy, symmetric
&#x2987;	⦇	z notation left image bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x2988;	⦈	z notation right image bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x2989;	⦉	z notation left binding bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x298A;	⦊	z notation right binding bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x298B;	⦋	left square bracket with underbar	prefix	20	0	0	fence, stretchy, symmetric
&#x298C;	⦌	right square bracket with underbar	postfix	20	0	0	fence, stretchy, symmetric
&#x298D;	⦍	left square bracket with tick in top corner	prefix	20	0	0	fence, stretchy, symmetric
&#x298E;	⦎	right square bracket with tick in bottom corner	postfix	20	0	0	fence, stretchy, symmetric
&#x298F;	⦏	left square bracket with tick in bottom corner	prefix	20	0	0	fence, stretchy, symmetric
&#x2990;	⦐	right square bracket with tick in top corner	postfix	20	0	0	fence, stretchy, symmetric
&#x2991;	⦑	left angle bracket with dot	prefix	20	0	0	fence, stretchy, symmetric
&#x2992;	⦒	right angle bracket with dot	postfix	20	0	0	fence, stretchy, symmetric
&#x2993;	⦓	left arc less-than bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x2994;	⦔	right arc greater-than bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x2995;	⦕	double left arc greater-than bracket	prefix	20	0	0	fence, stretchy, symmetric

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2996;	⦖	double right arc less-than bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x2997;	⦗	left black tortoise shell bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x2998;	⦘	right black tortoise shell bracket	postfix	20	0	0	fence, stretchy, symmetric
&#x29FC;	⧼	left-pointing curved angle bracket	prefix	20	0	0	fence, stretchy, symmetric
&#x29FD;	⧽	right-pointing curved angle bracket	postfix	20	0	0	fence, stretchy, symmetric
;	;	semicolon	infix	30	0	3	separator, linebreakstyle=after
,	,	comma	infix	40	0	3	separator, linebreakstyle=after
&#x2063;		invisible separator	infix	40	0	0	separator, linebreakstyle=after
&#x2234;	∴	therefore	infix	70	5	5	
&#x2235;	∵	because	infix	70	5	5	
->	->	multiple character operator: ->	infix	90	5	5	
..	..	multiple character operator: ..	postfix	100	0	0	
...	...	multiple character operator: ...	postfix	100	0	0	
:	:	colon	infix	100	1	2	
&#x3F6;	϶	greek reversed lunate epsilon symbol	infix	110	5	5	
&#x2026;	…	horizontal ellipsis	infix	150	0	0	
&#x22EE;	⋮	vertical ellipsis	infix	150	5	5	
&#x22EF;	⋯	midline horizontal ellipsis	infix	150	0	0	
&#x22F1;	⋱	down right diagonal ellipsis	infix	150	5	5	
&#x220B;	∋	contains as member	infix	160	5	5	
&#x22A2;	⊢	right tack	infix	170	5	5	
&#x22A3;	⊣	left tack	infix	170	5	5	
&#x22A4;	⊤	down tack	infix	170	5	5	
&#x22A8;	⊨	true	infix	170	5	5	
&#x22A9;	⊩	forces	infix	170	5	5	
&#x22AC;	⊬	does not prove	infix	170	5	5	
&#x22AD;	⊭	not true	infix	170	5	5	
&#x22AE;	⊮	does not force	infix	170	5	5	
&#x22AF;	⊯	negated double vertical bar double right turnstile	infix	170	5	5	
&#x2228;	∨	logical or	infix	190	4	4	
&amp;&amp;	&&	multiple character operator: &&	infix	200	4	4	
&#x2227;	∧	logical and	infix	200	4	4	
&#x2200;	∀	for all	prefix	230	2	1	
&#x2203;	∃	there exists	prefix	230	2	1	
&#x2204;	∄	there does not exist	prefix	230	2	1	
&#x2201;	∁	complement	infix	240	1	2	
&#x2208;	∈	element of	infix	240	5	5	
&#x2209;	∉	not an element of	infix	240	5	5	
&#x220C;	∌	does not contain as member	infix	240	5	5	
&#x2282;	⊂	subset of	infix	240	5	5	
&#x2282;&#x20D2;	⊂⃒	subset of with vertical line	infix	240	5	5	
&#x2283;	⊃	superset of	infix	240	5	5	
&#x2283;&#x20D2;	⊃⃒	superset of with vertical line	infix	240	5	5	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2284;	⊄	not a subset of	infix	240	5	5	
&#x2285;	⊅	not a superset of	infix	240	5	5	
&#x2286;	⊆	subset of or equal to	infix	240	5	5	
&#x2287;	⊇	superset of or equal to	infix	240	5	5	
&#x2288;	⊈	neither a subset of nor equal to	infix	240	5	5	
&#x2289;	⊉	neither a superset of nor equal to	infix	240	5	5	
&#x228A;	⊊	subset of with not equal to	infix	240	5	5	
&#x228B;	⊋	superset of with not equal to	infix	240	5	5	
&lt;=	<=	multiple character operator: <=	infix	241	5	5	
&#x2264;	≤	less-than or equal to	infix	241	5	5	
&#x2265;	≥	greater-than or equal to	infix	242	5	5	
>	>	greater-than sign	infix	243	5	5	
>=	>=	multiple character operator: >=	infix	243	5	5	
&#x226F;	≯	not greater-than	infix	244	5	5	
&lt;	<	less-than sign	infix	245	5	5	
&#x226E;	≮	not less-than	infix	246	5	5	
&#x2248;	≈	almost equal to	infix	247	5	5	
&#x223C;	∼	tilde operator	infix	250	5	5	
&#x2249;	≉	not almost equal to	infix	250	5	5	
&#x2262;	≢	not identical to	infix	252	5	5	
&#x2260;	≠	not equal to	infix	255	5	5	
!=	!=	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	
&#x221D;	∝	proportional to	infix	260	5	5	
&#x2224;	∤	does not divide	infix	260	5	5	
&#x2225;	∥	parallel to	infix	260	5	5	
&#x2226;	∦	not parallel to	infix	260	5	5	
&#x2241;	≁	not tilde	infix	260	5	5	
&#x2243;	≃	asymptotically equal to	infix	260	5	5	
&#x2244;	≄	not asymptotically equal to	infix	260	5	5	
&#x2245;	≅	approximately equal to	infix	260	5	5	
&#x2246;	≆	approximately but not actually equal to	infix	260	5	5	
&#x2247;	≇	neither approximately nor actually equal to	infix	260	5	5	
&#x224D;	≍	equivalent to	infix	260	5	5	
&#x2254;	≔	colon equals	infix	260	5	5	
&#x2257;	≗	ring equal to	infix	260	5	5	
&#x2259;	≙	estimates	infix	260	5	5	
&#x225A;	≚	equiangular to	infix	260	5	5	
&#x225B;	≛	star equals	infix	260	5	5	
&#x225C;	≜	delta equal to	infix	260	5	5	
&#x225F;	≟	questioned equal to	infix	260	5	5	
&#x2261;	≡	identical to	infix	260	5	5	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2268;	≨	less-than but not equal to	infix	260	5	5	
&#x2269;	≩	greater-than but not equal to	infix	260	5	5	
&#x226A;	≪	much less-than	infix	260	5	5	
&#x226A;&#x338;	≪̸	much less than with slash	infix	260	5	5	
&#x226B;	≫	much greater-than	infix	260	5	5	
&#x226B;&#x338;	≫̸	much greater than with slash	infix	260	5	5	
&#x226D;	≭	not equivalent to	infix	260	5	5	
&#x2270;	≰	neither less-than nor equal to	infix	260	5	5	
&#x2271;	≱	neither greater-than nor equal to	infix	260	5	5	
&#x227A;	≺	precedes	infix	260	5	5	
&#x227B;	≻	succeeds	infix	260	5	5	
&#x227C;	≼	precedes or equal to	infix	260	5	5	
&#x227D;	≽	succeeds or equal to	infix	260	5	5	
&#x2280;	⊀	does not precede	infix	260	5	5	
&#x2281;	⊁	does not succeed	infix	260	5	5	
&#x22A5;	⊥	up tack	infix	260	5	5	
&#x22B4;	⊴	normal subgroup of or equal to	infix	260	5	5	
&#x22B5;	⊵	contains as normal subgroup or equal to	infix	260	5	5	
&#x22C9;	⋉	left normal factor semidirect product	infix	260	4	4	
&#x22CA;	⋊	right normal factor semidirect product	infix	260	4	4	
&#x22CB;	⋋	left semidirect product	infix	260	4	4	
&#x22CC;	⋌	right semidirect product	infix	260	4	4	
&#x22D4;	⋔	pitchfork	infix	260	5	5	
&#x22D6;	⋖	less-than with dot	infix	260	5	5	
&#x22D7;	⋗	greater-than with dot	infix	260	5	5	
&#x22D8;	⋘	very much less-than	infix	260	5	5	
&#x22D9;	⋙	very much greater-than	infix	260	5	5	
&#x22EA;	⋪	not normal subgroup of	infix	260	5	5	
&#x22EB;	⋫	does not contain as normal subgroup	infix	260	5	5	
&#x22EC;	⋬	not normal subgroup of or equal to	infix	260	5	5	
&#x22ED;	⋭	does not contain as normal subgroup or equal	infix	260	5	5	
&#x25A0;	■	black square	infix	260	3	3	
&#x25A1;	□	white square	infix	260	3	3	
&#x25AA;	▪	black small square	infix	260	3	3	
&#x25AB;	▫	white small square	infix	260	3	3	
&#x25AD;	▭	white rectangle	infix	260	3	3	
&#x25AE;	▮	black vertical rectangle	infix	260	3	3	
&#x25AF;	▯	white vertical rectangle	infix	260	3	3	
&#x25B0;	▰	black parallelogram	infix	260	3	3	
&#x25B1;	▱	white parallelogram	infix	260	3	3	
&#x25B3;	△	white up-pointing triangle	infix	260	4	4	
&#x25B4;	▴	black up-pointing small triangle	infix	260	4	4	
&#x25B5;	▵	white up-pointing small triangle	infix	260	4	4	
&#x25B6;	▶	black right-pointing triangle	infix	260	4	4	
&#x25B7;	▷	white right-pointing triangle	infix	260	4	4	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x25B8;	▸	black right-pointing small triangle	infix	260	4	4	
&#x25B9;	▹	white right-pointing small triangle	infix	260	4	4	
&#x25BC;	▼	black down-pointing triangle	infix	260	4	4	
&#x25BD;	▽	white down-pointing triangle	infix	260	4	4	
&#x25BE;	▾	black down-pointing small triangle	infix	260	4	4	
&#x25BF;	▿	white down-pointing small triangle	infix	260	4	4	
&#x25C0;	◀	black left-pointing triangle	infix	260	4	4	
&#x25C1;	◁	white left-pointing triangle	infix	260	4	4	
&#x25C2;	◂	black left-pointing small triangle	infix	260	4	4	
&#x25C3;	◃	white left-pointing small triangle	infix	260	4	4	
&#x25C4;	◄	black left-pointing pointer	infix	260	4	4	
&#x25C5;	◅	white left-pointing pointer	infix	260	4	4	
&#x25C6;	◆	black diamond	infix	260	4	4	
&#x25C7;	◇	white diamond	infix	260	4	4	
&#x25C8;	◈	white diamond containing black small diamond	infix	260	4	4	
&#x25C9;	◉	fisheye	infix	260	4	4	
&#x25CC;	◌	dotted circle	infix	260	4	4	
&#x25CD;	◍	circle with vertical fill	infix	260	4	4	
&#x25CE;	◎	bullseye	infix	260	4	4	
&#x25CF;	●	black circle	infix	260	4	4	
&#x25D6;	◖	left half black circle	infix	260	4	4	
&#x25D7;	◗	right half black circle	infix	260	4	4	
&#x25E6;	◦	white bullet	infix	260	4	4	
&#x29C0;	⧀	circled less-than	infix	260	5	5	
&#x29C1;	⧁	circled greater-than	infix	260	5	5	
&#x29E3;	⧣	equals sign and slanted parallel	infix	260	5	5	
&#x29E4;	⧤	equals sign and slanted parallel with tilde above	infix	260	5	5	
&#x29E5;	⧥	identical to and slanted parallel	infix	260	5	5	
&#x29E6;	⧦	gleich stark	infix	260	5	5	
&#x29F3;	⧳	error-barred black circle	infix	260	3	3	
&#x2A87;	⪇	less-than and single-line not equal to	infix	260	5	5	
&#x2A88;	⪈	greater-than and single-line not equal to	infix	260	5	5	
&#x2AAF;	⪯	precedes above single-line equals sign	infix	260	5	5	
&#x2AAF;&#x338;	⪯̸	precedes above single-line equals sign with slash	infix	260	5	5	
&#x2AB0;	⪰	succeeds above single-line equals sign	infix	260	5	5	
&#x2AB0;&#x338;	⪰̸	succeeds above single-line equals sign with slash	infix	260	5	5	
&#x2044;	⁄	fraction slash	infix	265	4	4	stretchy
&#x2206;	∆	increment	infix	265	3	3	
&#x220A;	∊	small element of	infix	265	5	5	
&#x220D;	∍	small contains as member	infix	265	5	5	
&#x220E;	∎	end of proof	infix	265	3	3	
&#x2215;	∕	division slash	infix	265	4	4	stretchy

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2217;	∗	asterisk operator	infix	265	4	4	
&#x2218;	∘	ring operator	infix	265	4	4	
&#x2219;	∙	bullet operator	infix	265	4	4	
&#x221F;	∟	right angle	infix	265	5	5	
&#x2223;	∣	divides	infix	265	5	5	
&#x2236;	∶	ratio	infix	265	5	5	
&#x2237;	∷	proportion	infix	265	5	5	
&#x2238;	∸	dot minus	infix	265	4	4	
&#x2239;	∹	excess	infix	265	5	5	
&#x223A;	∺	geometric proportion	infix	265	4	4	
&#x223B;	∻	homothetic	infix	265	5	5	
&#x223D;	∽	reversed tilde	infix	265	5	5	
&#x223D;&#x331;	∽̱	reversed tilde with underline	infix	265	3	3	
&#x223E;	∾	inverted lazy s	infix	265	5	5	
&#x223F;	∿	sine wave	infix	265	3	3	
&#x2242;	≂	minus tilde	infix	265	5	5	
&#x2242;&#x338;	≂̸	minus tilde with slash	infix	265	5	5	
&#x224A;	≊	almost equal or equal to	infix	265	5	5	
&#x224B;	≋	triple tilde	infix	265	5	5	
&#x224C;	≌	all equal to	infix	265	5	5	
&#x224E;	≎	geometrically equivalent to	infix	265	5	5	
&#x224E;&#x338;	≎̸	geometrically equivalent to with slash	infix	265	5	5	
&#x224F;	≏	difference between	infix	265	5	5	
&#x224F;&#x338;	≏̸	difference between with slash	infix	265	5	5	
&#x2250;	≐	approaches the limit	infix	265	5	5	
&#x2251;	≑	geometrically equal to	infix	265	5	5	
&#x2252;	≒	approximately equal to or the image of	infix	265	5	5	
&#x2253;	≓	image of or approximately equal to	infix	265	5	5	
&#x2255;	≕	equals colon	infix	265	5	5	
&#x2256;	≖	ring in equal to	infix	265	5	5	
&#x2258;	≘	corresponds to	infix	265	5	5	
&#x225D;	≝	equal to by definition	infix	265	5	5	
&#x225E;	≞	measured by	infix	265	5	5	
&#x2263;	≣	strictly equivalent to	infix	265	5	5	
&#x2266;	≦	less-than over equal to	infix	265	5	5	
&#x2266;&#x338;	≦̸	less-than over equal to with slash	infix	265	5	5	
&#x2267;	≧	greater-than over equal to	infix	265	5	5	
&#x226C;	≬	between	infix	265	5	5	
&#x2272;	≲	less-than or equivalent to	infix	265	5	5	
&#x2273;	≳	greater-than or equivalent to	infix	265	5	5	
&#x2274;	≴	neither less-than nor equivalent to	infix	265	5	5	
&#x2275;	≵	neither greater-than nor equivalent to	infix	265	5	5	
&#x2276;	≶	less-than or greater-than	infix	265	5	5	
&#x2277;	≷	greater-than or less-than	infix	265	5	5	
&#x2278;	≸	neither less-than nor greater-than	infix	265	5	5	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2279;	≹	neither greater-than nor less-than	infix	265	5	5	
&#x227E;	≾	precedes or equivalent to	infix	265	5	5	
&#x227F;	≿	succeeds or equivalent to	infix	265	5	5	
&#x227F;&#x338;	≿̸	succeeds or equivalent to with slash	infix	265	5	5	
&#x228C;	⊌	multiset	infix	265	4	4	
&#x228D;	⊍	multiset multiplication	infix	265	4	4	
&#x228E;	⊎	multiset union	infix	265	4	4	
&#x228F;	⊏	square image of	infix	265	5	5	
&#x228F;&#x338;	⊏̸	square image of with slash	infix	265	5	5	
&#x2290;	⊐	square original of	infix	265	5	5	
&#x2290;&#x338;	⊐̸	square original of with slash	infix	265	5	5	
&#x2291;	⊑	square image of or equal to	infix	265	5	5	
&#x2292;	⊒	square original of or equal to	infix	265	5	5	
&#x2293;	⊓	square cap	infix	265	4	4	
&#x2294;	⊔	square cup	infix	265	4	4	
&#x229A;	⊚	circled ring operator	infix	265	4	4	
&#x229B;	⊛	circled asterisk operator	infix	265	4	4	
&#x229C;	⊜	circled equals	infix	265	4	4	
&#x229D;	⊝	circled dash	infix	265	4	4	
&#x22A6;	⊦	assertion	infix	265	5	5	
&#x22A7;	⊧	models	infix	265	5	5	
&#x22AA;	⊪	triple vertical bar right turnstile	infix	265	5	5	
&#x22AB;	⊫	double vertical bar double right turnstile	infix	265	5	5	
&#x22B0;	⊰	precedes under relation	infix	265	5	5	
&#x22B1;	⊱	succeeds under relation	infix	265	5	5	
&#x22B2;	⊲	normal subgroup of	infix	265	5	5	
&#x22B3;	⊳	contains as normal subgroup	infix	265	5	5	
&#x22B6;	⊶	original of	infix	265	5	5	
&#x22B7;	⊷	image of	infix	265	5	5	
&#x22B9;	⊹	hermitian conjugate matrix	infix	265	5	5	
&#x22BA;	⊺	intercalate	infix	265	4	4	
&#x22BB;	⊻	xor	infix	265	4	4	
&#x22BC;	⊼	nand	infix	265	4	4	
&#x22BD;	⊽	nor	infix	265	4	4	
&#x22BE;	⊾	right angle with arc	infix	265	3	3	
&#x22BF;	⊿	right triangle	infix	265	3	3	
&#x22C4;	⋄	diamond operator	infix	265	4	4	
&#x22C6;	⋆	star operator	infix	265	4	4	
&#x22C7;	⋇	division times	infix	265	4	4	
&#x22C8;	⋈	bowtie	infix	265	5	5	
&#x22CD;	⋍	reversed tilde equals	infix	265	5	5	
&#x22CE;	⋎	curly logical or	infix	265	4	4	
&#x22CF;	⋏	curly logical and	infix	265	4	4	
&#x22D0;	⋐	double subset	infix	265	5	5	
&#x22D1;	⋑	double superset	infix	265	5	5	
&#x22D2;	⋒	double intersection	infix	265	4	4	
&#x22D3;	⋓	double union	infix	265	4	4	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x22D5;	⋕	equal and parallel to	infix	265	5	5	
&#x22DA;	⋚	less-than equal to or greater-than	infix	265	5	5	
&#x22DB;	⋛	greater-than equal to or less-than	infix	265	5	5	
&#x22DC;	⋜	equal to or less-than	infix	265	5	5	
&#x22DD;	⋝	equal to or greater-than	infix	265	5	5	
&#x22DE;	⋞	equal to or precedes	infix	265	5	5	
&#x22DF;	⋟	equal to or succeeds	infix	265	5	5	
&#x22E0;	⋠	does not precede or equal	infix	265	5	5	
&#x22E1;	⋡	does not succeed or equal	infix	265	5	5	
&#x22E2;	⋢	not square image of or equal to	infix	265	5	5	
&#x22E3;	⋣	not square original of or equal to	infix	265	5	5	
&#x22E4;	⋤	square image of or not equal to	infix	265	5	5	
&#x22E5;	⋥	square original of or not equal to	infix	265	5	5	
&#x22E6;	⋦	less-than but not equivalent to	infix	265	5	5	
&#x22E7;	⋧	greater-than but not equivalent to	infix	265	5	5	
&#x22E8;	⋨	precedes but not equivalent to	infix	265	5	5	
&#x22E9;	⋩	succeeds but not equivalent to	infix	265	5	5	
&#x22F0;	⋰	up right diagonal ellipsis	infix	265	5	5	
&#x22F2;	⋲	element of with long horizontal stroke	infix	265	5	5	
&#x22F3;	⋳	element of with vertical bar at end of horizontal stroke	infix	265	5	5	
&#x22F4;	⋴	small element of with vertical bar at end of horizontal stroke	infix	265	5	5	
&#x22F5;	⋵	element of with dot above	infix	265	5	5	
&#x22F6;	⋶	element of with overbar	infix	265	5	5	
&#x22F7;	⋷	small element of with overbar	infix	265	5	5	
&#x22F8;	⋸	element of with underbar	infix	265	5	5	
&#x22F9;	⋹	element of with two horizontal strokes	infix	265	5	5	
&#x22FA;	⋺	contains with long horizontal stroke	infix	265	5	5	
&#x22FB;	⋻	contains with vertical bar at end of horizontal stroke	infix	265	5	5	
&#x22FC;	⋼	small contains with vertical bar at end of horizontal stroke	infix	265	5	5	
&#x22FD;	⋽	contains with overbar	infix	265	5	5	
&#x22FE;	⋾	small contains with overbar	infix	265	5	5	
&#x22FF;	⋿	z notation bag membership	infix	265	5	5	
&#x25B2;	▲	black up-pointing triangle	infix	265	4	4	
&#x2758;	❘	light vertical bar	infix	265	5	5	
&#x2981;	⦁	z notation spot	infix	265	3	3	
&#x2982;	⦂	z notation type colon	infix	265	3	3	
&#x29A0;	⦠	spherical angle opening left	infix	265	3	3	
&#x29A1;	⦡	spherical angle opening up	infix	265	3	3	
&#x29A2;	⦢	turned angle	infix	265	3	3	
&#x29A3;	⦣	reversed angle	infix	265	3	3	
&#x29A4;	⦤	angle with underbar	infix	265	3	3	
&#x29A5;	⦥	reversed angle with underbar	infix	265	3	3	
&#x29A6;	⦦	oblique angle opening up	infix	265	3	3	
&#x29A7;	⦧	oblique angle opening down	infix	265	3	3	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x29A8;	⦨	measured angle with open arm ending in arrow pointing up and right	infix	265	3	3	
&#x29A9;	⦩	measured angle with open arm ending in arrow pointing up and left	infix	265	3	3	
&#x29AA;	⦪	measured angle with open arm ending in arrow pointing down and right	infix	265	3	3	
&#x29AB;	⦫	measured angle with open arm ending in arrow pointing down and left	infix	265	3	3	
&#x29AC;	⦬	measured angle with open arm ending in arrow pointing right and up	infix	265	3	3	
&#x29AD;	⦭	measured angle with open arm ending in arrow pointing left and up	infix	265	3	3	
&#x29AE;	⦮	measured angle with open arm ending in arrow pointing right and down	infix	265	3	3	
&#x29AF;	⦯	measured angle with open arm ending in arrow pointing left and down	infix	265	3	3	
&#x29B0;	⦰	reversed empty set	infix	265	3	3	
&#x29B1;	⦱	empty set with overbar	infix	265	3	3	
&#x29B2;	⦲	empty set with small circle above	infix	265	3	3	
&#x29B3;	⦳	empty set with right arrow above	infix	265	3	3	
&#x29B4;	⦴	empty set with left arrow above	infix	265	3	3	
&#x29B5;	⦵	circle with horizontal bar	infix	265	3	3	
&#x29B6;	⦶	circled vertical bar	infix	265	4	4	
&#x29B7;	⦷	circled parallel	infix	265	4	4	
&#x29B8;	⦸	circled reverse solidus	infix	265	4	4	
&#x29B9;	⦹	circled perpendicular	infix	265	4	4	
&#x29BA;	⦺	circle divided by horizontal bar and top half divided by vertical bar	infix	265	4	4	
&#x29BB;	⦻	circle with superimposed x	infix	265	4	4	
&#x29BC;	⦼	circled anticlockwise-rotated division sign	infix	265	4	4	
&#x29BD;	⦽	up arrow through circle	infix	265	4	4	
&#x29BE;	⦾	circled white bullet	infix	265	4	4	
&#x29BF;	⦿	circled bullet	infix	265	4	4	
&#x29C2;	⧂	circle with small circle to the right	infix	265	3	3	
&#x29C3;	⧃	circle with two horizontal strokes to the right	infix	265	3	3	
&#x29C4;	⧄	squared rising diagonal slash	infix	265	4	4	
&#x29C5;	⧅	squared falling diagonal slash	infix	265	4	4	
&#x29C6;	⧆	squared asterisk	infix	265	4	4	
&#x29C7;	⧇	squared small circle	infix	265	4	4	
&#x29C8;	⧈	squared square	infix	265	4	4	
&#x29C9;	⧉	two joined squares	infix	265	3	3	
&#x29CA;	⧊	triangle with dot above	infix	265	3	3	
&#x29CB;	⧋	triangle with underbar	infix	265	3	3	
&#x29CC;	⧌	s in triangle	infix	265	3	3	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x29CD;	⧍	triangle with serifs at bottom	infix	265	3	3	
&#x29CE;	⧎	right triangle above left triangle	infix	265	5	5	
&#x29CF;	⧏	left triangle beside vertical bar	infix	265	5	5	
&#x29CF;&#x338;	⧏̸	left triangle beside vertical bar with slash	infix	265	5	5	
&#x29D0;	⧐	vertical bar beside right triangle	infix	265	5	5	
&#x29D0;&#x338;	⧐̸	vertical bar beside right triangle with slash	infix	265	5	5	
&#x29D1;	⧑	bowtie with left half black	infix	265	5	5	
&#x29D2;	⧒	bowtie with right half black	infix	265	5	5	
&#x29D3;	⧓	black bowtie	infix	265	5	5	
&#x29D4;	⧔	times with left half black	infix	265	5	5	
&#x29D5;	⧕	times with right half black	infix	265	5	5	
&#x29D6;	⧖	white hourglass	infix	265	4	4	
&#x29D7;	⧗	black hourglass	infix	265	4	4	
&#x29D8;	⧘	left wiggly fence	infix	265	3	3	
&#x29D9;	⧙	right wiggly fence	infix	265	3	3	
&#x29DB;	⧛	right double wiggly fence	infix	265	3	3	
&#x29DC;	⧜	incomplete infinity	infix	265	3	3	
&#x29DD;	⧝	tie over infinity	infix	265	3	3	
&#x29DE;	⧞	infinity negated with vertical bar	infix	265	5	5	
&#x29E0;	⧠	square with contoured outline	infix	265	3	3	
&#x29E1;	⧡	increases as	infix	265	5	5	
&#x29E2;	⧢	shuffle product	infix	265	4	4	
&#x29E7;	⧧	thermodynamic	infix	265	3	3	
&#x29E8;	⧨	down-pointing triangle with left half black	infix	265	3	3	
&#x29E9;	⧩	down-pointing triangle with right half black	infix	265	3	3	
&#x29EA;	⧪	black diamond with down arrow	infix	265	3	3	
&#x29EB;	⧫	black lozenge	infix	265	3	3	
&#x29EC;	⧬	white circle with down arrow	infix	265	3	3	
&#x29ED;	⧭	black circle with down arrow	infix	265	3	3	
&#x29EE;	⧮	error-barred white square	infix	265	3	3	
&#x29F0;	⧰	error-barred white diamond	infix	265	3	3	
&#x29F1;	⧱	error-barred black diamond	infix	265	3	3	
&#x29F2;	⧲	error-barred white circle	infix	265	3	3	
&#x29F5;	⧵	reverse solidus operator	infix	265	4	4	
&#x29F6;	⧶	solidus with overbar	infix	265	4	4	
&#x29F7;	⧷	reverse solidus with horizontal stroke	infix	265	4	4	
&#x29F8;	⧸	big solidus	infix	265	3	3	
&#x29F9;	⧹	big reverse solidus	infix	265	3	3	
&#x29FA;	⧺	double plus	infix	265	3	3	
&#x29FB;	⧻	triple plus	infix	265	3	3	
&#x29FE;	⧾	tiny	infix	265	4	4	
&#x29FF;	⧿	miny	infix	265	4	4	
&#x2A1D;	⨝	join	infix	265	3	3	
&#x2A1E;	⨞	large left triangle operator	infix	265	3	3	
&#x2A1F;	⨟	z notation schema composition	infix	265	3	3	
&#x2A20;	⨠	z notation schema piping	infix	265	3	3	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2A21;	⨡	z notation schema projection	infix	265	3	3	
&#x2A22;	⨢	plus sign with small circle above	infix	265	4	4	
&#x2A23;	⨣	plus sign with circumflex accent above	infix	265	4	4	
&#x2A24;	⨤	plus sign with tilde above	infix	265	4	4	
&#x2A25;	⨥	plus sign with dot below	infix	265	4	4	
&#x2A26;	⨦	plus sign with tilde below	infix	265	4	4	
&#x2A27;	⨧	plus sign with subscript two	infix	265	4	4	
&#x2A28;	⨨	plus sign with black triangle	infix	265	4	4	
&#x2A29;	⨩	minus sign with comma above	infix	265	4	4	
&#x2A2A;	⨪	minus sign with dot below	infix	265	4	4	
&#x2A2B;	⨫	minus sign with falling dots	infix	265	4	4	
&#x2A2C;	⨬	minus sign with rising dots	infix	265	4	4	
&#x2A2D;	⨭	plus sign in left half circle	infix	265	4	4	
&#x2A2E;	⨮	plus sign in right half circle	infix	265	4	4	
&#x2A30;	⨰	multiplication sign with dot above	infix	265	4	4	
&#x2A31;	⨱	multiplication sign with underbar	infix	265	4	4	
&#x2A32;	⨲	semidirect product with bottom closed	infix	265	4	4	
&#x2A33;	⨳	smash product	infix	265	4	4	
&#x2A34;	⨴	multiplication sign in left half circle	infix	265	4	4	
&#x2A35;	⨵	multiplication sign in right half circle	infix	265	4	4	
&#x2A36;	⨶	circled multiplication sign with circumflex accent	infix	265	4	4	
&#x2A37;	⨷	multiplication sign in double circle	infix	265	4	4	
&#x2A38;	⨸	circled division sign	infix	265	4	4	
&#x2A39;	⨹	plus sign in triangle	infix	265	4	4	
&#x2A3A;	⨺	minus sign in triangle	infix	265	4	4	
&#x2A3B;	⨻	multiplication sign in triangle	infix	265	4	4	
&#x2A3C;	⨼	interior product	infix	265	4	4	
&#x2A3D;	⨽	righthand interior product	infix	265	4	4	
&#x2A3E;	⨾	z notation relational composition	infix	265	4	4	
&#x2A40;	⩀	intersection with dot	infix	265	4	4	
&#x2A41;	⩁	union with minus sign	infix	265	4	4	
&#x2A42;	⩂	union with overbar	infix	265	4	4	
&#x2A43;	⩃	intersection with overbar	infix	265	4	4	
&#x2A44;	⩄	intersection with logical and	infix	265	4	4	
&#x2A45;	⩅	union with logical or	infix	265	4	4	
&#x2A46;	⩆	union above intersection	infix	265	4	4	
&#x2A47;	⩇	intersection above union	infix	265	4	4	
&#x2A48;	⩈	union above bar above intersection	infix	265	4	4	
&#x2A49;	⩉	intersection above bar above union	infix	265	4	4	
&#x2A4A;	⩊	union beside and joined with union	infix	265	4	4	
&#x2A4B;	⩋	intersection beside and joined with intersection	infix	265	4	4	
&#x2A4C;	⩌	closed union with serifs	infix	265	4	4	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2A4D;	⩍	closed intersection with serifs	infix	265	4	4	
&#x2A4E;	⩎	double square intersection	infix	265	4	4	
&#x2A4F;	⩏	double square union	infix	265	4	4	
&#x2A50;	⩐	closed union with serifs and smash product	infix	265	4	4	
&#x2A51;	⩑	logical and with dot above	infix	265	4	4	
&#x2A52;	⩒	logical or with dot above	infix	265	4	4	
&#x2A53;	⩓	double logical and	infix	265	4	4	
&#x2A54;	⩔	double logical or	infix	265	4	4	
&#x2A55;	⩕	two intersecting logical and	infix	265	4	4	
&#x2A56;	⩖	two intersecting logical or	infix	265	4	4	
&#x2A57;	⩗	sloping large or	infix	265	4	4	
&#x2A58;	⩘	sloping large and	infix	265	4	4	
&#x2A59;	⩙	logical or overlapping logical and	infix	265	5	5	
&#x2A5A;	⩚	logical and with middle stem	infix	265	4	4	
&#x2A5B;	⩛	logical or with middle stem	infix	265	4	4	
&#x2A5C;	⩜	logical and with horizontal dash	infix	265	4	4	
&#x2A5D;	⩝	logical or with horizontal dash	infix	265	4	4	
&#x2A5E;	⩞	logical and with double overbar	infix	265	4	4	
&#x2A5F;	⩟	logical and with underbar	infix	265	4	4	
&#x2A60;	⩠	logical and with double underbar	infix	265	4	4	
&#x2A61;	⩡	small vee with underbar	infix	265	4	4	
&#x2A62;	⩢	logical or with double overbar	infix	265	4	4	
&#x2A63;	⩣	logical or with double underbar	infix	265	4	4	
&#x2A64;	⩤	z notation domain antirestriction	infix	265	4	4	
&#x2A65;	⩥	z notation range antirestriction	infix	265	4	4	
&#x2A66;	⩦	equals sign with dot below	infix	265	5	5	
&#x2A67;	⩧	identical with dot above	infix	265	5	5	
&#x2A68;	⩨	triple horizontal bar with double vertical stroke	infix	265	5	5	
&#x2A69;	⩩	triple horizontal bar with triple vertical stroke	infix	265	5	5	
&#x2A6A;	⩪	tilde operator with dot above	infix	265	5	5	
&#x2A6B;	⩫	tilde operator with rising dots	infix	265	5	5	
&#x2A6C;	⩬	similar minus similar	infix	265	5	5	
&#x2A6D;	⩭	congruent with dot above	infix	265	5	5	
&#x2A6E;	⩮	equals with asterisk	infix	265	5	5	
&#x2A6F;	⩯	almost equal to with circumflex accent	infix	265	5	5	
&#x2A70;	⩰	approximately equal or equal to	infix	265	5	5	
&#x2A71;	⩱	equals sign above plus sign	infix	265	4	4	
&#x2A72;	⩲	plus sign above equals sign	infix	265	4	4	
&#x2A73;	⩳	equals sign above tilde operator	infix	265	5	5	
&#x2A74;	::=	double colon equal	infix	265	5	5	
&#x2A75;	==	two consecutive equals signs	infix	265	5	5	
&#x2A76;	===	three consecutive equals signs	infix	265	5	5	
&#x2A77;	⩷	equals sign with two dots above and two dots below	infix	265	5	5	
&#x2A78;	⩸	equivalent with four dots above	infix	265	5	5	
&#x2A79;	⩹	less-than with circle inside	infix	265	5	5	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2A7A;	⩺	greater-than with circle inside	infix	265	5	5	
&#x2A7B;	⩻	less-than with question mark above	infix	265	5	5	
&#x2A7C;	⩼	greater-than with question mark above	infix	265	5	5	
&#x2A7D;	⩽	less-than or slanted equal to	infix	265	5	5	
&#x2A7D;&#x338;	⩽̸	less-than or slanted equal to with slash	infix	265	5	5	
&#x2A7E;	⩾	greater-than or slanted equal to	infix	265	5	5	
&#x2A7E;&#x338;	⩾̸	greater-than or slanted equal to with slash	infix	265	5	5	
&#x2A7F;	⩿	less-than or slanted equal to with dot inside	infix	265	5	5	
&#x2A80;	⪀	greater-than or slanted equal to with dot inside	infix	265	5	5	
&#x2A81;	⪁	less-than or slanted equal to with dot above	infix	265	5	5	
&#x2A82;	⪂	greater-than or slanted equal to with dot above	infix	265	5	5	
&#x2A83;	⪃	less-than or slanted equal to with dot above right	infix	265	5	5	
&#x2A84;	⪄	greater-than or slanted equal to with dot above left	infix	265	5	5	
&#x2A85;	⪅	less-than or approximate	infix	265	5	5	
&#x2A86;	⪆	greater-than or approximate	infix	265	5	5	
&#x2A89;	⪉	less-than and not approximate	infix	265	5	5	
&#x2A8A;	⪊	greater-than and not approximate	infix	265	5	5	
&#x2A8B;	⪋	less-than above double-line equal above greater-than	infix	265	5	5	
&#x2A8C;	⪌	greater-than above double-line equal above less-than	infix	265	5	5	
&#x2A8D;	⪍	less-than above similar or equal	infix	265	5	5	
&#x2A8E;	⪎	greater-than above similar or equal	infix	265	5	5	
&#x2A8F;	⪏	less-than above similar above greater-than	infix	265	5	5	
&#x2A90;	⪐	greater-than above similar above less-than	infix	265	5	5	
&#x2A91;	⪑	less-than above greater-than above double-line equal	infix	265	5	5	
&#x2A92;	⪒	greater-than above less-than above double-line equal	infix	265	5	5	
&#x2A93;	⪓	less-than above slanted equal above greater-than above slanted equal	infix	265	5	5	
&#x2A94;	⪔	greater-than above slanted equal above less-than above slanted equal	infix	265	5	5	
&#x2A95;	⪕	slanted equal to or less-than	infix	265	5	5	
&#x2A96;	⪖	slanted equal to or greater-than	infix	265	5	5	
&#x2A97;	⪗	slanted equal to or less-than with dot inside	infix	265	5	5	
&#x2A98;	⪘	slanted equal to or greater-than with dot inside	infix	265	5	5	
&#x2A99;	⪙	double-line equal to or less-than	infix	265	5	5	
&#x2A9A;	⪚	double-line equal to or greater-than	infix	265	5	5	
&#x2A9B;	⪛	double-line slanted equal to or less-than	infix	265	5	5	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2A9C;	⪜	double-line slanted equal to or greater-than	infix	265	5	5	
&#x2A9D;	⪝	similar or less-than	infix	265	5	5	
&#x2A9E;	⪞	similar or greater-than	infix	265	5	5	
&#x2A9F;	⪟	similar above less-than above equals sign	infix	265	5	5	
&#x2AA0;	⪠	similar above greater-than above equals sign	infix	265	5	5	
&#x2AA1;	⪡	double nested less-than	infix	265	5	5	
&#x2AA1;&#x338;	⪡̸	double nested less-than with slash	infix	265	5	5	
&#x2AA2;	⪢	double nested greater-than	infix	265	5	5	
&#x2AA2;&#x338;	⪢̸	double nested greater-than with slash	infix	265	5	5	
&#x2AA3;	⪣	double nested less-than with underbar	infix	265	5	5	
&#x2AA4;	⪤	greater-than overlapping less-than	infix	265	5	5	
&#x2AA5;	⪥	greater-than beside less-than	infix	265	5	5	
&#x2AA6;	⪦	less-than closed by curve	infix	265	5	5	
&#x2AA7;	⪧	greater-than closed by curve	infix	265	5	5	
&#x2AA8;	⪨	less-than closed by curve above slanted equal	infix	265	5	5	
&#x2AA9;	⪩	greater-than closed by curve above slanted equal	infix	265	5	5	
&#x2AAA;	⪪	smaller than	infix	265	5	5	
&#x2AAB;	⪫	larger than	infix	265	5	5	
&#x2AAC;	⪬	smaller than or equal to	infix	265	5	5	
&#x2AAD;	⪭	larger than or equal to	infix	265	5	5	
&#x2AAE;	⪮	equals sign with bumpy above	infix	265	5	5	
&#x2AB1;	⪱	precedes above single-line not equal to	infix	265	5	5	
&#x2AB2;	⪲	succeeds above single-line not equal to	infix	265	5	5	
&#x2AB3;	⪳	precedes above equals sign	infix	265	5	5	
&#x2AB4;	⪴	succeeds above equals sign	infix	265	5	5	
&#x2AB5;	⪵	precedes above not equal to	infix	265	5	5	
&#x2AB6;	⪶	succeeds above not equal to	infix	265	5	5	
&#x2AB7;	⪷	precedes above almost equal to	infix	265	5	5	
&#x2AB8;	⪸	succeeds above almost equal to	infix	265	5	5	
&#x2AB9;	⪹	precedes above not almost equal to	infix	265	5	5	
&#x2ABA;	⪺	succeeds above not almost equal to	infix	265	5	5	
&#x2ABB;	⪻	double precedes	infix	265	5	5	
&#x2ABC;	⪼	double succeeds	infix	265	5	5	
&#x2ABD;	⪽	subset with dot	infix	265	5	5	
&#x2ABE;	⪾	superset with dot	infix	265	5	5	
&#x2ABF;	⪿	subset with plus sign below	infix	265	5	5	
&#x2AC0;	⫀	superset with plus sign below	infix	265	5	5	
&#x2AC1;	⫁	subset with multiplication sign below	infix	265	5	5	
&#x2AC2;	⫂	superset with multiplication sign below	infix	265	5	5	
&#x2AC3;	⫃	subset of or equal to with dot above	infix	265	5	5	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2AC4;	⫄	superset of or equal to with dot above	infix	265	5	5	
&#x2AC5;	⫅	subset of above equals sign	infix	265	5	5	
&#x2AC6;	⫆	superset of above equals sign	infix	265	5	5	
&#x2AC7;	⫇	subset of above tilde operator	infix	265	5	5	
&#x2AC8;	⫈	superset of above tilde operator	infix	265	5	5	
&#x2AC9;	⫉	subset of above almost equal to	infix	265	5	5	
&#x2ACA;	⫊	superset of above almost equal to	infix	265	5	5	
&#x2ACB;	⫋	subset of above not equal to	infix	265	5	5	
&#x2ACC;	⫌	superset of above not equal to	infix	265	5	5	
&#x2ACD;	⫍	square left open box operator	infix	265	5	5	
&#x2ACE;	⫎	square right open box operator	infix	265	5	5	
&#x2ACF;	⫏	closed subset	infix	265	5	5	
&#x2AD0;	⫐	closed superset	infix	265	5	5	
&#x2AD1;	⫑	closed subset or equal to	infix	265	5	5	
&#x2AD2;	⫒	closed superset or equal to	infix	265	5	5	
&#x2AD3;	⫓	subset above superset	infix	265	5	5	
&#x2AD4;	⫔	superset above subset	infix	265	5	5	
&#x2AD5;	⫕	subset above subset	infix	265	5	5	
&#x2AD6;	⫖	superset above superset	infix	265	5	5	
&#x2AD7;	⫗	superset beside subset	infix	265	5	5	
&#x2AD8;	⫘	superset beside and joined by dash with subset	infix	265	5	5	
&#x2AD9;	⫙	element of opening downwards	infix	265	5	5	
&#x2ADA;	⫚	pitchfork with tee top	infix	265	5	5	
&#x2ADB;	⫛	transversal intersection	infix	265	5	5	
&#x2ADD;	⫝	nonforking	infix	265	5	5	
&#x2ADD;&#x338;	⫝̸	nonforking with slash	infix	265	5	5	
&#x2ADE;	⫞	short left tack	infix	265	5	5	
&#x2ADF;	⫟	short down tack	infix	265	5	5	
&#x2AE0;	⫠	short up tack	infix	265	5	5	
&#x2AE1;	⫡	perpendicular with s	infix	265	5	5	
&#x2AE2;	⫢	vertical bar triple right turnstile	infix	265	5	5	
&#x2AE3;	⫣	double vertical bar left turnstile	infix	265	5	5	
&#x2AE4;	⫤	vertical bar double left turnstile	infix	265	5	5	
&#x2AE5;	⫥	double vertical bar double left turnstile	infix	265	5	5	
&#x2AE6;	⫦	long dash from left member of double vertical	infix	265	5	5	
&#x2AE7;	⫧	short down tack with overbar	infix	265	5	5	
&#x2AE8;	⫨	short up tack with underbar	infix	265	5	5	
&#x2AE9;	⫩	short up tack above short down tack	infix	265	5	5	
&#x2AEA;	⫪	double down tack	infix	265	5	5	
&#x2AEB;	⫫	double up tack	infix	265	5	5	
&#x2AEC;	⫬	double stroke not sign	infix	265	5	5	
&#x2AED;	⫭	reversed double stroke not sign	infix	265	5	5	
&#x2AEE;	⫮	does not divide with reversed negation slash	infix	265	5	5	
&#x2AEF;	⫯	vertical line with circle above	infix	265	5	5	
&#x2AF0;	⫰	vertical line with circle below	infix	265	5	5	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2AF1;	⫱	down tack with circle below	infix	265	5	5	
&#x2AF2;	⫲	parallel with horizontal stroke	infix	265	5	5	
&#x2AF3;	⫳	parallel with tilde operator	infix	265	5	5	
&#x2AF4;	⫴	triple vertical bar binary relation	infix	265	4	4	
&#x2AF5;	⫵	triple vertical bar with horizontal stroke	infix	265	4	4	
&#x2AF6;	⫶	triple colon operator	infix	265	4	4	
&#x2AF7;	⫷	triple nested less-than	infix	265	5	5	
&#x2AF8;	⫸	triple nested greater-than	infix	265	5	5	
&#x2AF9;	⫹	double-line slanted less-than or equal to	infix	265	5	5	
&#x2AFA;	⫺	double-line slanted greater-than or equal to	infix	265	5	5	
&#x2AFB;	⫻	triple solidus binary relation	infix	265	4	4	
&#x2AFD;	⫽	double solidus operator	infix	265	4	4	
&#x2AFE;	⫾	white vertical bar	infix	265	3	3	
|	|	vertical line	infix	270	2	2	fence, stretchy, symmetric
||	||	multiple character operator: ||	infix	270	2	2	fence, stretchy, symmetric
|||	|||	multiple character operator: |||	infix	270	2	2	fence, stretchy, symmetric
&#x2190;	←	leftwards arrow	infix	270	5	5	stretchy, accent
&#x2191;	↑	upwards arrow	infix	270	5	5	stretchy
&#x2192;	→	rightwards arrow	infix	270	5	5	stretchy, accent
&#x2193;	↓	downwards arrow	infix	270	5	5	stretchy
&#x2194;	↔	left right arrow	infix	270	5	5	stretchy, accent
&#x2195;	↕	up down arrow	infix	270	5	5	stretchy
&#x2196;	↖	north west arrow	infix	270	5	5	stretchy
&#x2197;	↗	north east arrow	infix	270	5	5	stretchy
&#x2198;	↘	south east arrow	infix	270	5	5	stretchy
&#x2199;	↙	south west arrow	infix	270	5	5	stretchy
&#x219A;	↚	leftwards arrow with stroke	infix	270	5	5	accent
&#x219B;	↛	rightwards arrow with stroke	infix	270	5	5	accent
&#x219C;	↜	leftwards wave arrow	infix	270	5	5	stretchy, accent
&#x219D;	↝	rightwards wave arrow	infix	270	5	5	stretchy, accent
&#x219E;	↞	leftwards two headed arrow	infix	270	5	5	stretchy, accent
&#x219F;	↟	upwards two headed arrow	infix	270	5	5	stretchy, accent
&#x21A0;	↠	rightwards two headed arrow	infix	270	5	5	stretchy, accent
&#x21A1;	↡	downwards two headed arrow	infix	270	5	5	stretchy
&#x21A2;	↢	leftwards arrow with tail	infix	270	5	5	stretchy, accent
&#x21A3;	↣	rightwards arrow with tail	infix	270	5	5	stretchy, accent
&#x21A4;	↤	leftwards arrow from bar	infix	270	5	5	stretchy, accent
&#x21A5;	↥	upwards arrow from bar	infix	270	5	5	stretchy
&#x21A6;	↦	rightwards arrow from bar	infix	270	5	5	stretchy, accent
&#x21A7;	↧	downwards arrow from bar	infix	270	5	5	stretchy
&#x21A8;	↨	up down arrow with base	infix	270	5	5	stretchy
&#x21A9;	↩	leftwards arrow with hook	infix	270	5	5	stretchy, accent
&#x21AA;	↪	rightwards arrow with hook	infix	270	5	5	stretchy, accent
&#x21AB;	↫	leftwards arrow with loop	infix	270	5	5	stretchy, accent
&#x21AC;	↬	rightwards arrow with loop	infix	270	5	5	stretchy, accent
&#x21AD;	↭	left right wave arrow	infix	270	5	5	stretchy, accent
&#x21AE;	↮	left right arrow with stroke	infix	270	5	5	accent

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x21AF;	↯	downwards zigzag arrow	infix	270	5	5	stretchy
&#x21B0;	↰	upwards arrow with tip leftwards	infix	270	5	5	stretchy
&#x21B1;	↱	upwards arrow with tip rightwards	infix	270	5	5	stretchy
&#x21B2;	↲	downwards arrow with tip leftwards	infix	270	5	5	stretchy
&#x21B3;	↳	downwards arrow with tip rightwards	infix	270	5	5	stretchy
&#x21B4;	↴	rightwards arrow with corner downwards	infix	270	5	5	stretchy
&#x21B5;	↵	downwards arrow with corner leftwards	infix	270	5	5	stretchy
&#x21B6;	↶	anticlockwise top semicircle arrow	infix	270	5	5	accent
&#x21B7;	↷	clockwise top semicircle arrow	infix	270	5	5	accent
&#x21B8;	↸	north west arrow to long bar	infix	270	5	5	
&#x21B9;	↹	leftwards arrow to bar over rightwards arrow to bar	infix	270	5	5	stretchy, accent
&#x21BA;	↺	anticlockwise open circle arrow	infix	270	5	5	
&#x21BB;	↻	clockwise open circle arrow	infix	270	5	5	
&#x21BC;	↼	leftwards harpoon with barb upwards	infix	270	5	5	stretchy, accent
&#x21BD;	↽	leftwards harpoon with barb downwards	infix	270	5	5	stretchy, accent
&#x21BE;	↾	upwards harpoon with barb rightwards	infix	270	5	5	stretchy
&#x21BF;	↿	upwards harpoon with barb leftwards	infix	270	5	5	stretchy
&#x21C0;	⇀	rightwards harpoon with barb upwards	infix	270	5	5	stretchy, accent
&#x21C1;	⇁	rightwards harpoon with barb downwards	infix	270	5	5	stretchy, accent
&#x21C2;	⇂	downwards harpoon with barb rightwards	infix	270	5	5	stretchy
&#x21C3;	⇃	downwards harpoon with barb leftwards	infix	270	5	5	stretchy
&#x21C4;	⇄	rightwards arrow over leftwards arrow	infix	270	5	5	stretchy, accent
&#x21C5;	⇅	upwards arrow leftwards of downwards arrow	infix	270	5	5	stretchy
&#x21C6;	⇆	leftwards arrow over rightwards arrow	infix	270	5	5	stretchy, accent
&#x21C7;	⇇	leftwards paired arrows	infix	270	5	5	stretchy, accent
&#x21C8;	⇈	upwards paired arrows	infix	270	5	5	stretchy
&#x21C9;	⇉	rightwards paired arrows	infix	270	5	5	stretchy, accent
&#x21CA;	⇊	downwards paired arrows	infix	270	5	5	stretchy
&#x21CB;	⇋	leftwards harpoon over rightwards harpoon	infix	270	5	5	stretchy, accent
&#x21CC;	⇌	rightwards harpoon over leftwards harpoon	infix	270	5	5	stretchy, accent
&#x21CD;	⇍	leftwards double arrow with stroke	infix	270	5	5	accent
&#x21CE;	⇎	left right double arrow with stroke	infix	270	5	5	accent
&#x21CF;	⇏	rightwards double arrow with stroke	infix	270	5	5	accent
&#x21D0;	⇐	leftwards double arrow	infix	270	5	5	stretchy, accent
&#x21D1;	⇑	upwards double arrow	infix	270	5	5	stretchy
&#x21D2;	⇒	rightwards double arrow	infix	270	5	5	stretchy, accent

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x21D3;	⇓	downwards double arrow	infix	270	5	5	stretchy
&#x21D4;	⇔	left right double arrow	infix	270	5	5	stretchy, accent
&#x21D5;	⇕	up down double arrow	infix	270	5	5	stretchy
&#x21D6;	⇖	north west double arrow	infix	270	5	5	stretchy
&#x21D7;	⇗	north east double arrow	infix	270	5	5	stretchy
&#x21D8;	⇘	south east double arrow	infix	270	5	5	stretchy
&#x21D9;	⇙	south west double arrow	infix	270	5	5	stretchy
&#x21DA;	⇚	leftwards triple arrow	infix	270	5	5	stretchy, accent
&#x21DB;	⇛	rightwards triple arrow	infix	270	5	5	stretchy, accent
&#x21DC;	⇜	leftwards squiggle arrow	infix	270	5	5	stretchy, accent
&#x21DD;	⇝	rightwards squiggle arrow	infix	270	5	5	stretchy, accent
&#x21DE;	⇞	upwards arrow with double stroke	infix	270	5	5	
&#x21DF;	⇟	downwards arrow with double stroke	infix	270	5	5	
&#x21E0;	⇠	leftwards dashed arrow	infix	270	5	5	stretchy, accent
&#x21E1;	⇡	upwards dashed arrow	infix	270	5	5	stretchy
&#x21E2;	⇢	rightwards dashed arrow	infix	270	5	5	stretchy, accent
&#x21E3;	⇣	downwards dashed arrow	infix	270	5	5	stretchy
&#x21E4;	⇤	leftwards arrow to bar	infix	270	5	5	stretchy, accent
&#x21E5;	⇥	rightwards arrow to bar	infix	270	5	5	stretchy, accent
&#x21E6;	⇦	leftwards white arrow	infix	270	5	5	stretchy, accent
&#x21E7;	⇧	upwards white arrow	infix	270	5	5	stretchy
&#x21E8;	⇨	rightwards white arrow	infix	270	5	5	stretchy, accent
&#x21E9;	⇩	downwards white arrow	infix	270	5	5	stretchy
&#x21EA;	⇪	upwards white arrow from bar	infix	270	5	5	stretchy
&#x21EB;	⇫	upwards white arrow on pedestal	infix	270	5	5	stretchy
&#x21EC;	⇬	upwards white arrow on pedestal with horizontal bar	infix	270	5	5	stretchy
&#x21ED;	⇭	upwards white arrow on pedestal with vertical bar	infix	270	5	5	stretchy
&#x21EE;	⇮	upwards white double arrow	infix	270	5	5	stretchy
&#x21EF;	⇯	upwards white double arrow on pedestal	infix	270	5	5	stretchy
&#x21F0;	⇰	rightwards white arrow from wall	infix	270	5	5	stretchy, accent
&#x21F1;	⇱	north west arrow to corner	infix	270	5	5	
&#x21F2;	⇲	south east arrow to corner	infix	270	5	5	
&#x21F3;	⇳	up down white arrow	infix	270	5	5	stretchy
&#x21F4;	⇴	right arrow with small circle	infix	270	5	5	accent
&#x21F5;	⇵	downwards arrow leftwards of upwards arrow	infix	270	5	5	stretchy
&#x21F6;	⇶	three rightwards arrows	infix	270	5	5	stretchy, accent
&#x21F7;	⇷	leftwards arrow with vertical stroke	infix	270	5	5	accent
&#x21F8;	⇸	rightwards arrow with vertical stroke	infix	270	5	5	accent
&#x21F9;	⇹	left right arrow with vertical stroke	infix	270	5	5	accent
&#x21FA;	⇺	leftwards arrow with double vertical stroke	infix	270	5	5	accent
&#x21FB;	⇻	rightwards arrow with double vertical stroke	infix	270	5	5	accent

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x21FC;	⇼	left right arrow with double vertical stroke	infix	270	5	5	accent
&#x21FD;	⇽	leftwards open-headed arrow	infix	270	5	5	stretchy, accent
&#x21FE;	⇾	rightwards open-headed arrow	infix	270	5	5	stretchy, accent
&#x21FF;	⇿	left right open-headed arrow	infix	270	5	5	stretchy, accent
&#x22B8;	⊸	multimap	infix	270	5	5	
&#x27F0;	⟰	upwards quadruple arrow	infix	270	5	5	stretchy
&#x27F1;	⟱	downwards quadruple arrow	infix	270	5	5	stretchy
&#x27F5;	⟵	long leftwards arrow	infix	270	5	5	stretchy, accent
&#x27F6;	⟶	long rightwards arrow	infix	270	5	5	stretchy, accent
&#x27F7;	⟷	long left right arrow	infix	270	5	5	stretchy, accent
&#x27F8;	⟸	long leftwards double arrow	infix	270	5	5	stretchy, accent
&#x27F9;	⟹	long rightwards double arrow	infix	270	5	5	stretchy, accent
&#x27FA;	⟺	long left right double arrow	infix	270	5	5	stretchy, accent
&#x27FB;	⟻	long leftwards arrow from bar	infix	270	5	5	stretchy, accent
&#x27FC;	⟼	long rightwards arrow from bar	infix	270	5	5	stretchy, accent
&#x27FD;	⟽	long leftwards double arrow from bar	infix	270	5	5	stretchy, accent
&#x27FE;	⟾	long rightwards double arrow from bar	infix	270	5	5	stretchy, accent
&#x27FF;	⟿	long rightwards squiggle arrow	infix	270	5	5	stretchy, accent
&#x2900;	⤀	rightwards two-headed arrow with vertical stroke	infix	270	5	5	accent
&#x2901;	⤁	rightwards two-headed arrow with double vertical stroke	infix	270	5	5	accent
&#x2902;	⤂	leftwards double arrow with vertical stroke	infix	270	5	5	accent
&#x2903;	⤃	rightwards double arrow with vertical stroke	infix	270	5	5	accent
&#x2904;	⤄	left right double arrow with vertical stroke	infix	270	5	5	accent
&#x2905;	⤅	rightwards two-headed arrow from bar	infix	270	5	5	accent
&#x2906;	⤆	leftwards double arrow from bar	infix	270	5	5	accent
&#x2907;	⤇	rightwards double arrow from bar	infix	270	5	5	accent
&#x2908;	⤈	downwards arrow with horizontal stroke	infix	270	5	5	
&#x2909;	⤉	upwards arrow with horizontal stroke	infix	270	5	5	
&#x290A;	⤊	upwards triple arrow	infix	270	5	5	stretchy
&#x290B;	⤋	downwards triple arrow	infix	270	5	5	stretchy
&#x290C;	⤌	leftwards double dash arrow	infix	270	5	5	stretchy, accent
&#x290D;	⤍	rightwards double dash arrow	infix	270	5	5	stretchy, accent
&#x290E;	⤎	leftwards triple dash arrow	infix	270	5	5	stretchy, accent
&#x290F;	⤏	rightwards triple dash arrow	infix	270	5	5	stretchy, accent
&#x2910;	⤐	rightwards two-headed triple dash arrow	infix	270	5	5	stretchy, accent
&#x2911;	⤑	rightwards arrow with dotted stem	infix	270	5	5	accent
&#x2912;	⤒	upwards arrow to bar	infix	270	5	5	stretchy
&#x2913;	⤓	downwards arrow to bar	infix	270	5	5	stretchy
&#x2914;	⤔	rightwards arrow with tail with vertical stroke	infix	270	5	5	accent
&#x2915;	⤕	rightwards arrow with tail with double vertical stroke	infix	270	5	5	accent

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2916;	⤖	rightwards two-headed arrow with tail	infix	270	5	5	accent
&#x2917;	⤗	rightwards two-headed arrow with tail with vertical stroke	infix	270	5	5	accent
&#x2918;	⤘	rightwards two-headed arrow with tail with double vertical stroke	infix	270	5	5	accent
&#x2919;	⤙	leftwards arrow-tail	infix	270	5	5	accent
&#x291A;	⤚	rightwards arrow-tail	infix	270	5	5	accent
&#x291B;	⤛	leftwards double arrow-tail	infix	270	5	5	accent
&#x291C;	⤜	rightwards double arrow-tail	infix	270	5	5	accent
&#x291D;	⤝	leftwards arrow to black diamond	infix	270	5	5	accent
&#x291E;	⤞	rightwards arrow to black diamond	infix	270	5	5	accent
&#x291F;	⤟	leftwards arrow from bar to black diamond	infix	270	5	5	accent
&#x2920;	⤠	rightwards arrow from bar to black diamond	infix	270	5	5	accent
&#x2921;	⤡	north west and south east arrow	infix	270	5	5	stretchy
&#x2922;	⤢	north east and south west arrow	infix	270	5	5	stretchy
&#x2923;	⤣	north west arrow with hook	infix	270	5	5	
&#x2924;	⤤	north east arrow with hook	infix	270	5	5	
&#x2925;	⤥	south east arrow with hook	infix	270	5	5	
&#x2926;	⤦	south west arrow with hook	infix	270	5	5	
&#x2927;	⤧	north west arrow and north east arrow	infix	270	5	5	
&#x2928;	⤨	north east arrow and south east arrow	infix	270	5	5	
&#x2929;	⤩	south east arrow and south west arrow	infix	270	5	5	
&#x292A;	⤪	south west arrow and north west arrow	infix	270	5	5	
&#x292B;	⤫	rising diagonal crossing falling diagonal	infix	270	5	5	
&#x292C;	⤬	falling diagonal crossing rising diagonal	infix	270	5	5	
&#x292D;	⤭	south east arrow crossing north east arrow	infix	270	5	5	
&#x292E;	⤮	north east arrow crossing south east arrow	infix	270	5	5	
&#x292F;	⤯	falling diagonal crossing north east arrow	infix	270	5	5	
&#x2930;	⤰	rising diagonal crossing south east arrow	infix	270	5	5	
&#x2931;	⤱	north east arrow crossing north west arrow	infix	270	5	5	
&#x2932;	⤲	north west arrow crossing north east arrow	infix	270	5	5	
&#x2933;	⤳	wave arrow pointing directly right	infix	270	5	5	accent
&#x2934;	⤴	arrow pointing rightwards then curving upwards	infix	270	5	5	
&#x2935;	⤵	arrow pointing rightwards then curving downwards	infix	270	5	5	
&#x2936;	⤶	arrow pointing downwards then curving leftwards	infix	270	5	5	
&#x2937;	⤷	arrow pointing downwards then curving rightwards	infix	270	5	5	
&#x2938;	⤸	right-side arc clockwise arrow	infix	270	5	5	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2939;	⤹	left-side arc anticlockwise arrow	infix	270	5	5	
&#x293A;	⤺	top arc anticlockwise arrow	infix	270	5	5	accent
&#x293B;	⤻	bottom arc anticlockwise arrow	infix	270	5	5	accent
&#x293C;	⤼	top arc clockwise arrow with minus	infix	270	5	5	accent
&#x293D;	⤽	top arc anticlockwise arrow with plus	infix	270	5	5	accent
&#x293E;	⤾	lower right semicircular clockwise arrow	infix	270	5	5	
&#x293F;	⤿	lower left semicircular anticlockwise arrow	infix	270	5	5	
&#x2940;	⥀	anticlockwise closed circle arrow	infix	270	5	5	
&#x2941;	⥁	clockwise closed circle arrow	infix	270	5	5	
&#x2942;	⥂	rightwards arrow above short leftwards arrow	infix	270	5	5	accent
&#x2943;	⥃	leftwards arrow above short rightwards arrow	infix	270	5	5	accent
&#x2944;	⥄	short rightwards arrow above leftwards arrow	infix	270	5	5	accent
&#x2945;	⥅	rightwards arrow with plus below	infix	270	5	5	accent
&#x2946;	⥆	leftwards arrow with plus below	infix	270	5	5	accent
&#x2947;	⥇	rightwards arrow through x	infix	270	5	5	accent
&#x2948;	⥈	left right arrow through small circle	infix	270	5	5	accent
&#x2949;	⥉	upwards two-headed arrow from small circle	infix	270	5	5	
&#x294A;	⥊	left barb up right barb down harpoon	infix	270	5	5	accent
&#x294B;	⥋	left barb down right barb up harpoon	infix	270	5	5	accent
&#x294C;	⥌	up barb right down barb left harpoon	infix	270	5	5	
&#x294D;	⥍	up barb left down barb right harpoon	infix	270	5	5	
&#x294E;	⥎	left barb up right barb up harpoon	infix	270	5	5	stretchy, accent
&#x294F;	⥏	up barb right down barb right harpoon	infix	270	5	5	stretchy
&#x2950;	⥐	left barb down right barb down harpoon	infix	270	5	5	stretchy, accent
&#x2951;	⥑	up barb left down barb left harpoon	infix	270	5	5	stretchy
&#x2952;	⥒	leftwards harpoon with barb up to bar	infix	270	5	5	stretchy, accent
&#x2953;	⥓	rightwards harpoon with barb up to bar	infix	270	5	5	stretchy, accent
&#x2954;	⥔	upwards harpoon with barb right to bar	infix	270	5	5	stretchy
&#x2955;	⥕	downwards harpoon with barb right to bar	infix	270	5	5	stretchy
&#x2956;	⥖	leftwards harpoon with barb down to bar	infix	270	5	5	stretchy
&#x2957;	⥗	rightwards harpoon with barb down to bar	infix	270	5	5	stretchy
&#x2958;	⥘	upwards harpoon with barb left to bar	infix	270	5	5	stretchy
&#x2959;	⥙	downwards harpoon with barb left to bar	infix	270	5	5	stretchy
&#x295A;	⥚	leftwards harpoon with barb up from bar	infix	270	5	5	stretchy, accent

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x295B;	⥛	rightwards harpoon with barb up from bar	infix	270	5	5	stretchy, accent
&#x295C;	⥜	upwards harpoon with barb right from bar	infix	270	5	5	stretchy
&#x295D;	⥝	downwards harpoon with barb right from bar	infix	270	5	5	stretchy
&#x295E;	⥞	leftwards harpoon with barb down from bar	infix	270	5	5	stretchy, accent
&#x295F;	⥟	rightwards harpoon with barb down from bar	infix	270	5	5	stretchy, accent
&#x2960;	⥠	upwards harpoon with barb left from bar	infix	270	5	5	stretchy
&#x2961;	⥡	downwards harpoon with barb left from bar	infix	270	5	5	stretchy
&#x2962;	⥢	leftwards harpoon with barb up above leftwards harpoon with barb down	infix	270	5	5	accent
&#x2963;	⥣	upwards harpoon with barb left beside upwards harpoon with barb right	infix	270	5	5	
&#x2964;	⥤	rightwards harpoon with barb up above rightwards harpoon with barb down	infix	270	5	5	accent
&#x2965;	⥥	downwards harpoon with barb left beside downwards harpoon with barb right	infix	270	5	5	
&#x2966;	⥦	leftwards harpoon with barb up above rightwards harpoon with barb up	infix	270	5	5	accent
&#x2967;	⥧	leftwards harpoon with barb down above rightwards harpoon with barb down	infix	270	5	5	accent
&#x2968;	⥨	rightwards harpoon with barb up above leftwards harpoon with barb up	infix	270	5	5	accent
&#x2969;	⥩	rightwards harpoon with barb down above leftwards harpoon with barb down	infix	270	5	5	accent
&#x296A;	⥪	leftwards harpoon with barb up above long dash	infix	270	5	5	accent
&#x296B;	⥫	leftwards harpoon with barb down below long dash	infix	270	5	5	accent
&#x296C;	⥬	rightwards harpoon with barb up above long dash	infix	270	5	5	accent
&#x296D;	⥭	rightwards harpoon with barb down below long dash	infix	270	5	5	accent
&#x296E;	⥮	upwards harpoon with barb left beside downwards harpoon with barb right	infix	270	5	5	stretchy
&#x296F;	⥯	downwards harpoon with barb left beside upwards harpoon with barb right	infix	270	5	5	stretchy
&#x2970;	⥰	right double arrow with rounded head	infix	270	5	5	accent
&#x2971;	⥱	equals sign above rightwards arrow	infix	270	5	5	accent
&#x2972;	⥲	tilde operator above rightwards arrow	infix	270	5	5	accent
&#x2973;	⥳	leftwards arrow above tilde operator	infix	270	5	5	accent
&#x2974;	⥴	rightwards arrow above tilde operator	infix	270	5	5	accent
&#x2975;	⥵	rightwards arrow above almost equal to	infix	270	5	5	accent
&#x2976;	⥶	less-than above leftwards arrow	infix	270	5	5	accent

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2977;	⥷	leftwards arrow through less-than	infix	270	5	5	accent
&#x2978;	⥸	greater-than above rightwards arrow	infix	270	5	5	accent
&#x2979;	⥹	subset above rightwards arrow	infix	270	5	5	accent
&#x297A;	⥺	leftwards arrow through subset	infix	270	5	5	accent
&#x297B;	⥻	superset above leftwards arrow	infix	270	5	5	accent
&#x297C;	⥼	left fish tail	infix	270	5	5	accent
&#x297D;	⥽	right fish tail	infix	270	5	5	accent
&#x297E;	⥾	up fish tail	infix	270	5	5	
&#x297F;	⥿	down fish tail	infix	270	5	5	
&#x2999;	⦙	dotted fence	infix	270	3	3	
&#x299A;	⦚	vertical zigzag line	infix	270	3	3	
&#x299B;	⦛	measured angle opening left	infix	270	3	3	
&#x299C;	⦜	right angle variant with square	infix	270	3	3	
&#x299D;	⦝	measured right angle with dot	infix	270	3	3	
&#x299E;	⦞	angle with s inside	infix	270	3	3	
&#x299F;	⦟	acute angle	infix	270	3	3	
&#x29DF;	⧟	double-ended multimap	infix	270	3	3	
&#x29EF;	⧯	error-barred black square	infix	270	3	3	
&#x29F4;	⧴	rule-delayed	infix	270	5	5	
&#x2B45;	⭅	leftwards quadruple arrow	infix	270	5	5	stretchy
&#x2B46;	⭆	rightwards quadruple arrow	infix	270	5	5	stretchy
+	+	plus sign	infix	275	4	4	
+	+	plus sign	prefix	275	0	1	
-	-	hyphen-minus	infix	275	4	4	
-	-	hyphen-minus	prefix	275	0	1	
&#xB1;	±	plus-minus sign	infix	275	4	4	
&#xB1;	±	plus-minus sign	prefix	275	0	1	
&#x2212;	−	minus sign	infix	275	4	4	
&#x2212;	−	minus sign	prefix	275	0	1	
&#x2213;	∓	minus-or-plus sign	infix	275	4	4	
&#x2213;	∓	minus-or-plus sign	prefix	275	0	1	
&#x2214;	∔	dot plus	infix	275	4	4	
&#x229E;	⊞	squared plus	infix	275	4	4	
&#x229F;	⊟	squared minus	infix	275	4	4	
&#x2211;	∑	n-ary summation	prefix	290	1	2	largeop, movablelimits, symmetric
&#x2A0A;	⨊	modulo two sum	prefix	290	1	2	largeop, movablelimits, symmetric
&#x2A0B;	⨋	summation with integral	prefix	290	1	2	largeop, symmetric
&#x222C;	∬	double integral	prefix	300	0	1	largeop, symmetric
&#x222D;	∭	triple integral	prefix	300	0	1	largeop, symmetric
&#x2295;	⊕	circled plus	infix	300	4	4	
&#x2296;	⊖	circled minus	infix	300	4	4	
&#x2298;	⊘	circled division slash	infix	300	4	4	
&#x2A01;	⨁	n-ary circled plus operator	prefix	300	1	2	largeop, movablelimits, symmetric
&#x222B;	∫	integral	prefix	310	0	1	largeop, symmetric
&#x222E;	∮	contour integral	prefix	310	0	1	largeop, symmetric

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x222F;	∯	surface integral	prefix	310	0	1	largeop, symmetric
&#x2230;	∰	volume integral	prefix	310	0	1	largeop, symmetric
&#x2231;	∱	clockwise integral	prefix	310	0	1	largeop, symmetric
&#x2232;	∲	clockwise contour integral	prefix	310	0	1	largeop, symmetric
&#x2233;	∳	anticlockwise contour integral	prefix	310	0	1	largeop, symmetric
&#x2A0C;	⨌	quadruple integral operator	prefix	310	0	1	largeop, symmetric
&#x2A0D;	⨍	finite part integral	prefix	310	1	2	largeop, symmetric
&#x2A0E;	⨎	integral with double stroke	prefix	310	1	2	largeop, symmetric
&#x2A0F;	⨏	integral average with slash	prefix	310	1	2	largeop, symmetric
&#x2A10;	⨐	circulation function	prefix	310	1	2	largeop, movablelimits, symmetric
&#x2A11;	⨑	anticlockwise integration	prefix	310	1	2	largeop, movablelimits, symmetric
&#x2A12;	⨒	line integration with rectangular path around pole	prefix	310	1	2	largeop, movablelimits, symmetric
&#x2A13;	⨓	line integration with semicircular path around pole	prefix	310	1	2	largeop, movablelimits, symmetric
&#x2A14;	⨔	line integration not including the pole	prefix	310	1	2	largeop, movablelimits, symmetric
&#x2A15;	⨕	integral around a point operator	prefix	310	1	2	largeop, symmetric
&#x2A16;	⨖	quaternion integral operator	prefix	310	1	2	largeop, symmetric
&#x2A17;	⨗	integral with leftwards arrow with hook	prefix	310	1	2	largeop, symmetric
&#x2A18;	⨘	integral with times sign	prefix	310	1	2	largeop, symmetric
&#x2A19;	⨙	integral with intersection	prefix	310	1	2	largeop, symmetric
&#x2A1A;	⨚	integral with union	prefix	310	1	2	largeop, symmetric
&#x2A1B;	⨛	integral with overbar	prefix	310	1	2	largeop, symmetric
&#x2A1C;	⨜	integral with underbar	prefix	310	1	2	largeop, symmetric
&#x22C3;	⋃	n-ary union	prefix	320	1	2	largeop, movablelimits, symmetric
&#x2A03;	⨃	n-ary union operator with dot	prefix	320	1	2	largeop, movablelimits, symmetric
&#x2A04;	⨄	n-ary union operator with plus	prefix	320	1	2	largeop, movablelimits, symmetric
&#x22C0;	⋀	n-ary logical and	prefix	330	1	2	largeop, movablelimits, symmetric
&#x22C1;	⋁	n-ary logical or	prefix	330	1	2	largeop, movablelimits, symmetric
&#x22C2;	⋂	n-ary intersection	prefix	330	1	2	largeop, movablelimits, symmetric
&#x2A00;	⨀	n-ary circled dot operator	prefix	330	1	2	largeop, movablelimits, symmetric
&#x2A02;	⨂	n-ary circled times operator	prefix	330	1	2	largeop, movablelimits, symmetric
&#x2A05;	⨅	n-ary square intersection operator	prefix	330	1	2	largeop, movablelimits, symmetric
&#x2A06;	⨆	n-ary square union operator	prefix	330	1	2	largeop, movablelimits, symmetric
&#x2A07;	⨇	two logical and operator	prefix	330	1	2	largeop, movablelimits, symmetric

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2A08;	⨈	two logical or operator	prefix	330	1	2	largeop, movablelimits, symmetric
&#x2A09;	⨉	n-ary times operator	prefix	330	1	2	largeop, movablelimits, symmetric
&#x2AFC;	⫼	large triple vertical bar operator	prefix	330	1	2	largeop, movablelimits, symmetric
&#x2AFF;	⫿	n-ary white vertical bar	prefix	330	1	2	largeop, movablelimits, symmetric
&#x2240;	≀	wreath product	infix	340	4	4	
&#x220F;	∏	n-ary product	prefix	350	1	2	largeop, movablelimits, symmetric
&#x2210;	∐	n-ary coproduct	prefix	350	1	2	largeop, movablelimits, symmetric
&#x2229;	∩	intersection	infix	350	4	4	
&#x222A;	∪	union	infix	350	4	4	
*	*	asterisk	infix	390	3	3	
.	.	full stop	infix	390	3	3	
&#xD7;	×	multiplication sign	infix	390	4	4	
&#x2022;	•	bullet	infix	390	4	4	
&#x2043;	⁃	hyphen bullet	infix	390	4	4	
&#x2062;		invisible times	infix	390	0	0	
&#x22A0;	⊠	squared times	infix	390	4	4	
&#x22A1;	⊡	squared dot operator	infix	390	4	4	
&#x22C5;	⋅	dot operator	infix	390	4	4	
&#x2A2F;	⨯	vector or cross product	infix	390	4	4	
&#x2A3F;	⨿	amalgamation or coproduct	infix	390	4	4	
&#xB7;	·	middle dot	infix	400	4	4	
&#x2297;	⊗	circled times	infix	410	4	4	
%	%	percent sign	infix	640	3	3	
\	\	reverse solidus	infix	650	0	0	
&#x2216;	∖	set minus	infix	650	4	4	
/	/	solidus	infix	660	1	1	
&#xF7;	÷	division sign	infix	660	4	4	
&#x2220;	∠	angle	prefix	670	0	0	
&#x2221;	∡	measured angle	prefix	670	0	0	
&#x2222;	∢	spherical angle	prefix	670	0	0	
&#xAC;	¬	not sign	prefix	680	2	1	
&#x2299;	⊙	circled dot operator	infix	710	4	4	
&#x2202;	∂	partial differential	prefix	740	2	1	
&#x2207;	∇	nabla	prefix	740	2	1	
**	**	multiple character operator: **	infix	780	1	1	
&lt;>	<>	multiple character operator: <>	infix	780	1	1	
^	^	circumflex accent	infix	780	1	1	
&#x2032;	′	prime	postfix	800	0	0	
&#x266D;	♭	music flat sign	postfix	800	0	2	
&#x266E;	♮	music natural sign	postfix	800	0	2	
&#x266F;	♯	music sharp sign	postfix	800	0	2	
!	!	exclamation mark	postfix	810	1	0	
!!	!!	multiple character operator: !!	postfix	810	1	0	

Character	Glyph	Name	form	priority	lspace	rspace	Properties
//	//	multiple character operator: //	infix	820	1	1	
@	@	commercial at	infix	825	1	1	
?	?	question mark	infix	835	1	1	
&#x2145;	ⅅ	double-struck italic capital d	prefix	845	2	1	
&#x2146;	ⅆ	double-struck italic small d	prefix	845	2	0	
&#x221A;	√	square root	prefix	845	1	1	stretchy
&#x221B;	∛	cube root	prefix	845	1	1	
&#x221C;	∜	fourth root	prefix	845	1	1	
&#x2061;		function application	infix	850	0	0	
"	"	quotation mark	postfix	880	0	0	accent
&amp;	&	ampersand	postfix	880	0	0	
'	'	apostrophe	postfix	880	0	0	accent
++	++	multiple character operator: ++	postfix	880	0	0	
--	--	multiple character operator: --	postfix	880	0	0	
^	^	circumflex accent	postfix	880	0	0	stretchy, accent
_	_	low line	postfix	880	0	0	stretchy, accent
`	`	grave accent	postfix	880	0	0	accent
~	~	tilde	postfix	880	0	0	stretchy, accent
&#xA8;	¨	diaeresis	postfix	880	0	0	accent
&#xAA;	ª	feminine ordinal indicator	postfix	880	0	0	accent
&#xAF;	¯	macron	postfix	880	0	0	stretchy, accent
&#xB0;	°	degree sign	postfix	880	0	0	
&#xB2;	²	superscript two	postfix	880	0	0	accent
&#xB3;	³	superscript three	postfix	880	0	0	accent
&#xB4;	´	acute accent	postfix	880	0	0	accent
&#xB8;	¸	cedilla	postfix	880	0	0	accent
&#xB9;	¹	superscript one	postfix	880	0	0	accent
&#xBA;	º	masculine ordinal indicator	postfix	880	0	0	accent
&#x2C6;	ˆ	modifier letter circumflex accent	postfix	880	0	0	stretchy, accent
&#x2C7;	ˇ	caron	postfix	880	0	0	stretchy, accent
&#x2C9;	ˉ	modifier letter macron	postfix	880	0	0	stretchy, accent
&#x2CA;	ˊ	modifier letter acute accent	postfix	880	0	0	accent
&#x2CB;	ˋ	modifier letter grave accent	postfix	880	0	0	accent
&#x2CD;	ˍ	modifier letter low macron	postfix	880	0	0	stretchy, accent
&#x2D8;	˘	breve	postfix	880	0	0	accent
&#x2D9;	˙	dot above	postfix	880	0	0	accent
&#x2DA;	˚	ring above	postfix	880	0	0	accent
&#x2DC;	˜	small tilde	postfix	880	0	0	stretchy, accent
&#x2DD;	˝	double acute accent	postfix	880	0	0	accent
&#x2F7;	˷	modifier letter low tilde	postfix	880	0	0	stretchy, accent
&#x302;	̂	combining circumflex accent	postfix	880	0	0	stretchy, accent
&#x311;	̑	combining inverted breve	postfix	880	0	0	accent
&#x201A;	‚	single low-9 quotation mark	postfix	880	0	0	accent
&#x201B;	‛	single high-reversed-9 quotation mark	postfix	880	0	0	accent
&#x201E;	„	double low-9 quotation mark	postfix	880	0	0	accent
&#x201F;	‟	double high-reversed-9 quotation mark	postfix	880	0	0	accent
&#x2033;	″	double prime	postfix	880	0	0	accent
&#x2034;	‴	triple prime	postfix	880	0	0	accent

Character	Glyph	Name	form	priority	lspace	rspace	Properties
&#x2035;	‵	reversed prime	postfix	880	0	0	accent
&#x2036;	‶	reversed double prime	postfix	880	0	0	accent
&#x2037;	‷	reversed triple prime	postfix	880	0	0	accent
&#x203E;	‾	overline	postfix	880	0	0	stretchy, accent
&#x2057;	⁗	quadruple prime	postfix	880	0	0	accent
&#x2064;		invisible plus	infix	880	0	0	
&#x20DB;	⃛	combining three dots above	postfix	880	0	0	accent
&#x20DC;	⃜	combining four dots above	postfix	880	0	0	accent
&#x23B4;	⎴	top square bracket	postfix	880	0	0	stretchy, accent
&#x23B5;	⎵	bottom square bracket	postfix	880	0	0	stretchy, accent
&#x23DC;	⏜	top parenthesis	postfix	880	0	0	stretchy, accent
&#x23DD;	⏝	bottom parenthesis	postfix	880	0	0	stretchy, accent
&#x23DE;	⏞	top curly bracket	postfix	880	0	0	stretchy, accent
&#x23DF;	⏟	bottom curly bracket	postfix	880	0	0	stretchy, accent
&#x23E0;	⏠	top tortoise shell bracket	postfix	880	0	0	stretchy, accent
&#x23E1;	⏡	bottom tortoise shell bracket	postfix	880	0	0	stretchy, accent
_	_	low line	infix	900	1	1	

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 Content MathML 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 Markup).

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 Summary of Presentation Elements

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 Operator, Fence, Separator or Accent <`mo`>.

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 Token Elements and Section 3.5.5 Alignment Markers `<maligngroup/>`, `<malignmark/>`, or the empty elements `none` and `mprescripts` defined in Section 3.4.7 Prescripts and Tensor Indices `<mmultiscripts>`, `<mprescripts/>`, `<none/>`). 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 Content Markup 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 Operator, Fence, Separator or Accent `<mo>`).

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.

Presentation elements

MathML tags and entities intended to express the syntactic structure of mathematical notation (defined in Chapter 3 Presentation Markup).

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 schemata). 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 Enlivening Expressions) 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 aforementioned 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 Space `<mspace/>`.

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 Presentation Markup; the ones that use other terms defined here occur mainly in Section 3.2.5 Operator, Fence, Separator or Accent `<mo>` and in Section 3.7 Enlivening Expressions.

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 The Top-Level `<math>` Element).

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.

E Working Group Membership and Acknowledgments (Non-Normative)

E.1 The Math Working Group Membership

The present W3C Math Working Group (2012–2013) is co-chaired by David Carlisle of NAG and Patrick Ion of the AMS; Patrick Ion and and Robert Miner of Design Science were co-chairs 2006-2011. Contact the co-chairs about membership in the Working Group. For the current membership see the W3C Math home page.

Robert Miner, whose leadership and contributions were essential to the development of the Math Working Group and MathML from their beginnings, died tragically young in December 2011.

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, Integre Technical Publishing Co., Inc., Albuquerque NM, 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, Edinburgh, UK
- Bruce Miller, National Institute of Standards and Technology (NIST), Gaithersburg MD, USA
- Robert Miner, Design Science Inc., Long Beach CA, USA
- Chris 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 repurposed 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 Characters, Entities and Fonts, 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.

F Changes (Non-Normative)

F.1 Changes between MathML 3.0 First Edition and Second Edition

- Changes to the Frontmatter
  - Changes to **Editors and Authors** to acknowlege the death of Robert Miner.
  - Changes to **Abstract** to highlight MathML may be used in HTML as well as XML.
  - Update the **Status**.
- Changes to the Chapter 1 Introduction
  - Changes to Section 1.3 Overview to highlight MathML may be used in HTML as well as XML.
- Changes to the Chapter 2 MathML Fundamentals
  - Changes to Section 2.1.1 General Considerations to highlight MathML may be used in HTML as well as XML.
  - Add element markup to heading in Section 2.2 The Top-Level `<math>` Element.
  - Changes to Section 2.1.2 MathML and Namespaces the `xmlns` syntax for namespaces only applies to the XML serialisation.
  - Changes to Section 2.1.5.2 Length Valued Attributes to clarify that values specified with a `%` or no unit are multiples of a *reference* value, which may differ from the *default* value used when the value is not specified.
  - Changes to *namedspace* in Section 2.1.5.2 Length Valued Attributes some attribute values such as "thinmathspace" were marked up as attribute *names*. (This affected formatting and also the index Section I.2 MathML Attributes).
  - Additional paragraph describing global document property defaults in Section 2.1.5.4 Default values of attributes
- Changes to the Chapter 3 Presentation Markup
  - Refer to "characters" rather than "MathML Characters" in Section 3.1 Introduction.
  - Delete the note about bidi in HTML in Section 3.1.5.2 Bidirectional Layout in Token Elements (As there are proposals to change the HTML behavior).
  - Corrected mistaken refererence to `mtext`, replaced by reference to `mo` in Section 3.1.8.2 Warning: spacing should not be used to convey meaning
  - Refer to HTML rather than XHTML in Section 3.1.10 Mathematics style attributes common to presentation elements
  - Modify heading of Section 3.2.1 Token Element Content Characters, `<mglyph/>`.
  - Note in Section 3.2.1.2 Using images to represent symbols `<mglyph/>` that the requirement to use `src` and `alt` is not enforced by the schema.
  - New section Section 3.2.2.2 Embedding HTML in MathML detailing the use of HTML elements on MathML token elements
  - Use U+2026 rather than `. . .` in the example in Section 3.2.3 Identifier `<mi>`.
  - Use percentage lengths rather than unitless lengths in examples in Section 3.2.5.8 Stretching of operators, fences and accents and Section 3.3.2 Fractions `<mfrac>`
  - Reference the Arabic Mathematical Symbols block in describing `mathvariant` in Section 3.2.2 Mathematics style attributes common to token elements.
  - Do not specify that Math defaults may be set by using attributes in the MathML Namespace on the containing document, leave the mechanism open. Section 3.2.5 Operator, Fence, Separator or Accent `<mo>`

  - Changes to Section 3.2.5.2 Attributes to clarify defaults may be specified in any containing document.
  - Changes to Section 3.2.5.2.1 Dictionary-based attributes to clarify the interpretation of `maxsize`, `minsize` and `symmetric` values.
  - Changes to Section 3.2.5.2.3 Indentation attributes to clarify behaviour if `indenttarget` results in an unachievable alignment specification.
  - Changes to the examples in Section 3.2.5.5 Invisible operators so that each example is rendered as a separate `math` expression.
  - Suggest CSS Counters as a possible mechanism for equation numbering in Section 3.5.3 Labeled Row in Table or Matrix `<mlabeledtr>`
  - Minor improvements to the markup in Section 3.3.4 Style Change `<mstyle>`.
  - Minor improvements to the markup in Section 3.3.9 Enclose Expression Inside Notation `<menclose>`.
  - Changes to the attribute table Section 3.3.6.2 Attributes To clarify that unitless lengths are allowed on `mpadded`, meaning, as usual, a multiplier of the stated default. Note that this change also affects the `mpadded-length` grammar uin the extracted schema.
  - Explictly list mprescripts and none in heading for Section 3.4.7 Prescripts and Tensor Indices `<mmultiscripts>`, `<mprescripts/>`, `<none/>`.
  - Changes to Section 3.5.1.2 Attributes to clarify that `displaystyle` defaults to "false" and to clarify the wording describing percentage values for `width`.
  - Minor improvements to the markup in Section 3.6.1 Stacks of Characters `<mstack>`.
  - Editorial improvements to Section 3.6.8.1 Addition and Subtraction.
  - Modify the markup in the examples in Section 3.6.8.4 Repeating decimal so that MathML renderings are shown in some versions of this specification.
  - Note that attributes in other namespaces are not available in HTML in Section 3.7.1 Bind Action to Sub-Expression `<maction>`

- Changes to the Chapter 4 Content Markup
  - Add element markup to heading in Section 4.2.1.1 Rendering `<cn>`, `<sep/>` - Represented Numbers .
  - Add syntax table for qualifier elements in Section 4.3.3.1 Uses of `<domainofapplication>`, `<interval>`, `<condition>`, `<lowlimit>` and `<uplimit>` and Section 4.3.3.2 Uses of `<degree>`.
  - Modify the text in Section 4.1.5 Content MathML Concepts to clarify the role of the **Qualifier** row of syntax tables. (AM)
  - Spurious `apply` removed from the "0" case in the example in Section 4.4.1.9 Piecewise declaration `<piecewise>`, `<piece>`, `<otherwise>`.
  - Changes to Rewrite: partialdiffdegree The expression *expression-in-x1-xk* was rewritten to *A*. (AM)
  - Additional note added to the mathmltypes description clarifying that "complex" should be taken as an alias for "complex-cartesian" when rewriting to Strict Content MathML. (AM)
  - Changes to s_data1.mean, s_dist1.mean, s_dist1.moment and s_data1.moment examples to use new values for &lang; and &rang; so the result is in Unicode NFC form.
  - Changes to markup of syntax tables in Section 4.2.5 Function Application `<apply>` and Section 4.2.7.1 The `share` element to avoid redundant colspans, which make the html5 version invalid.
  - Clarify the behavior of qualifiers in Step 4b of the rewrite to Strict Content MathML. (AM)

  - Clarify that the types of the *arguments* are used to distinguish between set and multiset use of the `set` constructor in Section 4.3.4.1.2 Rewriting to Strict Content MathML and Section 4.3.4.2.2 Rewriting to Strict Content MathML. (AM)
  - Fix spelling in Section 4.4.2.16 Not `<not/>`.
  - Fix spelling in Section 4.4.3.1 Equals `<eq/>`.
  - Split Section 4.4.7.1 Common trigonometric functions `<sin/>`, `<cos/>`, `<tan/>`, `<sec/>`, `<csc/>`, `<cot/>` into separate sections Section 4.4.7.1 Common trigonometric functions `<sin/>`, `<cos/>`, `<tan/>`, `<sec/>`, `<csc/>`, `<cot/>` , Section 4.4.7.1 Common trigonometric functions `<sin/>`, `<cos/>`, `<tan/>`, `<sec/>`, `<csc/>`, `<cot/>` , Section 4.4.7.2 Common inverses of trigonometric functions `<arcsin/>`, `<arccos/>`, `<arctan/>`, `<arcsec/>`, `<arccsc/>`, `<arccot/>`, Section 4.4.7.3 Common hyperbolic functions `<sinh/>`, `<cosh/>`, `<tanh/>`, `<sech/>`, `<csch/>`, `<coth/>`, Section 4.4.7.4 Common inverses of hyperbolic functions `<arcsinh/>`, `<arccosh/>`, `<arctanh/>`, `<arcsech/>`, `<arccsch/>`, `<arccoth/>` , add new presentation images for arcsin.
  - Add element markup to heading in Section 4.4.7.7 Logarithm `<log/>`, `<logbase>` .
  - Minor rearrangement of heading in Section 4.4.8.6 Moment `<moment/>`, `<momentabout>`
  - Add syntax table for deprecated elements in Section 4.5.1 Declare `<declare>`, Section 4.5.3 Relation `<fn>` and Section 4.5.2 Relation `<reln>`.
- Changes to Chapter 5 Mixing Markup Languages for Mathematical Expressions.
  - Changes to Section 5.1.1 Annotation elements to highlight MathML may be used in HTML as well as XML.
  - Add additional note warning namespace extensibility example not applicable to HTML.
  - Add additional note warning namespace extensibility example not applicable to HTML.
  - Add additional note warning namespace extensibility example not applicable to HTML.
  - Additional section Section 5.2.3.3 Using `annotation-xml` in HTML documents detailing the use of `annotation-xml`in HTML documents
  - Show tag markup around element names in section headings in semantics, annotation and annotation-xml.
- Changes to Chapter 6 Interactions with the Host Environment.
  - Editorial wording changes in Section 6.4 Combining MathML and Other Formats.
  - Editorial wording changes in Section 6.5 Using CSS with MathML.
  - Changes to wording on namespace use in Section 6.1 Introduction.
  - Additional section Section 6.2.2 Recognizing MathML in HTML.
  - Remove XML Declaration and `mml` namespace prefix from the examples in Section 6.3.4 Examples.
  - Delete recommendation to use prefixed element names in XHTML in Section 6.4.1 Mixing MathML and XHTML.
  - Split HTML into a separate section from other non-XML use Section 6.4.3 Mixing MathML and HTML and Section 6.4.2 Mixing MathML and non-XML contexts
  - Remove the reference, Layout engines that lack native MathML support, to [MathMLforCSS] in Chapter 6 Interactions with the Host Environment.
- Changes to Chapter 7 Characters, Entities and Fonts.
  - Change the DTD description in Section 7.3 Entity Declarations to reference the Combined HTML MathML entity set rather than the legacy ISO entity sets. This does not change any existing definition, but adds the following 38 entity definitions: `&QUOT;` (U+0022), `&AMP;` (U+0026), `&LT;` (U+003C), `&GT;` (U+003E), `&COPY;` (U+00A9), `&REG;` (U+00AE), `&Alpha;` (U+0391), `&Beta;` (U+0392), `&Epsilon;` (U+0395), `&Zeta;` (U+0396), `&Eta;` (U+0397), `&Iota;` (U+0399), `&Kappa;` (U+039A), `&Mu;` (U+039C), `&Nu;` (U+039D), `&Omicron;` (U+039F), `&Rho;` (U+03A1),

&Tau; (U+03A4), &Chi; (U+03A7), &epsilon; (U+03B5), &omicron; (U+03BF), &sigmaf; (U+03C2), &thetasym; (U+03D1), &upsih; (U+03D2), &zwnj; (U+200C), &zwj; (U+200D), &lrm; (U+200E), &rlm; (U+200F), &sbquo; (U+201A), &bdquo; (U+201E), &lsaquo; (U+2039), &rsaquo; (U+203A), &oline; (U+203E), &frasl; (U+2044), &euro; (U+20AC), &TRADE; (U+2122), &alefsym; (U+2135), &crarr; (U+21B5).

- Changes to Appendix A Parsing MathML.
  - Modify the schema regular expression to allow the deprecated unitless length attributes.
  - The schema now enforces a mandatory space and optional minus sign before *rownumber* in the `align` attribute of `mtable` and `mstack`.
  - Modify the schema (including DTD and XSD versions) to include the attributes listed in Section 3.2.5.2.3 Indentation attributes on `mspace` to match the text description in Section 3.2.7 Space `<mspace/>`.
  - Modify the schema to include the Common Attributes such as `id` on `bvar` and `cn` to match the attribute tables in the text
  - Modify the regular expressions used for `mpadded-length` and `length` so that there must be at most one `.` and at least one digit. (FW)
  - New sections: Section A.5 Parsing MathML in XHTML and Section A.6 Parsing MathML in HTML.
- Changes to Appendix C Operator Dictionary.
  - Add entries for the characters listed in Section 7.7.2 Pseudo-scripts.
- Changes to Appendix E Working Group Membership and Acknowledgments.
  - Changes to Section E.1 The Math Working Group Membership to note the death of Robert Miner.
- Changes to Appendix G Normative References.
  - Make [HTML5] normative.
  - Update the references to current versions, [Bidi], [Entities], [ISO10646], [Normal], [RELAX-NG], [Unicode].
- Changes to Appendix H References.
  - Update the references to current versions, [AAP-math], [Abramowitz1977], [CSS21], [ISO-12083], [XLink].
- Changes to Appendix I Index.
  - Changes to Section I.2 MathML Attributes.

F.2 Changes between MathML 2.0 Second Edition and MathML 3.0

- Changes to Chapter 2 MathML Fundamentals.
  - The attribute `href` added to the common MathML attributes, Section 2.1.6 Attributes Shared by all MathML Elements to allow hypertext links.
  - Additional attributes added to the `math` element, see Section 2.2.1 Attributes.
- Changes to Chapter 3 Presentation Markup.
  - Introduced mechanisms for controlling the Directionality of layout, as described in Section 3.1.5 Directionality.
  - Introduced mechanisms for controlling linebreaking Section 3.1.7 Linebreaking of Expressions.
  - Extended `mglyph` to support general image inclusion, Section 3.2.1.2 Using images to represent symbols `<mglyph/>`.

  - The facilities for adjusting spacing with `mpadded` have been extended and rationalised, Section 3.3.6 Adjust Space Around Content `<mpadded>`.
  - Introduced new presentation elements for elementary math layouts, Section 3.6 Elementary Math: `mstack`, `mlongdiv`, `msgroup`, `msrow`, `mscarries`, `mscarry`, and `msline`.
- Changes to Chapter 4 Content Markup.
  - 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 Mixing Markup Languages for Mathematical Expressions.
- New Chapter: Chapter 6 Interactions with the Host Environment.
- Changes to Chapter 7 Characters, Entities and Fonts.

  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 Parsing MathML.
  - The Normative version of the grammar is now expressed in Relax NG, with DTD and XSD versions being derived.
  - Three MathML 1 attrbutes on `math` that were deprecated and undocumented in MathML2 but retained in the MathML2 DTD have been removed. `name` (use `id` instead), `baseline` and `type` (These are not used by any known implementation, so can be removed.) See Chapter 7 of [MathML1].
- New Appendix: Appendix B Media Types Registrations.
- Changes to Appendix C Operator Dictionary.

  The Operator Dictionary table has been updated and rationalised and presented in a new format.
- MathML DOM

  The 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.

G Normative References

Bidi

Mark Davis; *Unicode Bidirectional Algorithm*, Unicode Standard Annex #9, Unicode 6.3.0 2013-09-24. (http://www.unicode.org/reports/tr9/)

Entities

David Carlisle and Patrick Ion (editors); *XML Entity Definitions for Characters (2nd Edition)*, W3C Recommendation 10 April 2014. (http://www.w3.org/TR/xml-entity-names/)

HTML4

Dave Raggett, Arnaud Le Hors, and Ian Jacobs (editors); *HTML 4.01 Specification*, W3C Recommendation, 24 December 1999. (http://www.w3.org/TR/html401/)

HTML5

Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, Silvia Pfeiffer, Ian Hickson; *HTML 5, A vocabulary and associated APIs for HTML and XHTML* W3C Candidate Recommendation 04 February 2014 (http://www.w3.org/TR/html5/)

IRI

M. Duerst and M. Suignard; *Internationalized Resource Identifiers (IRIs)*, January 2005 (http://www.ietf.org/rfc/rfc3987.txt). See also the proposed update http://tools.ietf.org/html/draft-duerst-iri-bis-07.

ISO10646

Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 2. *ISO/IEC 10646:2012, Information technology -- Universal Coded Character Set (UCS)*. International Standards Organization, Geneva, Switzerland, 2012.

Namespaces

Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin, and Henry S. Thompson (editors); *Namespaces in XML*, W3C Recommendation, 8 December 2009. (http://www.w3.org/TR/REC-xml-names/)

Normal

Mark Davis, Ken Whistler, and Martin Dürst; *Unicode Normalization Forms*, Unicode Standard Annex #15, Unicode 6.3.0 2013-09-20. (http://www.unicode.org/reports/tr15/)

OpenMath2004

S. Buswell, O. Caprotti, D. P. Carlisle, M. C. Dewar, M. Gaëtano and M. Kohlhase (editors); *The OpenMath Standard Version 2.0*, The OpenMath Society, 30 June 2004. (http://www.openmath.org/standard/om20-2004-06-30/)

RELAX-NG

ISO/IEC JTC 1/SC 34; *Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG 2008*, International Organization for Standardization, 15 December 2008. (http://relaxng.org/spec-20011203.html)

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)

Unicode

The Unicode Consortium. *The Unicode Standard, Version 6.3.0*, The Unicode Consortium, 2012. ISBN 978-1-936213-07-8. (http://www.unicode.org/versions/Unicode6.3.0/)

XHTML

Steven Pemberton, et al.; *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition): A Reformulation of HTML 4 in XML 1.0*, W3C Recommendation, 26 January 2000, revised 1 August 2002. (http://www.w3.org/TR/xhtml1/)

XML

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau (editors); *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, W3C Recommendation, 26 November 2008. (http://www.w3.org/TR/xml/)

H 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.

Abramowitz1977

Abramowitz, Milton, Irene A. Stegun (editors); *Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*. Dover Publications Inc., December 1977, ISBN: 0-4866-1272-4.

CSS21

Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie (editors); *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*, W3C Recommendation 07 June 2011. (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).

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.

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.

IEEE754

IEEE *IEEE Standard for Floating-Point Arithmetic* (http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4610935)

ISO-12083

ISO 12083:1994; *ISO 12083 DTD* Information and Documentation - Electronic Manuscript Preparation and Markup. International Standards Organization, Geneva, Switzerland, 1994.

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 (Second Edition)* W3C Recommendation 21 October 2003 (http://www.w3.org/TR/MathML2)

MathMLTypes

Stan Devitt, Michael Kohlhase, Max Froumentin (editors); *Structured Types in MathML 2.0*, 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 Recommendation 07 June 2011 (http://www.w3.org/TR/mathml-for-css)

Modularization

Daniel Austin, Subramanian Peruvemba, Shane McCarron, Masayasu Ishikawa, Mark Birbeck (editors); *XHTML[tm] Modularization 1.1 - Second Edition*, World Wide Web Consortium Recommendation, 29 July 2010. (http://www.w3.org/TR/2010/REC-xhtml-modularization-20100729)

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) .

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.

RelaxNGBook

Eric van der Vlist; *RELAXNG: A simple schema language for XML* O'Reilly 2004

SVG1.1

Dean Jackson, Jon Ferraiolo, Jun Fujisawa, eds. *Scalable Vector Graphics (SVG) 1.1 Specification (Second Edition)* W3C Recommendation, 16 August 2011 (http://www.w3.org/TR/2011/REC-SVG11-20110816/)

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, Norman Walsh (editors); *XML Linking Language (XLink) Version 1.1*, W3C Recommendation 06 May 2010. (http://www.w3.org/TR/2010/REC-xlink11-20100506/)

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/2004/REC-xmlschema-2-20041028/)

XMLSchemas

David C. Fallside, editor; *XML Schema Part 0: Primer Second Edition*, W3C Recommendation 28 October 2004. (http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/)

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)

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/

I Index (Non-Normative)

I.1 MathML Elements

References to sections in which an element is defined are marked in bold.

I.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.

This document has been amended as follows:

- The errata for MathML 3 Second Edition was reflected on this document.
- The page number reference was given to Table of Contents.
- MathML rendering images (GIF and PNG) were changed to MathML. However, some sample codes in the specification are not rendered as expected. They are as follows:
    - Section 3.3.2 Fractions: the MathML example code is not rendered like the first sample image. In this document, MathML code is being rendered.
    - Section 3.4.6 Underscript-overscript Pair: movablelimits="true" should be specified for the integral examples.
    - The last example of Section 3.6.8.4 Repeating decimal needs to specify charalign="center".
    - Section 4.4.4.1 Integral: Integrals in Sample Presentation are specified by <mi>. They should be <mo>.
- Some images were changed to SVG.
- Some images of Sample Presentation were missing. They were supplemented.
    - Section 4.4.4.2 Differentiation
    - Section 4.4.7.3 Common hyperbolic functions
    - Section 4.4.7.4 Common inverses of hyperbolic functions
- The following typos were corrected:
    - Section 3.1.8 Warning about fine-tuning of presentation: the ~~the~~ same MathML
    - Section 3.4.7.1 Description: the ~~the~~ base of the mmultiscripts
    - Section 3.6.7.2 Attributes: Specifies the ~~the~~ number of columns
    - Section 4.2.5.2 Rendering Applications: the ~~the~~ sample rendering
    - Section 6.3.2 Recommended Behaviors when Transferring: in the ~~the~~ HTTP Accept header
    - Section 6.4.2 Mixing MathML and non-XML contexts: <mfrac> and <mi~~"~~>
    - Appendix A.2.6 MathML as a module in a RelaxNG Schema: to collect the ~~the~~ content models
- Section 7.5 Mathematical Alphanumeric Symbols, Section 7.7 Anomalous Mathematical Characters: Unnecessary comments in MathML were removed.
    - <mi>&#x1D44E;<!--MATHEMATICAL ITALIC SMALL A-->~~<!--MATHEMATICAL ITALIC SMALL A-->~~ </mi>
    - <mi>&#x1D504;<!--MATHEMATICAL FRAKTUR CAPITAL A-->~~<!--BLACK-LETTER CAPITAL A-->~~</mi>
    - <mo>&#x2032;<!--PRIME-->~~<!--PRIME-->~~</mo>
- Appendix H References (Non-Normative): Removed the empty link.
- Appendix I.2 MathML Attributes: Duplicate index items existed. They were removed.
    - indentalignfirst
    - indentalignlast
    - indentshift
    - indentshiftfirst
- Appendix I Index were changed to refer to page number.