Mathematical Markup Language (MathML) Version 3.0

W3C Working Draft 09 April 2008

This version: http://www.w3.org/TR/2008/WD-MathML3-20080409/
Latest MathML 3 version: http://www.w3.org/TR/MathML3/
Latest MathML Recommendation: http://www.w3.org/TR/MathML/
Previous version:
http://www.w3.org/TR/2007/WD-MathML3-20071005/
Editors: David Carlisle(NAG)
Patrick Ion(Mathematical Reviews, American Mathematical Society)
Robert Miner(Design Science, Inc.)
Principal Authors: Ron Ausbrooks, Bert Bos, Olga Caprotti, David Carlisle, Giorgi Chavchanidze, Ananth Coorg, St\'ephane Dalmas, Stan Devitt, Sam Dooley, Margaret Hinchcliffe, Patrick Ion, Michael Kohlhase, Azzeddine Lazrek, Dennis Leas, Paul Libbrecht, Manolis Mavrikis, Bruce Miller, Robert Miner, Murray Sargent, Kyle Siegrist, Neil Soiffer, Stephen Watt, Mohamed Zergaoui

Please refer to the errata for this document, http://www.w3.org/2007/04/mathml3-errata.html which may include some normative corrections.

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

See also http://www.w3.org/2007/04/MathML3-translations for **translations** of this document.

Copyright © 1998-2008 W3C® (MIT, ERCIM, Keio), All Rights Reserved.W3C liability, trademark, document use and software licensing rules apply.

Abstract

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

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

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

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

Status of this document

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

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

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

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

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

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

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

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

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

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

- With the second Working Draft, much of the non-normative explication that formerly was found in Chapters 1 and 2, and many examples from elsewhere in the previous MathML specifications, were removed from the MathML3 specification and incorporated into a MathML Primer being prepared as a separate document. It is expected this will help the use of this formal MathML3 specification as a reference document in implementations, and offer the new user better help in understanding MathML's deployment. The remaining content of Chapters 1 and 2 is being edited to reflect the changes elsewhere in the document, and in the rapidly evolving Web environment. Some of their text used to go back to early days of the Web and XML, and its explanations are now commonplace.

- Chapter 3, on presentation-oriented markup, in this draft adds new material on linebreaking and on markup for elementary math notations. Material introduced in the last draft revising the `mpadded` and `maction` elements has been further revised as a result of active discussion. It is possible it may undergo further modification. In addition, the layout of schemata such as that for long division and its associated `mcolumn` element have been carefully revised. Earlier work, as recorded in the W3C Note Arabic mathematical notation, has allowed clarification of the relationship with bidirectional text and examples with RTL text have been added.
- Chapter 4, on content-oriented markup, contains major changes and additions in this Working Draft. The meaning of the actual content remains as before in principle, but a lot of work has been done on expressing it better. The text of this chapter is generated by filtered extraction from XML Content Dictionaries written in accordance with OpenMath. The details of the Content Dictionary format have been further specified and the generation procedure improved. It is expected that the Content Dictionaries will become a separate joint publication of the W3C and OpenMath referenced by the MathML3 specification. The Content Dictionaries are now publicly available in draft and much work has already been done on refining them. Their format is given in Chapter 8.
- Chapter 5 is being refined as its purpose has been further clarified. This chapter deals with interrelations of parts of the MathML specification, especially with presentation and content markup.
- Chapter 6 has been rewritten and reorganized to reflect the new situation in regard to Unicode, and the changed W3C context with regard to named character entities. The new W3C specification of Entity Definitions for Characters in XML, which incorporates those used for mathematics is becoming a public working draft [Entities]. It is expected that some new ancillary tables will be provided that reflect requests the Math WG has received.
- Chapter 7 has been restored with a new and clearer purpose. This chapter looks outward to the larger world in which MathML must function.
- Chapter 8 now specifies the format of MathML3 Content Dictionaries, as previously handled more briefly in sections 4.5 and 4.6. The DOM for MathML, previously in a chapter at this point, is being prepared as a separate specification.
- The Appendices, of which there are eight shown, have not been fully reworked. Eventually what amount to revisions of the present appendices A, F, G, H, I and J are all that are expected to remain. Appendix A now contains the new RelaxNG schema for MathML3 as well as discussion of MathML3 DTD issues.

Contents

Open Issues

fund: MathML Fundamentals, presm: Presentation Markup, contm: Content Markup, world-interactions: MathML interactions with the Wide World, mcds: MathML3 Content Dictionaries, parsing: Parsing MathML, oper-dict: Operator Dictionary, changes: Changes

Chapter 1

Introduction

1.1 Mathematics and its Notation

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

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

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

It is remarkable how widespread the current conventions of mathematical notations have become. The general two-dimensional layout, and most of the same symbols, are used in all modern mathematical communications, whether the participants are European, writing left-to-right, or Middle-Eastern, writing right-to-left. Of course, conventions for the symbols used, particularly those naming functions and variables, may tend to favor a local language and script. The largest variation from the most common is a form used in some Arabic-speaking communities which lays out the entire mathematical notation from right-to-left, roughly in mirror image of the European tradition.

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

1.2 Origins and Goals

1.2.1 Design Goals of MathML

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

MathML should ideally:

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

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

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

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

The extent to which these goals are ultimately met depends on the cooperation and support of browser vendors, and other software developers. The W3C Math Working Group has continued to work with other working groups of the W3C, and outside the W3C, to ensure that the needs of the scientific community will be met in the future. MathML 2 and it 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.

Chapter 2

MathML Fundamentals

Issue (): The current chapter is based largely from MathML2, and needs further work in later drafts. The outline of the contents is settled but, to a certain extent, the tone to be set depends on details still being settled by the Working Group.

2.1 MathML Syntax and Grammar

2.1.1 MathML Syntax and Grammar

MathML is an application of [XML], Extensible Markup Language, and as such it is governed by the rules of XML syntax. XML syntax is a notation for rooted labeled planar trees. Planarity means that the children of a node may be viewed as given a natural order and MathML depends on this. MathML also specifies some syntax and grammar rules in addition to the general rules it inherits as an XML application. These rules allow MathML to encode a great deal more information than would ordinarily be possible with pure XML, without introducing many more elements, and using a substantially more structure. The grammar of MathML is now specified by using a RelaxNG Schema. In other words, the details of using tags, attributes, entity references and the like are defined in the XML language specification, and the details about MathML elements and attribute names, which elements can be nested inside each other, and their possible relationships are specified in the MathML Schema. This is in Appendix A.

The grammatical aspects of MathML2 were in part specified by a DTD, or Document Type Definition, and alternatively by an XML Schema, as specified by the W3C [XMLSchemas]. In an attempt to maintain continuity as MathML is revised a new MathML3 XML Schema is provided in Appendix A, but the developments since MathML2 have meant that the normative schema for MathML3 is provided in Relax_NG form [RELAX-NG].

A special aspect of the MathML specification is that there are two main strains of markup, Chapter 3 and Chapter 4, which address, necessarily separately, the presentational and semantic aspects of formulas. Content markup, as the semantically oriented part is called, is specified in particular special detail. This specification makes use of a different type of format called Content Dictionaries, which is also an XML application. This has been developed in collaboration with the OpenMath movement and is given in Chapter 8. Of course, one drawback to using MathML specific rules could be that they are invisible to generic XML processors and validators. The use of an new XML format for Content Dictionaries addresses this problem.

Editor's note: MiKo What was there is not true any more (and contained offending references).

Editor's note:P. IonThere has to be a mention of xml-base and its inheritance and defaults; probably cdbase has to be pointed out too, although discussed later under content math (or CDs).

There are basically two kinds of additional MathML grammar and syntax rules. One kind involves placing additional criteria on attribute values. For example, it is not possible in pure XML to require that an attribute value be a positive integer. The second kind of rule specifies more detailed restrictions on the child elements (for example on ordering) than are given in the DTD or even a schema. For example, it is not possible in XML to specify that the first child be interpreted one way, and the second in another.

The following sections discuss features both of XML syntax and grammar in general, and of MathML in particular. Throughout the remainder of the MathML specification, we will usually take care to distinguish between usage required by XML syntax and the MathML Schema and usage required by MathML specific rules. However, we will often allude to 'MathML errors' without identifying which part of the specification is being violated.

2.1.2 Children versus Arguments

Many MathML elements require a specific number of children or attach additional meanings to child elements in certain positions. As noted above, these kinds of requirements are specific to MathML, and cannot be given entirely using XML syntax and grammar. When the children of a given MathML element are subject to these kinds of additional conditions, we will often refer to them as *arguments* instead of merely as children, in order to emphasize their MathML specific usage. Note that, especially in Chapter 3, the term 'argument' is usually used in this technical sense, unless otherwise noted, and therefore refers to a child element.

In the detailed discussions of element syntax given with each element throughout the MathML specification, the number of required arguments and their order is implicitly indicated by giving names for the arguments at various positions. This information is also given for presentation elements in the table of argument requirements in Section 3.1.3.

A few elements have other requirements on the number or type of arguments. These additional requirements are described together with the individual elements.

2.1.3 MathML Attribute Values

An XML attribute's value, which in general in MathML can be a string of arbitrary characters, must be surrounded by a pair of either double quotes (`"`) or single quotes (`'`). The kind of quotation marks not used to surround the value may be included within it. Attribute names are generally shown in a `monospaced` font within descriptive text in this specification, just as the `monospaced` font is used for examples.

MathML uses a more complicated syntax for attribute values than the generic XML syntax. These additional rules are intended for use by MathML applications, and it is a MathML error to violate them, though they cannot be enforced by XML processing. The MathML syntax of each attribute value is specified in the table of attributes provided with the description of each element, using a notation described below. When MathML applications process attribute values, whitespace is ignored except to separate letter and digit sequences into individual words or numbers. Attribute values may contain any MathML characters as specified in Chapter 6 permitted by the syntax restrictions for an attribute. Character data can be included directly in attribute values, or by using entity references as described in Section 6.2. However, modern practice suggest that it is preferable to use numeric character references to XML entities to avoid the need for the presence of a DTD with the entity definitions. After the initial parsing the character entities are all resolved to Unicode character codes in any case.

In particular, the characters `"`, `'`, `&` and `<` can be included in MathML attribute values (when permitted by the attribute value syntax) using the entity references `"`, `'`, `&` and `<`, respectively. These characters have special roles in XML and for that reason are usable in character entity form without resorting to Unicode character codes, which are, of course, valid too.

2.1.3.1 Syntax notations used in the MathML specification

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

Notation	What it matches
number	a decimal integer or rational number (a string of digits with one decimal point), optionally starting with '-'
unsigned-number	a decimal integer or real number, no sign
integer	a decimal integer, optionally starting with '-'
positive-integer	a decimal integer, unsigned, not 0
string	an arbitrary character string (always the entire attribute value)
character	a single non-whitespace character, or MathML entity reference; whitespace separation is optional
#rrggbb	RGB color value; the three pairs of hexadecimal digits in the example #5599dd define proportions of red, green and blue on a scale of x00 through xFF, which gives a strong sky blue.
h-unit	a unit of horizontal length (allowable units are listed below)
v-unit	a unit of vertical length (allowable units are listed below)
css-fontfamily	explained in the CSS subsection below
css-color-name	explained in the CSS subsection below
other italicized words	explained in the text for each attribute
form +	one or more instances of 'form'
form *	zero or more instances of 'form'
f1 f2 ... fn	one instance of each form, in sequence, perhaps separated by whitespace
f1 | f2 | ... | fn	any one of the specified forms
[form]	an optional instance of 'form'
(form)	same as form
word in plain text	that same word, literally present in the attribute value (unless it is obviously part of an explanatory phrase)
quoted symbol	that same symbol, literally present in the attribute value (e.g. "+" or '+')

The order of precedence of the syntax notation operators is, from highest to lowest precedence:

- form + or form *
- f1 f2 ... fn (sequence of forms)
- f1 | f2 | ... | fn (alternative forms)

A *string* can contain arbitrary characters which are specifiable within XML `CDATA` attribute values. See Chapter 6 for a full discussion of MathML characters. No syntax rule in MathML includes a *string* as only part of an attribute value; a string can only be the entire value.

Adjacent keywords and numbers must be separated by whitespace in the actual attribute values, except for unit identifiers (denoted by `h-unit` or `v-unit` syntax symbols) following numbers. Whitespace is not otherwise required, but is permitted between any of the tokens listed above, except (for compatibility with CSS) immediately before unit identifiers, between the '-' signs and digits of negative numbers, or between `#` and `"rrggbb"` or `"rgb"`.

Numerical attribute values for dimensions that should depend upon the current font can be given in font-related units, or in named absolute units (described in a separate subsection below). Horizontal dimensions are conventionally given in `em`'s, and vertical dimensions in `ex`'s, by immediately following a number by one of the unit identifiers `"em"` or `"ex"`. For example, the horizontal spacing around an operator such as '+' is conventionally given in `"em"`s, though other units can be used. Using font-related units is usually preferable to using absolute units, since it allows renderings to grow or shrink in proportion to the current font size.

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).

The symbols h-unit, v-unit, css-fontfamily, and css-color-name are explained in the following subsections.

2.1.3.2 Attributes with units

Some attributes accept horizontal or vertical lengths as numbers followed by a 'unit identifier' (often just called a 'unit'). The syntax symbols h-unit and v-unit refer to a unit for horizontal or vertical length, respectively. The possible units and the lengths they refer to are shown in the table below; they are the same for horizontal and vertical lengths, but the syntax symbols are distinguished in attribute syntaxes as a reminder of the direction each is used in.

The unit identifiers and meanings are taken from CSS. However, the syntax of numbers followed by unit identifiers in MathML is not identical to the syntax of length values with units in CSS style sheets, since numbers in CSS cannot end with decimal points, and are allowed to start with '+' signs.

The possible horizontal or vertical units in MathML are:

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

The typesetting units "em" and "ex" are defined in Appendix D, and discussed further under 'Additional notes' below.

% is a 'relative unit'; when an attribute value is given as "n%" (for any numerical value "n"), the value being specified is the default value for the property being controlled multiplied by "n" divided by 100. The default value (or the way in which it is obtained, when it is not constant) is listed in the table of attributes for each element, and its meaning is described in the subsequent documentation about that attribute. (The mpadded element has its own syntax for % and does not allow it as a unit identifier.)

For consistency with lengths in CSS, length units in MathML are rarely optional. When they are, the unit symbol is enclosed in square brackets in the attribute syntax, following the number to which it applies, e.g. number [h-unit]. The meaning of specifying no unit is given in the description for each attribute; in general it is that the number given is a multiplier for the default value of the attribute. (In such cases, specifying the number "nnn" without a unit is equivalent to specifying the number "nnn" times 100 followed by %. For example, <mo maxsize="2"> (</mo> is equivalent to <mo maxsize="200%"> (</mo>.)

As a special exception (also consistent with CSS), a numerical value equal to 0 need not be followed by a unit identifier even if the syntax specified here requires one. In such cases, the unit identifier (or lack of one) would not matter, since 0 times any unit is 0.

For most attributes, the typical unit which would be used to describe them in typesetting is chosen as the one used in that attribute's default value in this specification; when a specific default value is not given, the typical unit is usually mentioned in the syntax table or in the documentation for that attribute. The most common units are `em` or `ex`. However, any unit can be used, unless otherwise specified for a specific attribute.

Additional notes about units

Note that some attributes, e.g. `framespacing` on a `<mtable>`, can contain more than one numerical value, each followed by its own unit.

It is conventional to use the font-relative unit `ex` mainly for vertical lengths, and `em` mainly for horizontal lengths, but this is not required. These units are relative to the font and font size which would be used for rendering the element in whose attribute value they are specified, which means they should be interpreted *after* attributes such as `fontfamily` and `fontsize` are processed, if those occur on the same element, since changing the current font or font size can change the length of one of these units.

The definition of the length of each unit, but not the MathML syntax for length values, is as specified in CSS, except that if a font provides specific values for `em` and `ex` which differ from the values defined by CSS (the font size and 'x'-height respectively), those values should be used.

2.1.3.3 CSS-compatible attributes

Several MathML attributes, listed below, correspond closely to text rendering properties defined originally in [CSS1]. In MathML 1.01, the names and values of these attributes were aligned with the CSS Recommendation where possible. This was done so that renderers in CSS environments could query the environment for the corresponding property when determining the default values for the attributes.

Allowing style properties to be set both via MathML attributes and CSS style sheets has drawbacks. At a minimum, duplication is confusing, and at worst, it leads to the meaning of equations being inadvertently changed by document-wide CSS changes. For these reasons, these attributes have been deprecated. In their place, MathML 2.0 introduced four new mathematical style attributes. These attributes use logical values to better capture the abstract categories of letter-like symbols used in math, and afford a much cleaner separation between MathML and CSS. See Section 3.2.2 for more details.

For reference, a table showing the correspondence of the deprecated MathML 1.01 style attributes with their CSS counterparts is given below:

MathML attribute	CSS property	syntax symbol	MathML elements	refer to
fontsize	font-size	-	presentation tokens; `mstyle`	Section 3.2.2
fontweight	font-weight	-	presentation tokens; `mstyle`	Section 3.2.2
fontstyle	font-style	-	presentation tokens; `mstyle`	Section 3.2.2
fontfamily	font-family	css-fontfamily	presentation tokens; `mstyle`	Section 3.2.2
color	color	css-color-name	presentation tokens; `mstyle`	Section 3.3.4
background	background	css-color-name	`mstyle`	Section 3.3.4

See also Section 2.1.4 below for a discussion of the `class`, `style` and `xml:id` attributes for use with style sheets.

Order of processing attributes versus style sheets

CSS or analogous style sheets can specify changes to rendering properties of selected MathML elements. Since rendering properties can also be changed by attributes on an element, or be changed automatically by the renderer, it is necessary to specify the order in which changes requested by various sources should occur. An example of automatic adjustment is what happens for `fontsize`, as explained in the discussion on `scriptlevel` in Section 3.3.4. In the case of 'absolute' changes, i.e., setting a new property value independent of the old value (as opposed to 'relative' changes, such as increments or multiplications by a factor), the absolute change performed last will be the only absolute change which is effective, so the sources of changes which should have the highest priority must be processed last.

In the case of CSS, the order of processing of changes from various sources which affect one MathML element's rendering properties should be as follows:

(first changes; lowest priority)

- Automatic changes to properties or attributes based on the type of the parent element, and this element's position in the parent, as for the changes to `fontsize` in relation to `scriptlevel` mentioned above; such changes will usually be implemented by the parent element itself before it passes a set of rendering properties to this element
- From a style sheet from the reader: styles which are *not* declared 'important'
- Explicit attribute settings on this MathML element
- From a style sheet from the author: styles which are *not* declared 'important'
- From a style sheet from the author: styles which *are* declared 'important'
- From a style sheet from the reader: styles which *are* declared 'important'

(last changes; highest priority)

Note that the order of the changes derived from CSS style sheets is specified by CSS itself (this is the order specified by CSS2). The following rationale is related only to the issue of where in this pre-existing order the changes caused by explicit MathML attribute settings should be inserted.

Rationale: MathML rendering attributes are analogous to HTML rendering attributes such as `align`, which the CSS section on cascading order specifies should be processed with the same priority. Furthermore, this choice of priority permits readers, by declaring certain CSS styles as 'important', to decide which of their style preferences should override explicit attribute settings in MathML. Since MathML expressions, whether composed of 'presentation' or 'content' elements, are primarily intended to convey meaning, with their 'graphic design' (if any) intended mainly to aid in that purpose but not to be essential in it, it is likely that readers will often want their own style preferences to have priority; the main exception will be when a rendering attribute is intended to alter the meaning conveyed by an expression, which is generally discouraged in the presentation attributes of MathML.

2.1.3.4 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 (unless they are obviously explanatory phrases), 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 under `mstyle`, or in some cases (described individually) 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 rendering environment.

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

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

Note that, in general, there is no value which can be given explicitly for a MathML attribute which will simulate the effect of not specifying the attribute at all for attributes which are *inherited* or *automatic*. Giving the words 'inherited' or 'automatic' explicitly will not work, and is not generally allowed. Furthermore, even for presentation attributes for which a specific default value is documented here, the `mstyle` element (Section 3.3.4) can be used to change this for the elements it contains. Therefore, the MathML DTD declares most presentation attribute default values as `#IMPLIED`, which prevents XML preprocessors from adding them with any specific default value. This point of view is carried through to the MathML schema.

2.1.3.5 Attribute values in the MathML DTD

In an XML DTD, allowed attribute values can be declared as general strings, or they can be constrained in various ways, either by enumerating the possible values, or by declaring them to be certain special data types. The choice of an XML attribute type affects the extent to which validity checks can be performed using a DTD.

The MathML DTD specifies formal XML attribute types for all MathML attributes, including enumerations of legitimate values in some cases. In general, however, the MathML DTD is relatively permissive, frequently declaring attribute values as strings; this is done to provide for interoperability with SGML parsers while allowing multiple attributes on one MathML element to accept the same values (such as `"true"` and `"false"`), and also to allow extension to the lists of predefined values.

At the same time, even though an attribute value may be declared as a string in the DTD, only certain values are legitimate in MathML, as described above and in the rest of this specification. For example, many attributes expect numerical values. In the sections which follow, the allowed attribute values are described for each element. To determine when these constraints are actually enforced in the MathML DTD, consult Appendix A. However, lack of enforcement of a requirement in the DTD does *not* imply that the requirement is not part of the MathML language itself, or that it will not be enforced by a particular MathML renderer. (See Section 2.3.2 for a description of how MathML renderers should respond to MathML errors.)

Furthermore, the MathML DTD is provided for convenience; although it is intended to be fully compatible with the text of the specification, the text should be taken as definitive if there is a contradiction. (Any contradictions which may exist between various chapters of the text should be resolved by favoring Chapter 6 first, then Chapter 3, Chapter 4, then Section 2.1, and then other parts of the text.) For the MathML schema the situation will be the same: the published Recommendation text takes precedence. Though this is what is intended to happen, there is a practical difficulty. If the system processing the MathML uses a validating parser, whether it be based on a DTD or on a schema, the process will probably simply stop when it hits something held to be incorrect syntax, whether or not further MathML processing in full harmony with the specification would have processed the piece correctly.

2.1.4 Attributes Shared by all MathML Elements

In order to facilitate use with style sheet mechanisms such as [XSLT] and [CSS2] all MathML elements accept `class`, `style`, and `xml:id` attributes in addition to the attributes described specifically for each element. MathML renderers not supporting CSS may ignore these attributes. MathML specifies these attribute values as general strings, even if style sheet mechanisms have more restrictive syntaxes for them. That is, any value for them is valid in MathML.

In order to facilitate compatibility with linking mechanisms, all MathML elements accept the `xlink:href` attribute.

All MathML elements also accept the `xref` attribute for use in parallel markup (Section 5.5). The `xml:id` is also used in this context.

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

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

2.1.5 Collapsing Whitespace in Input

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

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.

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

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

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

Authors wishing to encode whitespace characters at the start or end of the content of a token, or in sequences other than a single space, without having them ignored, must use ` ` or other 'whitespace' non-marking entities as described in Section 6.6. For example, compare

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

with

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

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

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

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

2.2 Interfacing

Issue (): The current section needs continuing and updating further in later drafts.

To be effective, MathML must work well with a wide variety of renderers, processors, translators and editors. This section raises some of the interface issues involved in generating and rendering MathML. Since MathML exists primarily to encode mathematics in Web documents, perhaps the most important interface issues are related to embedding MathML in [HTML4] and [XHTML], and in a newer HTML5 when it appears.

There are three kinds of interface issues that arise in embedding MathML in other XML documents. First, MathML must be semantically integrated. MathML markup must be recognized as valid embedded XML content, and not as an error. This could be seen as primarily a question of managing namespaces in XML [Namespaces]. However, the implementation of XML namespaces and their management has not been well supported by recent commercial software. So there have grown up other ways of dealing with 'foreign content' in an XML document which is viewed as of a particular type. The Compound Document Formats Working Group (CDF WG) of the W3C has been grappling with the questions of putting together XML vocabularies and has been defining ways to do so for particular combinations of vocabularies. Their initial success has been with specifying profiles for combining XHTML and SVG, with special attention paid to the needs of mobile phone technology. The W3C Math WG is working with the CDF WG to define profiles for full scientific documents involving XHTML for text, MathML for equations and SVG for diagrams and images.

Second, in the case of HTML/XHTML, MathML rendering must be integrated with browser software. Some browsers already implement MathML rendering natively, and one can expect more browsers will do so in the future. At the same time, other browsers have developed infrastructure to facilitate the rendering of MathML and other embedded XML content by third-party software or other built-in technology. Examples of this built-in technology are the sophisticated CSS rendering engines now available, and the powerful implementations of ECMAscript (or JavaScript) that are becoming common. Using these browser-specific mechanisms generally requires additional interface markup of some sort to activate them. In the case of CSS, there is a special restricted form of MathML3 tailored for use with present-day CSS, up to CSS2.1, which is specified in "A MathML for CSS profile" [MathMLforCSS]. This does not offer the full expressiveness afforded by MathML3 but provides a portable simpler form that can be rendered acceptably on the screen by modern CSS engines.

Third, other tools for generating and processing MathML must be able to intercommunicate. A number of MathML tools have been or are being developed, including editors, translators, computer algebra systems, and other scientific software. However, since MathML expressions tend to be lengthy, and prone to error when entered by hand, special emphasis must be given to ensuring that MathML can be easily generated by user-friendly conversion and authoring tools, and that these tools work together in a dependable, platform and vendor independent way.

2.3 Conformance

Information is nowadays commonly generated, processed and rendered by software tools. The exponential growth of the Web is fueling the development of advanced systems for automatically searching, categorizing, and interconnecting information. Thus, although MathML can be written by hand and read by humans, 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 a equation editor.

In this section, guidelines are given for describing different types of MathML support, and for making clear the extent of MathML support in a given application. Developers, users and reviewers are encouraged to use these guidelines in characterizing products. The intention behind these guidelines is to facilitate reuse and interoperability between MathML applications by accurately setting out their capabilities in quantifiable terms.

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

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

2.3.1 MathML Conformance

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

Editor's note:P. IonThe Relax_NG Schema seems to be dominant now, not the DTD.

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

Three forms of MathML conformance are specified:

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

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

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

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

2.3.1.1 MathML Test Suite and Validator

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

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

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

For MathML-output-conformant processors, there is also a MathML validator accessible over the Web. 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 Features

MathML 2.0 contains a number of MathML 1.x features 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 MathML 2.0 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/reading tools must support deprecated features if they are to be in conformance with MathML 1.x. They do not have to support deprecated features to be considered in conformance with MathML 2.0. However, all tools are encouraged to support the old forms as much as possible.
3. In order to be MathML-roundtrip-conformant, a processor need only preserve MathML equivalence on expressions containing no deprecated features.

2.3.1.3 MathML 2.0 Extension Mechanisms and Conformance

MathML 2.0 defines three 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 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 2.0, which allows MathML to be used for exploring new ideas not yet susceptible to standardization. However, as new ideas take hold, they may become part of future standards. For example, an emerging character that must be represented by an `mglyph` element today may be assigned a Unicode codepoint in the future. At that time, representing the character directly by its Unicode codepoint would be preferable.

Because the possibility of future obsolescence is inherent in the use of extension mechanisms to facilitate the discussion of new ideas, MathML 2.0 makes no conformance requirement concerning the use of extension mechanisms, even when alternative standard markup is available. For example, using an `mglyph` element to represent an 'x' is permitted. However, authors and implementors are strongly encouraged to use standard markup whenever possible. Similarly, maintainers of documents employing MathML 2.0 extension mechanisms are encouraged to monitor relevant standards activity (e.g. Unicode, OpenMath, etc) and update documents as more standardized markup becomes available.

2.3.2 Handling of Errors

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

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

2.3.3 Attributes for unspecified data

The MathML attributes described in the MathML specification are necessary for presentation and content markup. Ideally, the MathML attributes should be an open-ended list so that users can add specific attributes for specific renderers. However, this cannot be done within the confines of a single XML DTD. Although it can be done using extensions of the standard DTD, 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 meantime, however, the development of a general XML namespace mechanism has made 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.

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 3.6.1), that information should be passed in using the other attribute:

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

In MathML 2.0, 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.

2.4 Future Extensions

If MathML is to remain useful in the future, it is to be expected that MathML will need to be extended and revised in various ways. Some of these extensions can be easily foreseen; for example, as work on behavioral extensions to CSS proceeds, MathML will likely need to be extended as well.

Similarly, there are several kinds of functionality that are fairly obvious candidates for future MathML extensions. These include macros, style sheets, and perhaps a general facility for 'labeled diagrams'. However, there will no doubt be other desirable extensions to MathML that will only emerge as MathML is widely used. For these extensions, the W3C Math Working Group relies on the extensible architecture of XML, and the common sense of the larger Web community.

2.4.1 Macros and Style Sheets

The development of style-sheet mechanisms for XML is part of the ongoing XML activity of the World Wide Web Consortium. Both XSL and CSS are working to incorporate greater support for mathematics.

In particular, XSL Transformations [XSLT] are likely to have a large impact on the future development of MathML. Macros have traditionally contributed greatly the usability and effectiveness of mathematics encodings. Further work developing applications of XSLT tailored specifically to MathML is clearly called for.

Some of the possible uses of macro capabilities for MathML include:

Abbreviation One common use of macros is for abbreviation. Authors needing to repeat some complicated but constant notation can define a macro. This greatly facilitates hand authoring. Macros that allow for substitution of parameters facilitate such usage even further.

Extension of Content Markup By defining macros for semantic objects, for example a binomial coefficient, or a Bessel function, one can in effect extend the content markup for MathML. Such a macro could include an explicit semantic binding, or such a binding could be easily added by an external application. Narrowly defined disciplines should be able to easily introduce standardized content markup by using standard macro packages. For example, the OpenMath project could release macro packages for attaching OpenMath content markup.

Rendering and Style Control Another basic way in which macros are often used is to provide a way of controlling style and rendering behavior by replacing high-level macro definitions. This is especially important for controlling the rendering behavior of MathML content tags in a context sensitive way. Such a macro capability is also necessary to provide a way of attaching renderings to user-defined XML extensions to the MathML core.

Accessibility Reader-controlled style sheets are important in providing accessibility to MathML. For example, a reader listening to a voice renderer might, by default, hear a bit of MathML presentation markup read as 'D sub x sup 2 of f'. Knowing the context to be multi-variable calculus, the reader may wish to use a style sheet or macro package that instructs the renderer to render this `<msubsup>` element as 'second derivative with respect to x of f'.

2.4.2 XML Extensions to MathML

The set of elements and attributes specified in the MathML specification are necessary for rendering common mathematical expressions. It is recognized that not all mathematical notation is covered by this set of elements, that new notations are continually invented, and that sub-communities within mathematics often have specialized notations; and furthermore that the explicit extension of a standard is a necessarily slow and conservative process. This implies that the MathML standard could never explicitly cover all the presentational forms used by every sub-community of authors and readers of mathematics, much less encode all mathematical content.

In order to facilitate the use of MathML by the widest possible audience, and to enable its smooth evolution to encompass more notational forms and more mathematical content (perhaps eventually covered by explicit extensions to the standard), the set of tags and attributes is open-ended, in the sense described in this section.

MathML is described by an XML DTD, which necessarily limits the elements and attributes to those occurring in the DTD. Renderers desiring to accept non-standard elements or attributes, and authors desiring to include these in documents, should accept or produce documents that conform to an appropriately extended XML DTD that has the standard MathML DTD as a subset.

MathML renderers are allowed, but not required, to accept non-standard elements and attributes, and to render them in any way. If a renderer does not accept some or all non-standard tags, it is encouraged either to handle them as errors as described above for elements with the wrong number of arguments, or to render their arguments as if they were arguments to an `mrow`, in either case rendering all standard parts of the input in the normal way.

2.5 Embedding MathML in other Documents

While MathML can be used in isolation as a language for exchanging mathematical expressions between MathML-aware applications, the primary anticipated use of MathML is to encode mathematical expression within larger documents. MathML is ideal for embedding math expressions in other applications of XML.

In particular, the focus here is on the mechanics of embedding MathML in [XHTML]. XHTML is a W3C Recommendation formulating a family of current and future XML-based document types and modules that reproduce, subset, and extend HTML. While [HTML4] is the dominant language of the Web at the time of this writing, one may anticipate a shift from HTML to XHTML. Indeed, XHTML can already be made to render properly in most HTML user agents.

Since MathML and XHTML share a common XML framework, namespaces provide a standard mechanism for embedding MathML in XHTML. While some popular user agents also support inclusion of MathML directly in HTML as "XML data islands," this is a transitional strategy. Consult user agent documentation for specific information on its support for embedding XML in HTML.

2.5.1 MathML and Namespaces

Embedding MathML in XML-based documents in general, and XHTML in particular, is a matter of managing namespaces. See the W3C Recommendation "Namespaces in XML" [Namespaces] for full details.

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

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

Using namespaces, embedding a MathML expression in a larger XML document is merely a matter of identifying the MathML markup as residing in the MathML namespace. This can be accomplished by either explicitly identifying each MathML element name by attaching a namespace prefix, or by declaring a default namespace on an enclosing element.

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

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.

Example:

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

These two methods of namespace declaration can be used together. For example, by using both an explicit document-wide namespace prefix, and default namespace declarations on individual mathematical elements, it is possible to localize namespace related markup to the top-level `math` element.

Example:

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

2.5.1.1 Document Validation Issues

The use of namespace prefixes creates an issue for DTD validation of documents embedding MathML. DTD validation requires knowing the literal (possibly prefixed) element names used in the document. However, the Namespaces in XML Recommendation [Namespaces] allows the prefix to be changed at arbitrary points in the document, since namespace prefixes may be declared on any element.

The 'historical' method of bridging this gap was to write a DTD with a fixed prefix, or in the case of XHTML and MathML, with no prefix, and mandate that the specified form must be used throughout the document. However, this is somewhat restricting for a modular DTD that is intended for use in conjunction with another DTD, which is exactly the situation with MathML in XHTML. In essence, the MathML DTD would have to allocate a prefix for itself and hope no other module uses the same prefix to avoid name clashes, thus losing one of the main benefits of XML namespaces.

One strategy for addressing this problem is to make every element name in the DTD be accessed by an entity reference. This means that by declaring a couple of entities to specify the prefix before the DTD is loaded, the prefix can be chosen by a document author, and compound DTDs that include several modules can, without changing the module DTDs, specify unique prefixes for each module to avoid clashes. The MathML DTD has been designed in this fashion. See Section A.3 and [Modularization] for details.

An extra issue arises in the case where explicit prefixes are used on the top-level `math` element, but a default namespace is used for other MathML elements. In this case, one wants the MathML module to be included into XHTML with the prefix set to empty. However, the 'driver' DTD file that sets up the inclusion of the MathML module would then need to define a new element called m:math. This would allow the top-level `math` element to use an explicit prefix, for attaching rendering behaviors in current browsers, while the contents would not need an explicit prefix, for ease of interoperability between authoring tools, etc.

2.5.1.2 Compatibility Suggestions

While the use of namespaces to embed MathML in other XML applications is completely described by the relevant W3C Recommendations, a certain degree of pragmatism is still called for at present. Support for XML, namespaces and rendering behaviors in popular user agents is not always fully in alignment with W3C Recommendations. In some cases, the software predates the relevant standards, and in other cases, the relevant standards are not yet complete.

During the transitional period, in which some software may not be fully namespace-aware, a few conventional practices will ease compatibility problems:

1. When using namespace prefixes with MathML markup, use m: as a conventional prefix for the MathML namespace. Using an explicit prefix is probably safer for compatibility in current user agents.
2. When using namespace prefixes, pick one and use it consistently within a document.
3. Explicitly declare the MathML namespace on all `math` elements.

Examples.

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

Or

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

Note that these suggestions alone may not be sufficient for creating functional Web pages containing MathML markup. It will generally be the case that some additional document-wide markup will be required. Additional work may also be required to make all MathML instances in a document compatible with document-wide declarations. This is particularly true when documents are created by cutting and pasting MathML expressions, since current tools will probably not be able to query global namespace information.

Consult the W3C Math Working Group home page for compatibility and implementation suggestions for current browsers and other MathML-aware tools.

2.5.2 The Top-Level `math` Element

MathML specifies a single top-level or root `math` element, which encapsulates each instance of MathML markup within a document. All other MathML content must be contained in a `math` element; equivalently, every valid, complete MathML expression must be contained in `<math>` tags. The `math` element must always be the outermost element in a MathML expression; it is an error for one `math` element to contain another.

Applications that return sub-expressions of other MathML expressions, for example, as the result of a cut-and-paste operation, should always wrap them in `<math>` tags. Ideally, the presence of enclosing `<math>` tags should be a very good heuristic test for MathML material. Similarly, applications which insert MathML expressions in other MathML expressions must take care to remove the `<math>` tags from the inner expressions.

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

The attributes of the `math` element are:

class, id, style Provided for use with stylesheets.

xref Provided along with `xml:id` for use in parallel markup (Section 5.5)

macros This attribute provides a way of pointing to external macro definition files. Macros are not part of the MathML specification, and much of the functionality provided by macros in MathML can be accommodated by XSL transformations [XSLT]. However, the `macros` attribute is provided to make possible future development of more streamlined, MathML-specific macro mechanisms. The value of this attribute is a sequence of URLs or URIs, separated by whitespace

mode The `mode` attribute specifies whether the enclosed MathML expression should be rendered in a display style or an in-line style. Allowed values are `"display"` and `"inline"` (default). This attribute is deprecated in favor of the new `display` attribute, or the CSS2 'display' property with the analogous `block` and `inline` values.

display The `display` attribute replaces the deprecated `mode` attribute. It specifies whether the enclosed MathML expression should be rendered in a display style or an in-line style. Allowed values are `"block"` and `"inline"` (default).

dir The `dir` attribute specifies the overall directionality of layout. Allowed values are `"ltr"`(default) or `"rtl"`. This attribute, in addition to the directionality of the text content of token elements, is used for presentation of mathematics in Right-to-Left scripts. See Section 3.1.5 for further discussion.

The attributes of the `math` element affect the entire enclosed expression. They are, in a sense, 'inward looking'. However, to render MathML properly in a browser, and to integrate it properly into an XHTML document, a second collection of 'outward looking' attributes are also useful.

While general mechanisms for attaching rendering behaviors to elements in XML documents are under development, wide variations in strategy and level of implementation remain between various existing user agents. Consequently, the remainder of this section describes attributes and functionality that are desirable for integrating third-party rendering modules with user agents:

overflow 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), this attribute is provided to suggest a processing method to the renderer. Allowed values are:

- **linebreak** (Default) The expression will be broken across several lines. The line breaking algorithm is not specified, but it is recommended that line breaking should try to keep meaningful subexpressions together and indent lines in a manner that aids in understanding the expression.
- **scroll** The window provides a viewport into the larger complete display of the mathematical expression. Horizontal or vertical scrollbars are added to the window as necessary to allow the viewport to be moved to a different position.
- **elide** The display is abbreviated by removing enough of it so that the remainder fits into the window. For example, a large polynomial might have the first and last terms displayed with '+ ... +' between them. Advanced renderers may provide a facility to zoom in on elided areas.
- **truncate** The display is abbreviated by simply truncating it at the right and bottom borders. It is recommended that some indication of truncation is made to the viewer.
- **scale** The fonts used to display the mathematical expression are chosen so that the full expression fits in the window. Note that this only happens if the expression is too large. In the case of a window larger than necessary, the expression is shown at its normal size within the larger window.

altimg This attribute provides a graceful fall-back for browsers that do not support embedded elements. The value of the attribute is an URL.

alttext This attribute provides a graceful fall-back for browsers that do not support embedded elements or images. The value of the attribute is a text string.

altimg-width This attribute provides a width for the `altimg` (if any). The value of attribute is an `h-unit`. This value is useful for high resolution images which, if displayed at their full resolution, would be too large.

If neither `altimg-width` nor `altimg-height` is given, then for those renderers that use an image, they should use the image's natural size. If only the width is given, the renderer should scale the height so as to preserve the aspect ration of the image.

altimg-height This attribute provides a total height for the `altimg` (if any). The value of attribute is a `v-unit`. This value is useful for high resolution images which, if displayed at their full resolution, would be too large. If neither `altimg-width` nor `altimg-height` is given, then for those renderers that use an image, they should use the image's natural size. If only the width is given, the renderer should scale the width so as to preserve the aspect ration of the image.

altimg-valign By default, the bottom of the image aligns to the current baseline. The `valign` attribute specifies the alignment point within the image. The value of attribute is a `v-unit`. A positive value of `valign` shifts the bottom of the image below the current baseline, while a negative value will raise it above the baseline.

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

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

Chapter 3

Presentation Markup

3.1 Introduction

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

3.1.1 What Presentation Elements Represent

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

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

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

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

Similarly, superscripts are attached not just to the preceding character, but to the full expression constituting their base. This structure allows for better-quality rendering of mathematics, especially when details of the rendering environment such as display widths are not known to the document author; it also greatly eases automatic interpretation of the mathematical structures being represented.

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

The complete list of MathML entities is described in Chapter 6.

3.1.2 Terminology Used In This Chapter

It is strongly recommended that, before reading the present chapter, one read Section 2.1 on MathML syntax and grammar, which contains important information on MathML notations and conventions. In particular, in this chapter it is assumed that the reader has an understanding of basic XML terminology described in Section 2.1.2, and the attribute value notations and conventions described in Section 2.1.3.

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

3.1.2.1 Types of presentation elements

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

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

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

3.1.2.2 Terminology for other classes of elements and their relationships

The terminology used in this chapter for special classes of elements, and for relationships between elements, is as follows: The *presentation elements* are the MathML elements defined in this chapter. These elements are listed in Section 3.1.6. The *content elements* are the MathML elements defined in Chapter 4.

A MathML *expression* is a single instance of any of the presentation elements with the exception of the empty elements `none` or `mprescripts`, or is a single instance of any of the content elements which are allowed as content of presentation elements (described in Section 5.4.3). A *sub-expression* of an expression *E* is any MathML expression that is part of the content of *E*, whether *directly* or *indirectly*, i.e. whether it is a 'child' of *E* or not.

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

3.1.3 Required Arguments

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

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

3.1.3.1 Inferred `mrow`s

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

For example,

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

is treated as if it were

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

and

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

is treated as if it were

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

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

In the descriptions in this chapter of the above-listed elements' rendering behaviors, their content can be assumed to consist of exactly one expression, which may be an `mrow` element formed from their arguments in this manner. However, their argument counts are shown in the following table as 1*, since they are most naturally understood as acting on a single expression.

3.1.3.2 Table of argument requirements

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

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

3.1.4 Elements with Special Behaviors

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

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

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

Certain elements treat their arguments as the arguments of an ‘inferred `mrow`’ if they are not given exactly one argument, as explained in Section 3.1.3.

In MathML 1.x, the `mtable` element could infer `mtr` elements around its arguments, and the `mtr` element could infer `mtd` elements. In MathML 2.0, `mtr` and `mtd` elements must be explicit. However, for backward compatibility renderers may wish to continue supporting inferred `mtr` and `mtd` elements.

3.1.5 Directionality

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

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

3.1.5.1 Overall Directionality of Mathematics Formulas

The overall directionality for a formula, basically the direction of the Layout Schemata, is specified by the `dir` attribute on the containing `math` element (see Section 2.5.2). The default is `ltr`. When `dir='rtl'` is used, the layout is simply the mirror image of the conventional European layout. That is, shifts up or down are unchanged, but the progression in laying out is from right to left. Sub- and superscripts appear to the left of the base; the surd for a root appears at the right, with the bar continuing over the base to the left.

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

3.1.5.2 Bidirectional Layout in Token Elements

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

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

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

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

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

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

3.1.6 Summary of Presentation Elements

3.1.6.1 Token Elements

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

3.1.6.2 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.

3.1.6.3 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

3.1.6.4 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

3.1.6.5 Enlivening Expressions

`maction`	bind actions to a sub-expression

3.2 Token Elements

Token elements in presentation markup are broadly intended to represent the smallest units of mathematical notation which carry meaning. Tokens are roughly analogous to words in text. However, because of the precise, symbolic nature of mathematical notation, the various categories and properties of token elements figure prominently in MathML markup. By contrast, in textual data, individual words rarely need to be marked up or styled specially.

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

3.2.1 MathML characters in token elements

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

While all Unicode character data is valid in token element content, MathML 2.0 distinguishes a special subset of named Unicode 3.2 characters, called MathML characters in this document. The complete list of MathML characters is defined in Chapter 6. MathML characters can be either represented directly as Unicode character data, or indirectly via numeric or character entity references. See Chapter 6 for a discussion of the advantages and disadvantages of numeric character references versus entity references. New mathematics characters that arise, or non-standard glyphs for existing MathML characters, may be represented by means of the `mglyph` element.

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

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

3.2.1.1 Alphanumeric symbol characters

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

For these reasons, Unicode 3.2 contains more than nine hundred Math Alphanumeric Symbol characters corresponding to letter-like symbols. These characters are in the Secondary Multilingual Plane (SMP). See Chapter 6 for more information. As valid Unicode data, these characters are permitted in MathML 2.0, and as tools and fonts for them become widely available, we anticipate they will be the predominant way of denoting letter-like symbols.

MathML 2.0 also provides an alternative encoding for these characters using only Basic Multilingual Plane (BMP) characters together with markup. MathML 2.0 defines a correspondence between token elements with certain combinations of BMP character data and the `mathvariant` attribute and tokens containing SMP Math Alphanumeric

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

The next section discusses the `mathvariant` attribute in more detail, and a complete technical description of the corresponding characters is given in Section 6.5.

3.2.2 Mathematics style attributes common to token elements

MathML 2.0 introduces four new *mathematics style* attributes. These attributes are valid on all presentation token elements , and on no other elements except `mstyle`. The attributes are:

Name	values	default
mathvariant	normal | bold | italic | bold-italic | double-struck | bold-fraktur | script | bold-script | fraktur | sans-serif | bold-sans-serif | sans-serif-italic | sans-serif-bold-italic | monospace | initial | tailed | looped | stretched	normal (*except on* `<mi>`)
mathsize	small | normal | big | number v-unit	inherited
mathcolor	#rgb | #rrggbb | html-color-name	inherited
mathbackground	#rgb | #rrggbb | html-color-name	transparent

(See Section 2.1.3 for terminology and notation used in attribute value descriptions.)

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

When MathML rendering takes place in an environment where CSS is available, the mathematics style attributes can be viewed as predefined selectors for CSS style rules. See Section 7.4 and Appendix C for further discussion 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.

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

It is important to note that only certain combinations of character data and `mathvariant` attribute values make sense. For example, there is no clear cut rendering for a 'fraktur' alpha, or a 'bold italic' Kanji character. By design, the only cases that have an unambiguous interpretation are exactly the ones that correspond to SMP Math Alphanumeric Symbol characters, which are enumerated in Section 6.5. The `mathvariant` values `"initial"`, `"tailed"`, `"looped"` and `"stretched"` are expected to apply only to Arabic characters. In all other cases, it is suggested that renderers ignore the value of the `mathvariant` attribute if it is present. Similarly, authors should refrain from using the `mathvariant` attribute with characters that do not have SMP counterparts, since renderings may not be useful or predictable. In the very rare case that it is necessary to specify a font variant for other characters or symbols within an equation, external styling mechanisms such as CSS are generally preferable, or in the last resort, the deprecated style attributes of MathML 1 could be used.

Token elements also permit `xml:id`, `xref`, `class` and `style` attributes for compatibility with style sheet mechanisms, as described in Section 2.1.4. However, some care must be taken when using CSS generally. Using CSS to produce visual effects that alter the meaning of an equation should be especially avoided, since MathML is used in many non-CSS environments. Similarly, care should be taken to insure arbitrary document-wide style transformations do not affect mathematics expressions in such a way that meaning is altered.

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 2.0 doesn't specify the mechanism by which style information is inherited from the rendering environment. For example, one browser plug-in might choose to rely completely on the CSS inheritance mechanism and use the fully resolved CSS properties for rendering, while another application might only consult a style environment at the root node, and then use its own internal style inheritance rules.

Most MathML renderers will probably want to rely on some degree to additional, internal style processing algorithms. In particular, inheritance of the `mathvariant` attribute does not follow the CSS model. The default value for this attribute is `"normal"` (non-slanted) for all tokens except `mi`. For `mi` tokens, the default depends on the number of characters in tokens' content. (The deprecated `fontslant` attribute also behaves this way.) See Section 3.2.3 for details.

3.2.2.1 Deprecated style attributes on token elements

The MathML 1.01 style attributes listed below have been deprecated in MathML 2.0. In rendering environments that support CSS, it is preferable to use CSS to control the rendering properties corresponding to these attributes. However as explained above, direct manipulation of these rendering properties by whatever means should usually be avoided.

If both a new mathematics style attribute and conflicting deprecated attributes are given, the new math style attribute value should be used. For example

```
<mi fontweight='bold' mathvariant='normal'> a </mi>
```

should render in a normal weight font, and

```
<mi fontweight='bold' mathvariant='sans-serif'> a </mi>
```

should render in a normal weight sans serif font. In the example

```
<mi fontweight='bold' mathvariant='fraktur'> a1 </mi>
```

the `mathvariant` attribute still overrides `fontweight` attribute, even though `"fraktur"` generally shouldn't be applied to a '1' since there is no corresponding SMP Math Alphanumeric Symbol character. In the absence of fonts containing Fraktur digits, this would probably render as a Fraktur 'a' followed by a Roman '1' in most renderers.

The new mathematics style attributes also override deprecated 1.01 style attribute values that are inherited. Thus

```
<mstyle fontstyle='italic'>
  <mi mathvariant='bold'> a </mi>
</mstyle>
```

renders in a bold upright font, not a bold italic font.

At the same time, the MathML 1.01 attributes still serve a purpose. Since they correspond directly to rendering properties needed for mathematics layout, they are very useful for describing MathML layout rules and algorithms. For this reason, and for backward compatibility, the MathML rendering rules suggested in this chapter continue to be described in terms of the rendering properties described by these MathML 1.01 style attributes.

The deprecated attributes are:

Name	values	default
fontsize	number v-unit	inherited
fontweight	normal | bold	inherited
fontstyle	normal | italic	normal (*except on* `<mi>`)
fontfamily	string | css-fontfamily	inherited
color	#rgb | #rrggbb | html-color-name	inherited

The `fontsize` attribute specifies the desired font size. `v-unit` represents a unit of vertical length (see Section 2.1.3.3). The most common unit for specifying font sizes in typesetting is `pt` (points).

If the requested size 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.

Many MathML elements automatically change `fontsize` in some of their children; see the discussion of `scriptlevel` in the section on `mstyle`, Section 3.3.4.

The value of the `fontfamily` attribute should be the name of a font that may be available to a MathML renderer, or information that permits the renderer to select a font in some manner; acceptable values and their meanings are dependent on the specific renderer and rendering environment in use, and are not specified by MathML (but see the note about `css-fontfamily` below). (Note that the renderer's mechanism for finding fonts by name may be case-sensitive.)

If the value of `fontfamily` is not recognized by a particular MathML renderer, this should never be interpreted as a MathML error; rather, the renderer should either use a font that it considers to be a suitable substitute for the requested font, or ignore the attribute and act as if no value had been given.

Note that any use of the `fontfamily` attribute is unlikely to be portable across all MathML renderers. In particular, it should never be used to try to achieve the effect of a reference to a non-ASCII MathML character (for example, by using a reference to a character in some symbol font that maps ordinary characters to glyphs for non-ASCII characters). As a corollary to this principle, MathML renderers should attempt to always produce intelligible renderings for the MathML characters listed in Chapter 6, even when these characters are not available in the font family indicated. Such a rendering is always possible – as a last resort, a character can be rendered to appear as an XML-style entity reference using one of the entity names given for the same character in Chapter 6.

The symbol `css-fontfamily` refers to a legal value for the `font-family` property in CSS, which is a comma-separated list of alternative font family names or generic font types in order of preference, as documented in more detail in CSS[CSS2]. MathML renderers are encouraged to make use of the CSS syntax for specifying fonts when this is practical in their rendering environment, even if they do not otherwise support CSS. (See also the subsection CSS-compatible attributes within Section 2.1.3.3).

3.2.2.2 Color-related attributes

The `mathcolor` (and deprecated `color`) attribute controls the color in which the content of tokens is rendered. Additionally, when inherited from `mstyle` or from a MathML expression's rendering environment, it controls the color of all other drawing by MathML elements, including the lines or radical signs that can be drawn by `mfrac`, `mtable`, or `msqrt`.

The values of `mathcolor`, `color`, `mathbackground`, and `background` can be specified as a string consisting of '#' followed without intervening whitespace by either 1-digit or 2-digit hexadecimal values for the red, green, and blue components, respectively, of the desired color. The same number of digits must be used for each component. No whitespace is allowed between the '#' and the hexadecimal values. The hexadecimal digits are not case-sensitive. The possible 1-digit values range from 0 (component not present) to F (component fully present), and the possible 2-digit values range from 00 (component not present) to FF (component fully present), with the 1-digit value *x* being equivalent to the 2-digit value *xx* (rather than *x0*).

These attributes can also be specified as an `html-color-name`, which is defined below. Additionally, the keyword `"transparent"` may be used for the `background` attribute.

The color syntax described above is a subset of the syntax of the `color` and `background-color` properties of CSS. The `background-color` syntax is in turn a subset of the full CSS `background` property syntax, which also permits specification of (for example) background images with optional repeats. The more general attribute name `background` is used in MathML to facilitate possible extensions to the attribute's scope in future versions of MathML.

Color values on either attribute can also be specified as an `html-color-name`, that is, as 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.

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

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.

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

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

3.2.3.2 Attributes

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

Name	values	default
mathvariant	normal | bold | italic | bold-italic | double-struck | bold-fraktur | script | bold-script | fraktur | sans-serif | bold-sans-serif | sans-serif-italic | sans-serif-bold-italic | monospace | initial | tailed | looped | stretched	(depends on content; described below)
fontstyle (deprecated)	normal | italic	(depends on content; described below)

A typical graphical renderer would render an `mi` element as the characters in its content, with no extra spacing around the characters (except spacing associated with neighboring elements). The default `mathvariant` and `fontstyle` would (typically) be `"normal"` (non-slanted) unless the content is a single character, in which case it would be `"italic"`. Note that this rule for `mathvariant` and `fontstyle` attributes is specific to `mi` elements; the default value for the `mathvariant` and `fontstyle` attributes on other MathML token elements is `"normal"`.

Note that for purposes of determining equivalences of Math Alphanumeric Symbol characters (See Section 6.5 and Section 3.2.1.1) the value of the `mathvariant` attribute should be resolved first, including the special defaulting behavior described above.

3.2.3.3 Examples

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

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

Identifiers include function names such as 'sin'. Expressions such as 'sin *x*' should be written using the `⁡` operator (which also has the short name `⁡`) as shown below; see also the discussion of invisible operators in Section 3.2.5.

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

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

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

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

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

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

Use of special entity references for such constants can simplify the interpretation of MathML presentation elements. See Chapter 6 for a complete list of character entity references in MathML.

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.

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

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

3.2.4.2 *Attributes*

mn elements accept the attributes listed in Section 3.2.2.

A typical graphical renderer would render an mn element as the characters of its content, with no extra spacing around them (except spacing from neighboring elements such as mo). Unlike mi, mn elements are (typically) rendered in an unslanted font by default, regardless of their content.

3.2.4.3 *Examples*

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

3.2.4.4 *Numbers that should* not *be written using* mn *alone*

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

```
<mrow>
  <mn> 2 </mn>
  <mo> + </mo>
  <mrow>
    <mn> 3 </mn>
    <mo> &InvisibleTimes; </mo>
    <mi> &ImaginaryI; </mi>
```

```
  </mrow>
</mrow>
<mfrac> <mn> 1 </mn> <mn> 2 </mn> </mfrac>
<mi> &pi; </mi>
<mi> &ExponentialE; </mi>
```

3.2.5 Operator, Fence, Separator or Accent (`mo`)

3.2.5.1 Description

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

The term 'operator' as used in the present chapter means any symbol or notation that should render as an operator, and that is therefore representable by an `mo` element. That is, the term 'operator' includes any ordinary operator, fence, separator, or accent unless otherwise specified or clear from the context.

All such symbols are represented in MathML with `mo` elements since they are subject to essentially the same rendering attributes and rules; subtle distinctions in the rendering of these classes of symbols, when they exist, are supported using the boolean attributes `fence`, `separator` and `accent`, which can be used to distinguish these cases.

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

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

3.2.5.2 Attributes

`mo` elements accept the attributes listed in Section 3.2.2, and the additional attributes listed here. Most attributes get their default values from the Section 3.2.5.7, as described later in this section. When a dictionary entry is not found for a given `mo` element, the default value shown here in parentheses is used.

<table>
<tr><th>Name</th><th>values</th><th>default</th></tr>
<tr><td>form</td><td>prefix | infix | postfix</td><td>set by position of operator in an mrow (rule given below); used with mo content to index operator dictionary</td></tr>
<tr><td>fence</td><td>true | false</td><td>set by dictionary (false)</td></tr>
<tr><td>separator</td><td>true | false</td><td>set by dictionary (false)</td></tr>
<tr><td>lspace</td><td>number h-unit | namedspace</td><td>set by dictionary (thickmathspace)</td></tr>
<tr><td>rspace</td><td>number h-unit | namedspace</td><td>set by dictionary (thickmathspace)</td></tr>
<tr><td>stretchy</td><td>true | false</td><td>set by dictionary (false)</td></tr>
<tr><td>symmetric</td><td>true | false</td><td>set by dictionary (true)</td></tr>
<tr><td>maxsize</td><td>number [v-unit | h-unit] | namedspace | infinity</td><td>set by dictionary (infinity)</td></tr>
<tr><td>minsize</td><td>number [v-unit | h-unit] | namedspace</td><td>set by dictionary (1)</td></tr>
<tr><td>largeop</td><td>true | false</td><td>set by dictionary (false)</td></tr>
<tr><td>movablelimits</td><td>true | false</td><td>set by dictionary (false)</td></tr>
<tr><td>accent</td><td>true | false</td><td>set by dictionary (false)</td></tr>
</table>

h-unit represents a unit of horizontal length, and v-unit represents a unit of vertical length (see Section 2.1.3.2). namedspace is one of "veryverythinmathspace", "verythinmathspace", "thinmathspace", "mediummathspace", "thickmathspace", "verythickmathspace", or "veryverythickmathspace". These values can be set by using the mstyle element as is further discussed in Section 3.3.4.

If no unit is given with maxsize or minsize, the number is a multiplier of the normal size of the operator in the direction (or directions) in which it stretches. These attributes are further explained below.

Typical graphical renderers show all mo elements as the characters of their content, with additional spacing around the element determined from the attributes listed above. Detailed rules for determining operator spacing in visual renderings are described in a subsection below. As always, MathML does not require a specific rendering, and these rules are provided as suggestions for the convenience of implementors.

Renderers without access to complete fonts for the MathML character set may choose not to render an mo element as precisely the characters in its content in some cases. For example, <mo> ≤ </mo> might be rendered as <= to a terminal. However, as a general rule, renderers should attempt to render the content of an mo element as literally as possible. That is, <mo> ≤ </mo> and <mo> <= </mo> should render differently. (The first one should render as a single character representing a less-than-or-equal-to sign, and the second one as the two-character sequence <=.)

Issue (op): Line breaks typically occur before or after operators (including fences and separators). We could add an attribute linebreakstyle to specify information to the automatic linebreaking algorithm about the preferred method of linebreaking around an operator. The potential values are: before, after, duplicateThe default for these values could be specified in the operator dictionary. As with other mo attributes, this value can be set by using the mstyle element. To be useful, there needs to be a level of indirection so the general behavior could be changed easily without having to list a new value for all operators. One such possibility is to define three additional attributes: operatorlinebreakstyle, separatorlinebreakstyle, and fencelinebreakstyle. The problem with this idea is it breaks the simple model used to find default values for mo attributes.

3.2.5.3 Examples with ordinary operators

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

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

3.2.5.4 Examples with fences and separators

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

$(a+b)$

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

$[0,1)$

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

$f(x,y)$

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

3.2.5.5 Invisible operators

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

Full name	Short name	Examples of use
⁢	⁢	xy
&InvisiblePlus;	&ip;	$2\,\frac{3}{4}$
⁡	⁡	$f(x)\ \sin x$
⁣	⁣	m_{12}

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

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

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

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

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

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

```
  </mrow>
</msub>
```

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

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

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

3.2.5.6 Names for other special operators

MathML also includes `ⅆ` for use in an mo element representing the differential operator symbol usually denoted by 'd'. The reasons for explicitly using this special entity are similar to those for using the special entities for invisible operators described in the preceding section.

3.2.5.7 Detailed rendering rules for mo elements

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

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

The operator dictionary

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

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

Default value of the form attribute

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

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

Note that these rules make reference to the mrow in which the mo element lies. In some situations, this mrow might be an inferred mrow implicitly present around the arguments of an element such as msqrt or mtd.

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

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

Exception for embellished operators

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

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

- an mo element;
- or one of the elements msub, msup, msubsup, munder, mover, munderover, mmultiscripts, mfrac, or semantics (Section 5.2), whose first argument exists and is an embellished operator;
- or one of the elements mstyle, mphantom, or mpadded, such that an mrow containing the same arguments would be an embellished operator;
- or an maction element whose selected sub-expression exists and is an embellished operator;
- or an mrow whose arguments consist (in any order) of one embellished operator and zero or more space-like elements.

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

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

Rationale for definition of embellished operators

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

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

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

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

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

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

Spacing around an operator

The amount of 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. In TEX, these values can only be one of three values; typically they are 3/18em, 4/18em, and 5/18em. MathML does not impose this limit.

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

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

3.2.5.8 Stretching of operators, fences and accents

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

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

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

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

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

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

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

Example of stretchy attributes

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

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

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

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

Vertical Stretching Rules

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

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

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

Horizontal Stretching Rules

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

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

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

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

```
<mrow>
  <mi> x </mi>
  <munder>
    <mo> &RightArrow; </mo>
```

```
    <mtext> maps to </mtext>
  </munder>
  <mi> y </mi>
</mrow>
```

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

Rules Common to both Vertical and Horizontal Stretching

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

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

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

3.2.5.9 Other attributes of `mo`

The `largeop` attribute specifies whether the operator should be drawn larger than normal if `displaystyle="true"` in the current rendering environment. This roughly corresponds to TEX's `\displaystyle` style setting. MathML uses two attributes, `displaystyle` and `scriptlevel`, to control orthogonal presentation features that TEX encodes into one 'style' attribute with values `\displaystyle`, `\textstyle`, `\scriptstyle`, and `\scriptscriptstyle`. These attributes are discussed further in Section 3.3.4 describing the `mstyle` element. Note that these attributes can be specified directly on an `mstyle` element's start tag, but not on most other elements. Examples of large operators include `∫` and `∏`.

The `movablelimits` attribute specifies whether underscripts and overscripts attached to this `mo` element should be drawn as subscripts and superscripts when `displaystyle="false"`. `movablelimits="false"` means that underscripts and overscripts should never be drawn as subscripts and superscripts. In general, `displaystyle` is `"true"` for displayed mathematics and `"false"` for inline mathematics. Also, `displaystyle` is `"false"` by default within tables, scripts and fractions, and a few other exceptional situations detailed in Section 3.3.4. Thus, operators with `movablelimits="true"` will display with limits (i.e. underscripts and overscripts) in displayed mathematics, and with subscripts and superscripts in inline mathematics, tables, scripts and so on. Examples of operators that typically have `movablelimits="true"` are `∑`, `∏`, and `lim`.

The `accent` attribute determines whether this operator should be treated by default as an accent (diacritical mark) when used as an underscript or overscript; see `munder`, `mover`, and `munderover` (Section 3.4.4, Section 3.4.5 and Section 3.4.6).

The `separator` attribute may affect automatic linebreaking in renderers that position ordinary infix operators at the beginnings of broken lines rather than at the ends (that is, which avoid linebreaking just after such operators), since linebreaking should be avoided just before separators, but is acceptable just after them.

The `fence` attribute has no effect in the suggested visual rendering rules given here; it is not needed for properly rendering traditional notation using these rules. It is provided so that specific MathML renderers, especially non-visual renderers, have the option of using this information.

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 2.1.5. Whitespace that is intended to be rendered as part of an element's content must be represented by entity references or mspace elements (unless it consists only of single blanks between non-whitespace characters).

3.2.6.2 Attributes

mtext elements accept the attributes listed in Section 3.2.2.

See also the warnings about the legal grouping of 'space-like elements' in Section 3.2.7, and about the use of such elements for 'tweaking' or conveying meaning in Section 3.3.6.

3.2.6.3 Examples

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

3.2.6.4 Mixing text and mathematics

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

```
<mrow>
  <mo> there exists </mo>
  <mrow>
    <mrow>
      <mi> &delta; </mi>
      <mo> &gt; </mo>
      <mn> 0 </mn>
    </mrow>
    <mo> such that </mo>
    <mrow>
      <mrow>
        <mi> f </mi>
        <mo> &ApplyFunction; </mo>
```

```
        <mrow>
          <mo> ( </mo>
          <mi> x </mi>
          <mo> ) </mo>
        </mrow>
      </mrow>
      <mo> &lt; </mo>
      <mn> 1 </mn>
    </mrow>
  </mrow>
</mrow>
```

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

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

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

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

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

3.2.7 Space (mspace)

3.2.7.1 Description

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

3.2.7.2 Attributes

mspace accepts the attributes described in Section 3.2.2, but note that mathvariant and mathcolor have no effect. mathsize only affects the interpretation of units in sizing attributes (see Section 2.1.3.2).

mspace also accepts the attributes listed below

Name	values	default
spacing	string	""
width	number h-unit | namedspace	0em
height	number v-unit	0ex
depth	number v-unit	0ex
linebreak	auto | newline | indentingnewline | nobreak | goodbreak | badbreak	auto
indentto	id	auto
indent	left | center | right | auto | number h-unit	auto

"h-unit" and "v-unit" represent units of horizontal or vertical length, respectively (see Section 2.1.3.2).

The "spacing" attribute is a string-valued variable whose default value is the empty string (""). The spacing attribute specifies that the width of the space is the same as the length of the attribute value in the current font.

The total width of a mspace is given by the sum of both the "width" and "spacing" attributes. The "spacing" attribute does not affect the height or depth of the mspace.

The linebreak attribute is used to give a linebreaking hint to a visual renderer. The default value is "auto", which indicates that a renderer should use whatever default linebreaking algorithm it would normally use. The meanings of the other values are described in the table below.

Value	Description
newline	start a new line and do not indent
indentingnewline	start a new line and do indent
nobreak	do not allow a linebreak here
goodbreak	if a linebreak is needed on the line, here is a good spot
badbreak	if a linebreak is needed on the line, try to avoid breaking here

In the case when both dimensional attributes and a linebreaking attribute are set, the linebreaking attribute is ignored.

The spacing that normally follows an operator is not used at the end of a line. Similarly, the space that normally preceeds an operator is not used at the beginning of a line.

The indent and indentto attributes allow control over how lines are indented after a manual linebreak. They are only meaningful when linebreak is one of "newline", "indentingnewline". "goodbreak", or "badbreak". If both indent and indentto are specified, indentto is used when the referenced id is valid; the value of indent (or its default value) is used if the referenced xml:id is invalid.

indentto specifies that the left side of the next element should align with the left side of the element that xml:id references. The automatic linebreaking algorithm (if turned on) is free to linebreak between id and this element.

The values of xml:id must be unique within the scope of the entire document. MathML generators that create xml:id values should take care to create unique values.

The legal values of indent are:

Value	Meaning
left	Align the left side of the next line to the left side of the first line
center	Align the center of the next line to the center of the first line
right	Align the right side of the next line to the right side of the first line
auto	(default) Indent using the algorithm used by the automatic linebreaking algorithm.
number h-unit	Indent the amount specified by the argument. If a percentage is given, this is the percentage of the linewrapping width currently in effect.

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

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

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

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

Note the warning about the legal grouping of 'space-like elements' given below, and the warning about the use of such elements for 'tweaking' or conveying meaning in Section 3.3.6. See also the other elements that can render as whitespace, namely `mtext`, `mphantom`, and `maligngroup`.

3.2.7.3 Examples

```
<mspace spacing="00"/>
<mspace spacing="&times;000,00"/>
<mspace height="3ex" depth="2ex"/>

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

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

3.2.7.4 Definition of space-like elements

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

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

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

3.2.7.5 Legal grouping of space-like elements

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

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

```
<msup>
  <mi> x </mi>
```

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

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

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

See also the warning about ‘tweaking’ in Section 3.3.6.

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

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

Like all token elements, ms *does* trim and collapse whitespace in its content according to the rules of Section 2.1.5, so whitespace intended to remain in the content should be encoded as described in that section.

3.2.8.2 Attributes

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

Name	values	default
lquote	string	"
rquote	string	"

In visual renderers, the content of an ms element is typically rendered with no extra spacing added around the string, and the quote characters at the beginning and the end of the string. By default, the left and right quote characters are both the standard double quote character ". However, these characters can be changed with the lquote and rquote attributes respectively (which should be interpreted as opening and closing quotes, respectively).

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 \"".

3.2.9 Using images to represent symbols (mglyph)

3.2.9.1 Description

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

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

3.2.9.2 Attributes

`mglyph` elements accept the attributes listed in Section 3.2.2, but note that `mathvariant` and `mathcolor` have no effect. `mathsize` only affects the interpretation of units in sizing attributes (see Section 2.1.3.2). The background color, `mathbackground`, should show through if the specified image has transparency.

`mglyph` also accepts the additional attributes listed here.

Name	values	default
alt	string	required
src	URI	required
width	unsigned-number h-unit	from image
height	unsigned-number v-unit	from image
valign	number v-unit	0em

The `alt` attribute 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.

The `src` attribute specifies the location of the image resource; it may be a URI relative to the base-uri of the source of the MathML, if any. Examples of widely recognized image formats include GIF, JPEG and PNG; However, it may be advisable to omit the extension from the `src` uri, so that a user agent may use content-negotiation to choose the most appropriate format. The `src` uniquely identifies the `mglyph`; two `mglyphs` with the same values for `src` should be considered identical by applications that must determine whether two characters/glyphs are identical. The `alt` attribute should not be part of the identity test.

The `width` and `height` attributes specify the desired size of the glyph. They are both optional. If neither are given, the renderer should render the image at its natural size. If only one is given, the renderer should respect that dimension and choose the other dimension so as to preserve the aspect ratio of the image.

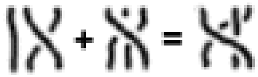
By default, the bottom of the image aligns to the current baseline. The `valign` attribute specifies the alignment point within the image. A positive value of `valign` shifts the bottom of the image below the current baseline, while a negative value will raise it above the baseline.

3.2.9.3 Example

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

```
<mrow>
  <mi><mglyph src="my-braid-23" alt="23braid"/></mi>
  <mo>+</mo>
  <mi><mglyph src="my-braid-123" alt="132braid"/></mi>
  <mo>=</mo>
  <mi><mglyph src="my-braid-13" alt="13braid"/></mi>
</mrow>
```

This might render as:



3.2.9.4 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. They are now optional, and will be ignored in most implementations.

Name	values
fontfamily	string | css-fontfamily
index	integer

The `fontfamily` and `index` attributes named a font and position within that font.

3.2.10 Line `mline`

3.2.10.1 Description

`mline` draws a horizontal line. The length and width of the line are specified as attributes.

3.2.10.2 Attributes

In addition to the attribute listed below, this element permits `"xml:id"`, `"xref"`, `"class"` and `"style"` attributes, as described in Section 2.1.4.

Name	values	default
linethickness	number [v-unit] | thin | medium | thick	1 (rule thickness)
spacing	string	""
length	number h-unit | namedspace	0

The `linethickness` attribute specifies how thick the line should be drawn.

The `spacing` attribute specifies that the length of the line is the same as the length of the attribute value in the current font.

The `length` attribute specifies the length of the line using a specification that is the same as the `width` attribute of `mspace` .

3.2.10.3 Examples

Here are some examples:

```
<mline spacing="000,000"/>
<mline length="2.5in"/>
```

Issue (generalization):A further generalization of `mline` would be to an arbitrary rectangular shape, where a height could also be specified instead of linethickness. With this generalization, `mline` would need to be renamed as would the "length" attribute. A minor restriction of this would be to add an attribute `direction` with values `"horizontal"` (default) and `"vertical"`.

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. See also `mfenced` (Section 3.3.8), which can effectively form an `mrow` containing its arguments separated by commas.

3.3.1.2 Attributes

This element only permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4, and the `dir` attribute as described in Section 3.1.5.1.

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

MathML is designed to allow renderers to automatically *linebreak* expressions (that is, to break excessively long expressions into several lines), without requiring authors to specify explicitly how this should be done. This is because linebreaking positions can't be chosen well without knowing the width of the display device and the current font size, which for many uses of MathML will not be known except by the renderer at the time of each rendering.

Determining good positions for linebreaks is complex, and rules for this are not described here; whether and how it is done is up to each MathML renderer. Typically, linebreaking will involve selection of 'good' points for insertion of linebreaks between successive arguments of `mrow` elements.

Although MathML does not require linebreaking or specify a particular linebreaking algorithm, it has several features designed to allow such algorithms to produce good results. These include the use of special entities for certain operators, including invisible operators (see Section 3.2.5), or for providing hints related to linebreaking when necessary (see Section 3.2.6), and the ability to use nested `mrow`s to describe sub-expression structure (see below).

`mrow` of one argument

MathML renderers are required to treat an `mrow` element containing exactly one argument as equivalent in all ways to the single argument occurring alone, provided there are no attributes on the `mrow` element's start tag. If there are attributes on the `mrow` element's start tag, no requirement of equivalence is imposed. This equivalence condition is intended to simplify the implementation of MathML-generating software such as template-based authoring tools.

It directly affects the definitions of embellished operator and space-like element and the rules for determining the default value of the `form` attribute of an `mo` element; see Section 3.2.5 and Section 3.2.7. See also the discussion of equivalence of MathML expressions in Section 2.3.

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.

Precise rule for proper grouping

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

Two adjacent operators (i.e. `mo` elements, possibly embellished), possibly separated by operands (i.e. anything other than operators), should occur in the same `mrow` only when the leading operator has an infix or prefix form (perhaps inferred), the following operator has an infix or postfix form, and the operators are listed in the same group of entries in the operator dictionary provided in Appendix B. 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. If this cannot be determined directly from the original expression, the operator that occurs later in the suggested operator dictionary (Appendix B) can be assumed to have a higher precedence for this purpose.

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

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

3.3.1.4 Examples

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

```
<mrow>
  <mrow>
    <mn> 2 </mn>
```

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

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

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

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

3.3.2 Fractions (`mfrac`)

3.3.2.1 Description

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

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

3.3.2.2 Attributes of `mfrac`

In addition to the attributes listed below, this element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
linethickness	number [v-unit] | thin | medium | thick	1 (rule thickness)
numalign	left | center | right	center
denomalign	left | center | right	center
bevelled	true | false	false

The `linethickness` attribute indicates the thickness of the horizontal ‘fraction bar’, or ‘rule’, typically used to render fractions. A fraction with `linethickness="0"` renders without the bar, and might be used within binomial coefficients. A `linethickness` greater than one might be used with nested fractions. These cases are shown below:

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

In general, the value of `linethickness` can be a number, as a multiplier of the default thickness of the fraction bar (the default thickness is not specified by MathML), or a number with a unit of vertical length (see Section 2.1.3.2), or one of the keywords `medium` (same as 1), `thin` (thinner than 1, otherwise up to the renderer), or `thick` (thicker than 1, otherwise up to the renderer).

The `numalign` and `denomalign` attributes control the horizontal alignment of the numerator and denominator respectively. Typically, numerators and denominators are centered, but a very long numerator or denominator might be displayed on several lines and a left alignment might be more appropriate for displaying them.

The `bevelled` attribute determines whether the fraction is displayed with the numerator above the denominator separated by a horizontal line or whether a diagonal line is used to separate a slightly raised numerator from a slightly lowered denominator. The latter form corresponds to the attribute value being `"true"` and provides for a more compact form for simple numerator and denominators. An example illustrating the bevelled form is show below:

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

Issue (arabic-bevelled-mfrac): Check with Azzeddine how a bevelled `mfrac` should be rendered.

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

The `mfrac` element sets `displaystyle` to `"false"`, or if it was already false increments `scriptlevel` by 1, within *numerator* and *denominator*. These attributes are inherited by every element from its rendering environment, but can be set explicitly only on the `mstyle` and `mtable` elements. (See Section 3.3.4.)

3.3.2.3 Examples

The examples shown above can be represented in MathML as:

```
<mrow>
   <mo> ( </mo>
   <mfrac linethickness="0">
      <mi> a </mi>
      <mi> b </mi>
   </mfrac>
   <mo> ) </mo>
</mrow>
<mfrac linethickness="2">
   <mfrac>
      <mi> a </mi>
      <mi> b </mi>
   </mfrac>
   <mfrac>
      <mi> c </mi>
      <mi> d </mi>
   </mfrac>
</mfrac>

<mfrac>
   <mn> 1 </mn>
   <mrow>
```

```
      <msup>
         <mi> x </mi>
         <mn> 3 </mn>
       </msup>
       <mo> + </mo>
       <mfrac>
         <mi> x </mi>
         <mn> 3 </mn>
       </mfrac>
   </mrow>
</mfrac>
<mo> = </mo>
<mfrac bevelled="true">
   <mn> 1 </mn>
   <mrow>
      <msup>
         <mi> x </mi>
         <mn> 3 </mn>
       </msup>
       <mo> + </mo>
       <mfrac>
         <mi> x </mi>
         <mn> 3 </mn>
       </mfrac>
   </mrow>
</mfrac>
```

A more generic example is:

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

3.3.3 Radicals (msqrt, mroot)

3.3.3.1 *Description*

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

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

The mroot element requires exactly 2 arguments. However, msqrt accepts any number of arguments; if this number is not 1, its contents are treated as a single 'inferred mrow' containing its arguments, as described in Section 3.1.3.

3.3.3.2 Attributes

This element only permits xml:id, xref, class and style attributes, as described in Section 2.1.4.

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 all its arguments. These attributes are inherited by every element from its rendering environment, but can be set explicitly only on mstyle. (See Section 3.3.4.)

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.

3.3.4 Style Change (mstyle)

3.3.4.1 Description

The mstyle element is used to make style changes that affect the rendering of its contents. mstyle can be given any attribute accepted by any MathML presentation element provided that the attribute value is inherited, computed or has a default value; presentation element attributes whose values are required are not accepted by the mstyle element. In addition mstyle can also be given certain special attributes listed below.

The mstyle element accepts any number of arguments. If this number is not 1, its contents are treated as a single 'inferred mrow' formed from all its arguments, as described in Section 3.1.3.

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

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

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

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

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

3.3.4.2 Attributes

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

This element also accepts `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

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
displaystyle	true | false	inherited
scriptsizemultiplier	number	0.71
scriptminsize	number v-unit	8pt
background	#rgb | #rrggbb | transparent | html-color-name	transparent
veryverythinmathspace	number h-unit	0.0555556em
verythinmathspace	number h-unit	0.111111em
thinmathspace	number h-unit	0.166667em
mediummathspace	number h-unit	0.222222em
thickmathspace	number h-unit	0.277778em
verythickmathspace	number h-unit	0.333333em
veryverythickmathspace	number h-unit	0.388889em

`scriptlevel` and `displaystyle`

MathML uses two attributes, `displaystyle` and `scriptlevel`, to control orthogonal presentation features that TEX encodes into one `style` attribute with values \displaystyle, \textstyle, \scriptstyle, and \scriptscriptstyle. The corresponding values of `displaystyle` and `scriptlevel` for those TEX styles would be `"true"` and `"0"`, `"false"` and `"0"`, `"false"` and `"1"`, and `"false"` and `"2"`, respectively.

The main effect of the `displaystyle` attribute is that it determines the effect of other attributes such as the `largeop` and `movablescripts` attributes of `mo`. The main effect of the `scriptlevel` attribute 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.) More sophisticated renderers may also choose to use these attributes in other ways, such as rendering expressions with displaystyle="false" in a more vertically compressed manner.

These attributes are given initial values for the outermost expression of an instance of MathML based on its rendering environment. A short list of layout schemata described below modify these values for some of their sub-expressions. Otherwise, values are determined by inheritance whenever they are not directly specified on a given element's start tag.

For an instance of MathML embedded in a textual data format (such as HTML) in 'display' mode, i.e. in place of a paragraph, displaystyle = "true" and scriptlevel = "0" for the outermost expression of the embedded MathML; if the MathML is embedded in 'inline' mode, i.e. in place of a character, displaystyle = "false" and scriptlevel = "0" for the outermost expression. See Section 2.5.2 for further discussion of the distinction between 'display' and 'inline' embedding of MathML and how this can be specified in particular instances. In general, a MathML renderer may determine these initial values in whatever manner is appropriate for the location and context of the specific instance of MathML it is rendering, or if it has no way to determine this, based on the way it is most likely to be used; as a last resort it is suggested that it use the most generic values displaystyle = ""true"" and scriptlevel = ""0"".

The MathML layout schemata that typically display some of their arguments in smaller type or with less vertical spacing, namely the elements for scripts, fractions, radicals, and tables or matrices, set displaystyle to "false", and in some cases increase scriptlevel, for those arguments. The new values are inherited by all sub-expressions within those arguments, unless they are overridden.

The specific rules by which each element modifies displaystyle and/or scriptlevel are given in the specification for each element that does so; the complete list of elements that modify either attribute are: the 'scripting' elements msub, msup, msubsup, munder, mover, munderover, and mmultiscripts; and the elements mfrac, mroot, and mtable.

When mstyle is given a scriptlevel attribute with no sign, it sets the value of scriptlevel within its contents to the value given, which must be a nonnegative integer. When the attribute value consists of a sign followed by an integer, the value of scriptlevel is incremented (for '+') or decremented (for '-') by the amount given. The incremental syntax for this attribute is an exception to the general rules for setting inherited attributes using mstyle, and is not allowed by any other attribute on mstyle.

Whenever the scriptlevel is changed, either automatically or by being explicitly incremented, decremented, or set, the current font size is multiplied by the value of scriptsizemultiplier to the power of the change in scriptlevel. For example, if scriptlevel is increased by 2, the font size is multiplied by scriptsizemultiplier twice in succession; if scriptlevel is explicitly set to 2 when it had been 3, the font size is divided by scriptsizemultiplier. References to fontsize in this section should be interpreted to mean either the fontsize attribute or the mathsize attribute.

The default value of scriptsizemultiplier is less than one (in fact, it is approximately the square root of 1/2), resulting in a smaller font size with increasing scriptlevel. To prevent scripts from becoming unreadably small, the font size is never allowed to go below the value of scriptminsize as a result of a change to scriptlevel, though it can be set to a lower value using the fontsize attribute (Section 3.2.2) on mstyle or on token elements. If a change to scriptlevel would cause the font size to become lower than scriptminsize using the above formula, the font size is instead set equal to scriptminsize within the sub-expression for which scriptlevel was changed.

In the syntax for scriptminsize, v-unit represents a unit of vertical length (as described in Section 2.1.3.2). The most common unit for specifying font sizes in typesetting is pt (points).

Explicit changes to the fontsize attribute have no effect on the value of scriptlevel.

Further details on `scriptlevel` for renderers

For MathML renderers that support CSS style sheets, or some other analogous style sheet mechanism, absolute or relative changes to `fontsize` (or other attributes) may occur implicitly on any element in response to a style sheet. Changes to `fontsize` of this kind also have no effect on `scriptlevel`. A style sheet-induced change to `fontsize` overrides `scriptminsize` in the same way as for an explicit change to `fontsize` in the element's start tag (discussed above), whether it is specified in the style sheet as an absolute or a relative change. (However, any subsequent `scriptlevel`-induced change to `fontsize` will still be affected by it.) As is required for inherited attributes in CSS, the style sheet-modified `fontsize` is inherited by child elements.

If the same element is subject to both a style sheet-induced and an automatic (`scriptlevel`-related) change to its own `fontsize`, the `scriptlevel`-related change is done first – in fact, in the simplest implementation of the element-specific rules for `scriptlevel`, this change would be done by the element's parent as part of producing the rendering properties it passes to the given element, since it is the parent element that knows whether `scriptlevel` should be changed for each of its child elements.

If the element's own `fontsize` is changed by a style sheet and it also changes `scriptlevel` (and thus `fontsize`) for one of its children, the style sheet-induced change is done first, followed by the change inherited by that child. If more than one child's `scriptlevel` is changed, the change inherited by each child has no effect on the other children. (As a mnemonic rule that applies to a 'parse tree' of elements and their children, style sheet-induced changes to `fontsize` can be associated to nodes of the tree, i.e. to MathML elements, and `scriptlevel`-related changes can be associated to the edges between parent and child elements; then the order of the associated changes corresponds to the order of nodes and edges in each path down the tree.) For general information on the relative order of processing of properties set by style sheets versus by attributes, see the appropriate subsection of CSS-compatible attributes in Section 2.1.3.3.

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`, `fontsize`.

Note that `scriptlevel` can, in principle, attain any integral value by being decremented sufficiently, even though it can only be explicitly set to nonnegative values. Negative values of `scriptlevel` generated in this way are legal and should work as described, generating font sizes larger than those of the surrounding expression. Since `scriptlevel` is initially 0 and never decreases automatically, it will always be nonnegative unless it is decremented past 0 using `mstyle`.

Explicit decrements of `scriptlevel` after the font size has been limited by `scriptminsize` as described above would produce undesirable results. This might occur, for example, in a representation of a continued fraction, in which the scriptlevel was decremented for part of the denominator back to its value for the fraction as a whole, if the continued fraction itself was located in a place that had a high `scriptlevel`. To prevent this problem, MathML renderers should, when decrementing `scriptlevel`, use as the initial font size the value the font size would have had if it had never been limited by `scriptminsize`. They should not, however, ignore the effects of explicit settings of `fontsize`, even to values below `scriptminsize`.

Since MathML renderers may be unable to make use of arbitrary font sizes with good results, they may wish to modify the mapping from scriptlevel to fontsize to produce better renderings in their judgment. In particular, if fontsizes have to be rounded to available values, or limited to values within a range, the details of how this is done are up to the renderer. Renderers should, however, ensure that a series of incremental changes to `scriptlevel` resulting in its return to the same value for some sub-expression that it had in a surrounding expression results in the same fontsize for that sub-expression as for the surrounding expression.

Color and background attributes

Color and background attributes are discussed in Section 3.2.2.2.

Precise background region not specified

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

Meaning of named mathspaces

The spacing between operators is often one of a small number of potential values. MathML names these values and allows their values to be changed. Because the default values for spacing around operators that are given in the operator dictionary Appendix B are defined using these named spaces, changing their values will produce tighter or looser spacing. These values can be used anywhere a `h-unit` or `v-unit` unit is allowed. See Section 2.1.3.2.

The predefined `namedspace`s are: `"negativeveryverythinmathspace"`, `"negativeverythinmathspace"`, `"negativethinmathspace"`, `"negativemediummathspace"`, `"negativethickmathspace"`, `"negativeverythickmathspace"`, `"negativeveryverythickmathspace"`, `"veryverythinmathspace"`, `"verythinmathspace"`, `"thinmathspace"`, `"mediummathspace"`, `"thickmathspace"`, `"verythickmathspace"`, or `"veryverythickmathspace"`. The default values of `"veryverythinmathspace"`... `"veryverythickmathspace"` are 1/18em...7/18em, respectively.

3.3.4.3 Examples

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

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

can be rewritten using `mstyle` as:

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

3.3.5 Error Message (`merror`)

3.3.5.1 Description

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

merror accepts any number of arguments; if this number is not 1, its contents are treated as a single 'inferred mrow' as described in Section 3.1.3.

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

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

No specific error message format is suggested here, but as with error messages from any program, the format should be designed to make as clear as possible (to a human viewer of the rendered error message) what was wrong with the input and how it can be fixed. If the erroneous input contains correctly formatted subsections, it may be useful for these to be preprocessed normally and included in the error message (within the contents of the merror element), taking advantage of the ability of merror to contain arbitrary MathML expressions rather than only text.

3.3.5.2 Attributes

This element only permits xml:id, xref, class and style attributes, as described in Section 2.1.4.

3.3.5.3 Example

If a MathML syntax-checking preprocessor received the input

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

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

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

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

3.3.6 Adjust Space Around Content (mpadded)

3.3.6.1 Description

An mpadded element renders the same as its content, but with its overall size and other dimensions (such as baseline position) modified according to its attributes. The mpadded element does not rescale (stretch or shrink) its content; its only effect is to modify the apparent size and position of the 'bounding box' around its content, so as to affect the relative position of the content with respect to the surrounding elements. While the name of the element reflects the use of `mpadded` to add 'padding', or extra space, around its content, by adding negative 'padding' it is possible to cause the content of `mpadded` to be rendered outside the `mpadded` element's bounding box; see below for warnings about several potential pitfalls of this effect.

The `mpadded` element accepts any number of arguments; if this number is not 1, its contents are treated as a single 'inferred `mrow`' as described in Section 3.1.3.

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

3.3.6.2 Attributes

In addition to the attributes listed below, this element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
width	[+ | -] *unsigned-number* (% [*pseudo-unit*] | *pseudo-unit* | *h-unit* | namedspace)	same as content
lspace	[+ | -] *unsigned-number* (% [*pseudo-unit*] | *pseudo-unit* | *h-unit* | namedspace)	same as content
height	[+ | -] *unsigned-number* (% [*pseudo-unit*] | *pseudo-unit* | *v-unit*)	same as content
depth	[+ | -] *unsigned-number* (% [*pseudo-unit*] | *pseudo-unit* | *v-unit*)	same as content

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

These attributes modify the size and position of the 'bounding box' of the `mpadded` element. The typographical layout parameters defined by these attributes are described in the next subsection. Depending on the format of the attribute value, a dimension may be set to a new value, or to an incremented or decremented version of the content's corresponding dimension. Values may be specified as multiples or percentages of any of the dimensions of the normal rendering of the element's content (using so-called 'pseudo-units'), or they can be set directly using standard units Section 2.1.3.2.

If an attribute value begins with a `+` or `-` sign, it specifies an increment or decrement of the corresponding dimension by the following length value (interpreted as explained below). Otherwise, the corresponding dimension is set directly to the following length value. Note that the `+` and `-` do not mean that the following value is positive or negative, even when an explicit length unit (*h-unit* or *v-unit*) is given.

Length values (after the optional sign, which is not part of the length value) can be specified in several formats. Each format begins with an *unsigned-number*, which may be followed by a `%` sign and an optional 'pseudo-unit' (denoted by *pseudo-unit* in the attribute syntaxes above), by a pseudo-unit alone, or by one of the length units (denoted by *h-unit* or *v-unit*) specified in Section 2.1.3.2, not including `%`. The possible pseudo-units are the keywords `width`, `advancewidth`, `lspace`, `height`, and `depth`; they each represent the length of the same-named dimension of the `mpadded` element's content (not of the `mpadded` element itself). The lengths represented by *h-unit* or *v-unit* are described in Section 2.1.3.2.

In any of these formats, the length value specified is the product of the specified number and the length represented by the unit or pseudo-unit. The result is multiplied by 0.01 if `%` is given. If no pseudo-unit is given after `%`, the one with the same name as the attribute being specified is assumed.

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

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

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

3.3.6.3 Meanings of size and position attributes

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

The content of an `mpadded` element defines some mathematical notation (e.g. a character, a fraction, an expression, etc.) that can be regarded as single typographical element with a positioning point at a fixed relative location to its natural visual bounding box, and an advance width that determines the natural placement of the next typographical element following it. The advance width, like the positioning point, is generally at a fixed location relative to the visual bounding box.

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

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

The `width` attribute refers to the horizontal width of the natural visual bounding box of the `mpadded` element's content. Note that decreasing the width will cause clipping to take place when rendering the child content. For example, setting the width to 0 would entirely suppress the rendering of the child content. Decreasing the width should generally be avoided.

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

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

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

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

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

3.3.6.4 Warning: nonportability of 'tweaking'

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

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

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

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

In cases where the temptation to improve spacing proves too strong, careful use of `mpadded`, `mphantom`, or the alignment elements (Section 3.5.5) may give more portable results than the direct insertion of extra space using `mspace` or `mtext`. Advice given to the implementors of MathML renderers might be still more productive, in the long run.

3.3.6.5 Warning: spacing should not be used to convey meaning

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

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

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

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

If such 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.

For example, the MathML expression

```
<mrow>
  <mi> C </mi>
  <mpadded lspace="+.5em" advancewidth="0em">
    <mtext> | </mtext>
  </mpadded>
</mrow>
```

forms an overstruck symbol in violation of the policy stated above; it might be intended to represent the set of complex numbers for a MathML renderer that lacks support for the standard symbol used for this purpose. This kind of construct should always be avoided in MathML, for the reasons stated above; indeed, it should never be necessary for standard symbols, since a MathML renderer with no better method of rendering them is free to use overstriking internally, so that it can still support general MathML input.

However, if for whatever reason such a construct is used in MathML, it should always be enclosed in a `semantics` element such as

```
<semantics>
    <mrow>
      <mi> C </mi>
        <mpadded lspace="+.5em" advancewidth="0em">
        <mtext> | </mtext>
      </mpadded>
    </mrow>
  <annotation-xml encoding="MathML-Presentation">
    <mi> &Copf; </mi>
  </annotation-xml>
</semantics>
```

which provides an alternative, standard encoding for the desired symbol, which is much more easily interpreted than the construct using negative spacing. The alternative encoding in this example uses MathML presentation elements; the content elements described in Chapter 4 should also be considered.

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 `fontweight`) used to distinguish one variable from another.

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 any number of arguments; if this number is not 1, its contents are treated as a single 'inferred `mrow`' formed from all its arguments, as described in Section 3.1.3.

3.3.7.2 Attributes

This element only permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

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

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

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

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

3.3.7.3 Examples

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. Thus, any mfenced element is completely equivalent to an expanded form described below; either form can be used in MathML, at the convenience of an author or of a MathML-generating program. A MathML renderer is required to render either of these forms in exactly the same way.

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

3.3.8.2 Attributes

In addition to the attributes listed below, this element permits xml:id, xref, class and style attributes, as described in Section 2.1.4.

Name	values	default
open	string	(
close	string	)
separators	character *	,

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>
```

The `"opening-fence"` and `"closing-fence"` are arbitrary strings. (Since they are used as the content of `mo` elements, any whitespace they contain will be trimmed and collapsed as described in Section 2.1.5.)

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

The value of `separators` is a sequence of zero or more separator characters (or entity references), optionally separated by whitespace. Each `sep#i` consists of exactly one character or entity reference. Thus, `separators=",;"` is equivalent to `separators=" , ; "`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.3.8.3 Examples

$(a+b)$

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

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

[0,1)

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

$f(x,y)$

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

3.3.9 Enclose Expression Inside Notation (`menclose`)

3.3.9.1 Description

The `menclose` element renders its content inside the enclosing notation specified by its `notation` attribute. `menclose` accepts any number of arguments; if this number is not 1, its contents are treated as a single ‘inferred `mrow`’ containing its arguments, as described in Section 3.1.3.

3.3.9.2 Attributes

In addition to the attributes listed below, this element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

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

Name	values	default
notation	longdiv | actuarial | radical | box | roundedbox | circle | left | right | top | bottom | updiagonalstrike | downdiagonalstrike | verticalstrike | horizontalstrike | madruwb	longdiv

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. For example, `notation="circle horizontalstrike"` should result in circle around the contents of `menclose` with a horizontal line through the contents.

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

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

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

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

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

should be rendered roughly as

$$[Imageof12factorialinArabicstyle]$$

.

3.3.9.3 Examples

The following markup might be used to encode an elementary US-style long division problem.

```
<mtable columnspacing='0pt' rowspacing='0pt'>
<mtr>
  <mtd></mtd>
  <mtd columnalign='right'><mn>10</mn></mtd>
</mtr>
<mtr>
  <mtd columnalign='right'><mn>131</mn></mtd>
  <mtd columnalign='right'>
    <menclose notation='longdiv'><mn>1413</mn></menclose>
  </mtd>
</mtr>
<mtr>
  <mtd></mtd>
  <mtd columnalign='right'>
    <mrow>
    <munder>
      <mn>131</mn>
      <mo> &UnderBar; </mo>
    </munder>
    <mphantom><mn>3</mn></mphantom>
    </mrow>
  </mtd>
</mtr>
<mtr>
  <mtd></mtd>
  <mtd columnalign='right'><mn>103</mn></mtd>
</mtr>
</mtable>
```

This might be rendered roughly as:

$$\begin{array}{r} 10 \\ 131\overline{)1413} \\ \underline{131} \\ 103 \end{array}$$

An example of using `menclose` for actuarial notation is

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

which renders roughly as

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

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 under `mstyle` (Section 3.3.4). 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 syntax for the `msub` element is:

```
<msub> base  subscript  </msub>
```

3.4.1.2 Attributes

In addition to the attributes listed below, this element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
subscriptshift	number v-unit	automatic (typical unit is ex)

The `subscriptshift` attribute specifies the minimum amount to shift the baseline of *subscript* down.

v-unit represents a unit of vertical length (see Section 2.1.3.2).

The `msub` element increments `scriptlevel` by 1, and sets `displaystyle` to `"false"`, within *subscript*, but leaves both attributes unchanged within *base*. (These attributes are inherited by every element through its rendering environment, but can be set explicitly only on `mstyle`; see Section 3.3.4.)

3.4.2 Superscript (`msup`)

3.4.2.1 Description

The syntax for the `msup` element is:

```
<msup> base  superscript  </msup>
```

3.4.2.2 Attributes

In addition to the attributes listed below, this element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
superscriptshift	number v-unit	automatic (typical unit is ex)

The `superscriptshift` attribute specifies the minimum amount to shift the baseline of *superscript* up.

v-unit represents a unit of vertical length (see Section 2.1.3.2).

The `msup` element increments `scriptlevel` by 1, and sets `displaystyle` to `"false"`, within *superscript*, but leaves both attributes unchanged within *base*. (These attributes are inherited by every element through its rendering environment, but can be set explicitly only on `mstyle`; see Section 3.3.4.)

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. 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 syntax for the `msubsup` element is:

```
<msubsup>  base   subscript   superscript  </msubsup>
```

3.4.3.2 Attributes

In addition to the attributes listed below, this element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
subscriptshift	number v-unit	automatic (typical unit is ex)
superscriptshift	number v-unit	automatic (typical unit is ex)

The `subscriptshift` attribute specifies the minimum amount to shift the baseline of *subscript* down. The `superscriptshift` attribute specifies the minimum amount to shift the baseline of *superscript* up.

v-unit represents a unit of vertical length (see Section 2.1.3.2).

The `msubsup` element increments `scriptlevel` by 1, and sets `displaystyle` to `"false"`, within *subscript* and *superscript*, but leaves both attributes unchanged within *base*. (These attributes are inherited by every element through its rendering environment, but can be set explicitly only on `mstyle`; see Section 3.3.4.)

3.4.3.3 Examples

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

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

would be represented as

```
<mrow>
  <msubsup>
    <mo> &int; </mo>
    <mn> 0 </mn>
    <mn> 1 </mn>
  </msubsup>
```

```
  <mrow>
    <msup>
      <mi> &ExponentialE; </mi>
      <mi> x </mi>
    </msup>
    <mo> &InvisibleTimes; </mo>
    <mrow>
      <mo> &DifferentialD; </mo>
      <mi> x </mi>
    </mrow>
  </mrow>
</mrow>
```

3.4.4 Underscript (munder)

3.4.4.1 Description

The syntax for the munder element is:

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

3.4.4.2 Attributes

In addition to the attributes listed below, this element permits xml:id, xref, class and style attributes, as described in Section 2.1.4.

Name	values	default
accentunder	true | false	automatic
align	left | right | center	center

The accentunder attribute controls whether *underscript* is drawn as an 'accent' or as a limit. The main difference between an accent and a limit is that the limit is reduced in size whereas an accent is the same size as the base. A second difference is that the accent is drawn closer to the base.

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

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

If the 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 ∑.

Within *underscript*, munder always sets displaystyle to "false", but increments scriptlevel by 1 only when accentunder is "false". Within *base*, it always leaves both attributes unchanged. (These attributes are inherited by every element through its rendering environment, but can be set explicitly only on mstyle; see Section 3.3.4.)

The align attribute specifies whether the script is aligned left, center, or right under/over the base.

3.4.4.3 Examples

The MathML representation for the example shown above is:

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

3.4.5 Overscript (`mover`)

3.4.5.1 Description

The syntax for the `mover` element is:

```
<mover>  base   overscript  </mover>
```

3.4.5.2 Attributes

In addition to the attributes listed below, this element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
accent	true | false	automatic
align	left | right | center	center

The `accent` attribute controls whether *overscript* is drawn as an 'accent' (diacritical mark) or as a limit. The main difference between an accent and a limit is that the limit is reduced in size whereas an accent is the same size as the base. A second difference is that the accent is drawn closer to the base. This is shown below (accent versus limit): $\hat{x}$ versus $\hat{x}$.

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

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

If the 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 `∑`.

Within *overscript*, `mover` always sets `displaystyle` to `"false"`, but increments `scriptlevel` by 1 only when `accent` is `"false"`. Within *base*, it always leaves both attributes unchanged. (These attributes are inherited by every element through its rendering environment, but can be set explicitly only on `mstyle`; see Section 3.3.4.)

The `align` attribute specifies whether the script is aligned left, center, or right under/over the base.

3.4.5.3 Examples

The MathML representation for the examples shown above is:

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

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

```
    <mo> &OverBrace; </mo>
  </mover>
</mrow>
```

3.4.6 Underscript-overscript Pair (munderover)

3.4.6.1 Description

The syntax for the munderover element is:

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

3.4.6.2 Attributes

In addition to the attributes listed below, this element permits xml:id, xref, class and style attributes, as described in Section 2.1.4.

Name	values	default
accent	true | false	automatic
accentunder	true | false	automatic
align	left | right | center	center

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

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

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

The accent and accentunder attributes have the same effect as the attributes with the same names on mover (Section 3.4.5) and munder (Section 3.4.4), respectively. Their default values are also computed in the same manner as described for those elements, with the default value of accent depending on *overscript* and the default value of accentunder depending on *underscript*.

If the 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 accent and accentunder attributes are ignored. This is often used for limits on symbols such as ∑.

Within *underscript*, munderover always sets displaystyle to "false", but increments scriptlevel by 1 only when accentunder is "false". Within *overscript*, munderover always sets displaystyle to "false", but increments scriptlevel by 1 only when accent is "false". Within *base*, it always leaves both attributes unchanged. (These attributes are inherited by every element through its rendering environment, but can be set explicitly only on mstyle; see Section 3.3.4).

The align attribute specifies whether the script is aligned left, center, or right under/over the base.

3.4.6.3 Examples

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

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

3.4.7 Prescripts and Tensor Indices (mmultiscripts)

3.4.7.1 Description

The syntax for the mmultiscripts element is:

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

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

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

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

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

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

3.4.7.2 Attributes

Same as the attributes of msubsup. See Section 3.4.3.2.

The mmultiscripts element increments scriptlevel by 1, and sets displaystyle to "false", within each of its arguments except *base*, but leaves both attributes unchanged within *base*. (These attributes are inherited by every element through its rendering environment, but can be set explicitly only on mstyle; see Section 3.3.4.)

3.4.7.3 Examples

Two examples of the use of mmultiscripts are:

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

```
<mrow>
  <mmultiscripts>
    <mi> F </mi>
    <mn> 1 </mn>
    <none/>
    <mprescripts/>
    <mn> 0 </mn>
    <none/>
  </mmultiscripts>
  <mo> &ApplyFunction; </mo>
  <mrow>
    <mo> ( </mo>
    <mrow>
      <mo> ; </mo>
      <mi> a </mi>
      <mo> ; </mo>
      <mi> z </mi>
    </mrow>
    <mo> ) </mo>
  </mrow>
</mrow>
```

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

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

An additional example of mmultiscripts shows how the binomial coefficient

$$[binomial(5,12) inenglishstyle]$$

can be displayed in Arabic style

$[binomial(5,12)inArabicstyle]$

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

3.5 Tabular Math

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

While somewhat similar to tables, the alignment issues for representing some two-dimensioal layouts in elemenatary such as addition and multiplication differ in some important ways. `mcolumn` is used for tabular elementary math notations. See Section 3.7 for a discussion about elementary math notations.

In addition to the table elements mentiond 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` element could infer `mtr` elements around its arguments, and the `mtr` element could infer `mtd` elements. In other words, if some argument to an `mtable` was not an `mtr` element, a MathML application was to assume a row with a single column (i.e. the argument was effectively wrapped with an inferred `mtr`). Similarly, if some argument to a (possibly inferred) `mtr` element was not an `mtd` element, that argument was to be treated as a table entry by wrapping it with an inferred `mtd` element. MathML 2.0 deprecates the inference of `mtr` and `mtd` elements; `mtr` and `mtd` elements must be used inside of `mtable` and `mtr` respectively.

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.

3.5.1.2 Attributes

In addition to the attributes listed below, this element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
align	(top | bottom | center | baseline | axis) [rownumber]	axis
rowalign	(top | bottom | center | baseline | axis) +	baseline
columnalign	(left | center | right) +	center
groupalign	group-alignment-list-list	left
alignmentscope	(true | false) +	true
columnwidth	(auto | number h-unit | namedspace | fit) +	auto
width	auto | number h-unit	auto
rowspacing	(number v-unit) +	1.0ex
columnspacing	(number h-unit | namedspace) +	0.8em
rowlines	(none | solid | dashed) +	none
columnlines	(none | solid | dashed) +	none
frame	none | solid | dashed	none
framespacing	(number h-unit | namedspace) (number v-unit | namedspace)	0.4em 0.5ex
equalrows	true | false	false
equalcolumns	true | false	false
displaystyle	true | false	false
side	left | right | leftoverlap | rightoverlap	right
minlabelspacing	number h-unit | namedspace	0.8em

Note that the default value for each of `rowlines`, `columnlines` and `frame` is the literal string 'none', meaning that the default is to render no lines, rather than that there is no default.

As described in Section 2.1.3, the notation `(x | y)+` means one or more occurrences of either `x` or `y`, separated by whitespace. For example, possible values for `columnalign` are `"left"`, `"left left"`, and `"left right center center"`. If there are more entries than are necessary (e.g. more entries than columns for `columnalign`), then only the first entries will be used. If there are fewer entries, then the last entry is repeated as often as necessary. For example, if `columnalign=`"right center" and the table has three columns, the first column will be right aligned and the second and third columns will be centered. The label in a `mlabeledtr` is not considered as a column in the table and the attribute values that apply to columns do not apply to labels.

The `align` attribute specifies where to align the table with respect to its environment. `"axis"` means to align the 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. The center of the table is the midpoint of the table's vertical extent.) `"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"` between 1 and *n* (for a table with *n* rows), the specified row is aligned in the way described above, rather than the table as a whole; the top (first) row is numbered 1, and the bottom (last) row is numbered *n*. The same is true if the row number is negative, between -1 and -*n*, except that the bottom row is referred to as -1 and the top row as -*n*. Other values of `"rownumber"` are illegal.

The `rowalign` attribute specifies how the entries in each row should be aligned. For example, `"top"` means that the tops of each entry in each row should be aligned with the tops of the other entries in that row. The `columnalign` attribute specifies how the entries in each column should be aligned.

The `groupalign` and `alignmentscope` attributes are described with the alignment elements, `maligngroup` and `malignmark`, in Section 3.5.5.

The `columnwidth` attribute specifies how wide a column should be. The `"auto"` value means that the column should be as wide as needed, which is the default. If an explicit value is given, then the column is exactly that wide and the contents of that column are made to fit in that width. The contents are linewrapped or clipped at the discretion of the renderer. If `"fit"` is given as a value, the remaining page width after subtracting the widths for columns specified as `"auto"` and/or specific widths is divided equally among the `"fit"` columns and this value is used for the column width. If insufficient room remains to hold the contents of the "fit" columns, renderers may linewrap or clip the contents of the "fit" columns. When the columnwidth is specified as a percentage, the value is relative to the width of the table. 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.

The width attribute specifies the desired width of the entire table and is intended for visual user agents. When the value is a percentage value, the value is relative to the horizontal space a MathML renderer has available for the math element. When the value is "auto", the MathML renderer should calculate the table width from its contents using whatever layout algorithm it chooses.

MathML 2.0 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.

The rowspacing and columnspacing attributes specify how much space should be added between each row and column. However, spacing before the first row and after the last row (i.e. at the top and bottom of the table) is given by the second number in the value of the framespacing attribute, and spacing before the first column and after the last column (i.e. on the left and on the right of the table) is given by the first number in the value of the framespacing attribute.

In those attributes' syntaxes, *h-unit* or *v-unit* represents a unit of horizontal or vertical length, respectively (see Section 2.1.3.2). The units shown in the attributes' default values (em or ex) are typically used.

The rowlines and columnlines attributes specify whether and what kind of lines should be added between each row and column. Lines before the first row or column and after the last row or column are given using the frame attribute.

If a frame is desired around the table, the frame attribute is used. If the attribute value is not 'none', then framespacing is used to add spacing between the lines of the frame and the first and last rows and columns of the table. If frame="none", then the framespacing attribute is ignored. The frame and framespacing attributes are not part of the rowlines/columnlines, rowspacing/columnspacing options because having them be so would often require that rowlines and columnlines would need to be fully specified instead of just giving a single value. For example, if a table had five columns and it was desired to have no frame around the table but to have lines between the columns, then columnlines="none solid solid solid solid none" would be necessary. If the frame is separated from the internal lines, only columnlines="solid" is needed.

The equalrows attribute forces the rows all to be the same total height when set to "true". The equalcolumns attribute forces the columns all to be the same width when set to "true".

The displaystyle attribute specifies the value of displaystyle (described under mstyle in Section 3.3.4) within each cell (mtd element) of the table. Setting displaystyle="true" can be useful for tables whose elements are whole mathematical expressions; the default value of "false" is appropriate when the table is part of an expression, for example, when it represents a matrix. In either case, scriptlevel (Section 3.3.4) is not changed for the table cells.

The side attribute specifies what side of a table a label for a table row should should be placed. This attribute is intended to be used for labeled expressions. If "left" or "right" is specified, the label is placed on the left or right side of the table row respectively. The other two attribute values are variations on "left" and "right": if the labeled row fits within the width allowed for the table without the label, but does not fit within the width if the label is included, then the label overlaps the row and is displayed above the row if rowalign for that row is "top"; otherwise the label is displayed below the row.

If there are multiple labels in a table, the alignment of the labels within the virtual column that they form is left-aligned for labels on the left side of the table, and right-aligned for labels on the right side of the table. The alignment can be overridden by specifying columnalignment for a mlabeledtr element.

The `minlabelspacing` attribute specifies the minimum space allowed between a label and the adjacent entry in the row.

3.5.1.3 Examples

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

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

This might be rendered as:

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

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

3.5.2 Row in Table or Matrix (`mtr`)

3.5.2.1 Description

An `mtr` element represents one row in a table or matrix. An `mtr` element is only allowed as a direct sub-expression of an `mtable` element, and specifies that its contents should form one row of the table. Each argument of `mtr` is placed in a different column of the table, starting at the leftmost column in a LTR context or rightmost column in a RTL context.

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

3.5.2.2 Attributes

In addition to the attributes listed below, this element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
rowalign	top | bottom | center | baseline | axis	inherited
columnalign	(left | center | right) +	inherited
groupalign	group-alignment-list-list	inherited

The `rowalign` and `columnalign` attributes allow a specific row to override the alignment specified by the same attributes in the surrounding `mtable` element.

As with `mtable`, if there are more entries than necessary in the value of `columnalign` (i.e. more entries than columns in the row), then the extra entries will be ignored. If there are fewer entries than columns, then the last entry will be repeated as many times as needed.

The `groupalign` attribute is described with the alignment elements, `maligngroup` and `malignmark`, in Section 3.5.5.

3.5.3 Labeled Row in Table or Matrix (`mlabeledtr`)

3.5.3.1 Description

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

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

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

While MathML 2.0 does not specify an algorithm for placing labels, implementors of visual renderers may find the following formatting model useful. To place a label, an implementor might think in terms of creating a larger table, with an extra column on both ends. The `columnwidth` attributes of both these border columns would be set to `"fit"` so that they expand to fill whatever space remains after the inner columns have been laid out. Finally, depending on the values of `side` and `minlabelspacing`, the label is placed in whatever border column is appropriate, possibly shifted down if necessary.

3.5.3.2 Attributes

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

```
<mlabeledtr rowalign='top'>
```

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

3.5.3.3 Equation Numbering

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

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

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

This should be rendered as:

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

3.5.4 Entry in Table or Matrix (mtd)

3.5.4.1 Description

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

The mtd element accepts any number of arguments; if this number is not 1, its contents are treated as a single ‘inferred mrow’ formed from all its arguments, as described in Section 3.1.3.

3.5.4.2 Attributes

Name	values	default
rowspan	positive-integer	1
columnspan	positive-integer	1
rowalign	top | bottom | center | baseline | axis	inherited
columnalign	left | center | right	inherited
groupalign	group-alignment-list	inherited

The `rowspan` and `columnspan` attributes allow a specific matrix element to be treated as if it occupied the number of rows or columns specified. The interpretation of how this larger element affects specifying subsequent rows and columns is meant to correspond with the similar attributes for HTML 4.01 tables.

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`.

The `rowalign` and `columnalign` attributes allow a specific matrix element to override the alignment specified by a surrounding `mtable` or `mtr` element.

The `groupalign` attribute is described with the alignment elements, `maligngroup` and `malignmark`, in Section 3.5.5.

3.5.5 Alignment Markers

3.5.5.1 Description

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

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

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

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

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

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

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

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

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

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

3.5.5.2 Specifying alignment groups

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

The alignment groups start at the locations of invisible `maligngroup` elements, which are rendered with zero width when they occur outside of an alignment scope, but within an alignment scope are rendered with just enough horizontal space to cause the desired alignment of the alignment group that follows them. A simple algorithm by which a MathML application can achieve this is given later. In the example above, each equation would have one `malígngroup` 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* `malígngroup` elements contains *n* alignment groups, with the *i*th group consisting of the elements entirely after the *i*th `malígngroup` element and before the (*i*+1)-th; no element within the table cell's content should occur entirely before its first `malígngroup` 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. 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 `malígngroup` element is allowed within a table cell. The `malígngroup` 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-argument `mtd` element;
- an `mstyle` element;
- an `mphantom` element;
- an `mfenced` element;
- an `maction` element, though only its selected sub-expression is checked;
- a `semantics` element.

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

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

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

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

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

3.5.5.3 Table cells that are not divided into alignment groups

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

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

or, with a shorter heading,

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

3.5.5.4 Specifying alignment points using `malignmark`

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

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

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

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

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

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

3.5.5.5 `malignmark` Attributes

In addition to the attributes listed below, the `malignmark` element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
edge	left | right	left

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

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

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

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

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

3.5.5.6 `maligngroup` Attributes

In addition to the attributes listed below, the `maligngroup` element permits `xml:id`, `xref`, `class` and `style` attributes, as described in Section 2.1.4.

Name	values	default
groupalign	left | center | right | decimalpoint	inherited

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

In the example given at the beginning of this section, there is one column of 2 table cells, with 7 alignment groups in each table cell; thus there are 7 columns of alignment groups, with 2 groups, one above the other, in each column. These columns of alignment groups should be given the 7 `groupalign` values ‘decimalpoint left left decimalpoint left left decimalpoint’, in that order. How to specify this list of values for a table cell or table column as a whole, using attributes on elements surrounding the `maligngroup` element is described later.

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

If `groupalign` is ‘decimalpoint’, the alignment point is the right edge of the last character before the decimal point. The decimal point is the first ‘.’ character (ASCII 0x2e) in the first `mn` element found along the alignment group’s baseline. More precisely, the alignment group is scanned recursively, depth-first, for the first `mn` element, descending into all arguments of each element of the types `mrow` (including inferred `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.

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

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

3.5.5.7 Inheritance of groupalign values

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

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

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

As described in Section 2.1.3, | 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
malign group	group-alignment	inherited from within mtd attribute

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

3.5.5.8 MathML representation of an alignment example

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

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

One way to represent that in MathML is:

```
<mtable groupalign="{decimalpoint left left decimalpoint left left decimalpoint}">
  <mtr>
    <mtd>
      <mrow>
        <mrow>
          <mrow>
            <maligngroup/>
            <mn> 8.44 </mn>
            <mo> &InvisibleTimes; </mo>
            <maligngroup/>
            <mi> x </mi>
          </mrow>
          <maligngroup/>
          <mo> + </mo>
          <mrow>
            <maligngroup/>
            <mn> 55 </mn>
            <mo> &InvisibleTimes; </mo>
            <maligngroup/>
            <mi> y </mi>
          </mrow>
        </mrow>
      <maligngroup/>
      <mo> = </mo>
      <maligngroup/>
      <mn> 0 </mn>
    </mrow>
    </mtd>
  </mtr>
  <mtr>
    <mtd>
      <mrow>
        <mrow>
          <mrow>
            <maligngroup/>
            <mn> 3.1 </mn>
            <mo> &InvisibleTimes; </mo>
            <maligngroup/>
            <mi> x </mi>
          </mrow>
          <maligngroup/>
          <mo> - </mo>
          <mrow>
            <maligngroup/>
            <mn> 0.7 </mn>
            <mo> &InvisibleTimes; </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 application can perform the alignment specified in this section is given here. Since the alignment specification is deterministic (except for the definition of the left and right edges of a character), any correct MathML alignment algorithm will have the same behavior as this one. Each `mtable` column (alignment scope) can be treated independently; the algorithm given here applies to one `mtable` column, and takes into account the alignment elements, the `groupalign` attribute described in this section, and the `columnalign` attribute described under `mtable` (Section 3.5.1).

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

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

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

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

The alignment groups are then positioned in the unique way that places the part of each ith group to the left of its alignment point in a section called L(i), and places the part of each ith group to the right of its alignment point in a section called R(i). This results in the alignment point of each ith group being on the boundary between adjacent sections L(i) and R(i), so that all alignment points of ith groups have the same horizontal position.

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

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

3.5.6 `mcolumn`

`mcolumn` is typically used to layout numbers that are aligned on each digit. This is common in many elementary math notations such as 2D addition and multiplication.

Inside an `mcolumn`, the character inside of the token elements `mi`, `mn`, `mo`, and `mtext` each occupy a column. The width of a column is the maximum of the widths of each character in that column. If a child of `mcolumn` is not one of the token elements listed above, then that element is considered to be a single digit wide. The exceptions to this are `mspace`, `mline`, `mstyle` and `mrow`. `mspace` and `mline` have the amount of space specifed by them and do not participate in the computation of the width of a column. The width rule should be applied (recursively) to the child of `mstyle`. For `mrow`, the width is the sum of the widths of each child. Inside of a `mcolumn`, `mrow` does not perform automatic spacing or linebreaking. 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).

If a child is too small or to large to fit within a column, the `columnalign` attribute controls whether it is left, center, or right aligned.

The width of a `mcolumn` is the sum of the widths of all of the columns; no spacing should be added between columns. The baseline of the `mcolumn` is specifed by the `align` attribute.

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

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

3.5.6.1 Attributes

In addition to the attribute listed below, this element permits `"xml:id"`, `"xref"`, `"class"` and `"style"` attributes, as described in Section 2.1.4.

Name	values	default
justify	left | right	right
columnalign	(left | center | right) +	center
align	(top | bottom | center | baseline | axis) [rownumber]	baseline

The `justify` attribute specifies whether the row is to be left justified or right justified.

The `columnalign` attribute specifies how the entries in each column should be aligned if they are bigger or smaller than the column width. The specification for `columnalign` is the same as `columnalign` in mtable. See Section 3.5.1 for the full specification of the attribute value. If an element is too large to fit within a column, the `columnalign` attribute controls its alignment with respect to that column and any excess overflows into the surrounding columns. This excess does not participate in the column width calculation. In these cases, authors should take care to avoid collisions between column overflows.

The `align` attribute specifies where to align the `mcolumn` with respect to its environment. Its specification is the same as that for `mtable`'s `align` attribute. See Section 3.5.1 for the full specification of the attribute value

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

3.5.6.2 Examples

Issue (to-display-mcolumn): The examples in this section should be images based on real typesetting, not ASCII approximations.

$$
\begin{array}{r}
123 \\
\times 456 \\
\hline
123 \\
246 \\
369 \\
\hline
\end{array}
$$

The MathML for this is:

```
<mcolumn>
  <mn>123</mn>
  <mrow> <mo>&times;</mo> <mn>321</mn> </mrow>
  <mline spacing='12345'/>
  <mn>123</mn>
```

```
  <mrow> <mn>246</mn> <mspace spacing='0'/></mrow>
  <mrow> <mn>369</mn> <mspace spacing='00'/></mrow>
  <mline spacing='36900'/>
</mcolumn>
```

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.

```
<mcolumn>
  <mn>123</mn>
  <mrow> <mn>456</mn> <mo>+</mo> </mrow>
  <mline spacing='456+'/>
  <mn>579</mn>
</mcolumn>
```

Because the default alignment is placed to the right of number, the numbers align properly and none of the rows need to be shifted.

$$\begin{array}{l} 123 \\ 456+ \\ \hline 579 \end{array}$$

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):

Issue (examples-missing-graphic): An image is required here.

Here is how it can be done with `mcolumn`:

```
<mcolumn>
  <mstyle mathsize='71%'> <menclose notation='bottom'> <mn>10</mn> </menclose> </mstyle>
  <mn>5&#x0338;2</mn>
  <mrow> <mo>&minus;</mo> <mn>7</mn> </mrow>
  <mline spacing='45'/>
  <mn>45</mn>
</mcolumn>
```

$$\begin{array}{r} {\scriptstyle \underline{10}} \\ \not{5}2 \\ -7 \\ \hline 45 \end{array}$$

Note that because `menclose` is not one of the listed elements above, it is considered to be a single digit wide so that its use does not make that column wider. If it is too wide, it overflows into the other columns.

Notice also that the combining long solidus (̸) is used rather than `menclose`. This is done because it logically keeps the number 57 as a single number in an `mn`. An `menclose` can be used, but the use of combining characters is recommended for the above reason. U+20E5 can be used for a reverse strike out, along with other overlay characters. If more than one character should be included in the cross out (as opposed to multiple characters that are individually crossed out), then `menclose` should be used.

Carries and borrows are typically reduced in size, but the computation of their size is based on the number of digits as specified above, and the digit size is taken as the size of a digit in effect at the `mcolumn`. If there is more than one carry, it may be more convenient to wrap all of the carries in a single `mstyle` as shown below:

```
<mcolumn>
  <mstyle mathsize='71%'> <mn>1</mn> <mn>1</mn> <mspace spacing='0'/></mstyle>
```

```
  <mn>987</mn>
  <mrow> <mo>+</mo> <mn>456</mn> </mrow>
  <mline spacing='+1443'/>
  <mn>1443</mn>
</mcolumn>
```

$$\begin{array}{r} {\scriptstyle 1\,1} \\ 987 \\ +456 \\ \hline 1443 \end{array}$$

Here is a bigger example that illustrates using various values besides digits as the "spacing" attribute's value.

$$\begin{array}{r} {\scriptstyle 1\,1} \\ {\scriptstyle 1\,1} \\ 1,234 \\ \times 4,321 \\ \hline {\scriptstyle 1\ \ 1\,1\,1\ \ 1} \\ 1,234 \\ 24,68 \\ 370,2 \\ 4,936 \\ \hline 5,332,114 \end{array}$$

This example has multiple rows of carries. It also (somewhat artificially) includes ","s as digit separators. The encoding includes these separators in the spacing attribute value, along non-ASCII values.

```
<mcolumn>
  <mstyle mathsize='71%'>
     <mn>1</mn>
     <mn>1</mn>
    <mspace spacing='0'/>
  </mstyle>
  <mstyle mathsize='71%'>
     <mn>1</mn>
     <mn>1</mn>
    <mspace spacing='0'/>
  </mstyle>
  <mrow>
    <mo rspace='thinmathspace'>&times;</mo>
    <mn>4,321</mn>
  </mrow>
  <mline spacing='&times; 0,000'/>
  <mstyle mathsize='71%'>
     <mn>1</mn>
     <mspace spacing=','/>
     <mn>1</mn>
     <mn>1</mn>
     <mn>1</mn>
     <mspace spacing=','/>
     <mn>1</mn>
     <mspace spacing='00'/>
  </mstyle>
```

```
  <mn>1,234</mn>
  <mrow><mn>24,68</mn><mspace spacing='0'/></mrow>
  <mrow><mn>370,2</mn><mspace spacing='00'/></mrow>
  <mrow><mn>4,936</mn><mspace spacing=',000'/></mrow>
  <mline spacing='5,332,114'/>
  <mn>5,332,114</mn>
</mcolumn>
```

3.6 Enlivening Expressions

3.6.1 Bind Action to Sub-Expression (maction)

Issue (): There is concensus that maction should be deprecated or restricted in some way. There is also consensus that in any event, all attribute values and their behavior should be fully specified (in contrast to the present text.) Note that maction is currently used for linking, so the fate of maction is tied to producing a satisfactory substitute. There is also a dependency on the decision on how to handle foreign markup within MathML. MathQTI has a requirement for form elements that appear in typeset equations, e.g. an input field for an exponent, which could be satisfied by either maction or XForms.

There are many ways in which it might be desirable to make mathematical content active. Adding a link to a MathML sub-expression is one basic kind of interactivity. See Section 7.3.1. However, many other kinds of interactivity cannot be easily accommodated by generic linking mechanisms. For example, in lengthy mathematical expressions, the ability to 'fold' expressions might be provided, i.e. a renderer might allow a reader to toggle between an ellipsis and a much longer expression that it represents.

To provide a mechanism for binding actions to expressions, MathML provides the maction element. This element accepts any number of sub-expressions as arguments.

3.6.1.1 Attributes

In addition to the attributes listed below, this element permits xml:id, xref, class and style attributes, as described in Section 2.1.4.

Name	values	default
actiontype	(described below)	(required attribute, no default value)
selection	positive-integer	1

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.

Since a MathML application is not required to recognize any particular actiontypes, an application can be in MathML conformance just by implementing the above-described default behavior.

The selection attribute is provided for those actiontypes that permit someone viewing a document to select one of several sub-expressions for viewing. Its value should be a positive integer that indicates one of the sub-expressions of the maction element, numbered from 1 to the number of children of the element. When this is the case, the sub-expression so indicated is defined to be the 'selected sub-expression' of the maction element; otherwise the 'selected sub-expression' does not exist, which is an error. When the selection attribute is not specified (including for actiontypes for which it makes no sense), its default value is 1, so the selected sub-expression will be the first sub-expression.

Furthermore, as described in Section 2.5.2, if a MathML application responds to a user command to copy a MathML sub-expression to the environment's 'clipboard', any maction elements present in what is copied should be given selection attributes that correspond to their selection state in the MathML rendering at the time of the copy command.

A suggested list of `actiontypes` and their associated actions is given below. Keep in mind, however, that this list is mainly for illustration, and recognized values and behaviors will vary from application to application.

<maction actiontype="toggle" selection="positive-integer" > (first expression) (second expression)... </maction>
For this action type, a renderer would alternately display the given expressions, cycling through them when a reader clicked on the active expression, starting with the selected expression and updating the `selection` attribute value as described above. 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>
In this case, the renderer would display the expression in context on the screen. When a reader clicked on the expression or moved the mouse over it, the renderer would send a rendering of the message to the browser statusline. Since most browsers in the foreseeable future are likely to be limited to displaying text on their statusline, authors would presumably use plain text in an `mtext` element for the message 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>
Here the renderer would also display the expression in context on the screen. When the mouse 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. These message boxes are also sometimes called ‘balloon help’ boxes. Presumably authors would use plain text in an `mtext` element for the message 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="highlight" my:color="red" my:background="yellow"> expression </maction>
In this case, a renderer might highlight the enclosed expression on a ‘mouse-over’ event. In the example given above, non-standard attributes from another namespace are being used to pass additional information to renderers that support them, without violating the MathML DTD (see Section 2.3.3). The `my:color` attribute changes the color of the characters in the presentation, while the `my:background` attribute changes the color of the background behind the characters.

3.7 Elementary Math

Mathematics used in the lower grades tends to be tabular in nature. However, the specific notation used varies among countries much more than it does for higher level math. Furthermore, elementary math often presents examples in some intermediate step and MathML must be able to capture these intermediate or intentionally missing partial forms.

The elements needed for elementary math are presented elsewhere in this chapter. In this section, examples are given of how these elements can be used to display various notations used for elementary mathematics.

3.7.1 Addition, Subtraction, and Multiplication

Two-dimensional addition, subtraction, and multiplication typically involve numbers, carrries/borrows, lines, and the sign of the operation. These are supported by MathML inside of `mcolumn`. Lines are drawn using `mline` and alignment is achieved via padding each line with `mspace`.

Issue (ldiv-img): Should move some of the examples from Section 3.5.6 here.

3.7.2 Long Division

The notation used for long division varies considerably among countries. Many notations share the common characteristics of aligning intermediate results and drawing lines for the operands to be subtracted. The line that is drawn various in length depending upon the notation.

The position of the divisor varies, as does the location of the quotient, remainder, and intermediate terms.

Issue (ldiv-img2): Image of two-dimensional long division needed. Need several images showing different styles of long division.

Issue (ldiv-example): Need to include MathML for the following examples.

The US method for long division is

$$
\begin{array}{rr}
 & 435.\bar{3} \\
3 & \overline{)1306} \\
 & \underline{12} \\
 & 10 \\
 & \underline{9} \\
 & 16 \\
 & \underline{15} \\
 & 1.0 \\
 & \underline{9} \\
 & 1
\end{array}
$$

The MathML for this is:

```
<mtable>
  <mtr>
    <mtd></mtd>
    <mtd columnalign="right"><mn>435.3&#x0305;</mn></mtd>
  </mtr>
  <mtr>
    <mtd columnalign="left"><mn>3</mn></mtd>
    <mtd columnalign="left">
      <mcolumn align="left">
        <menclose notation="longdiv"><mn>1306</mn></menclose>
        <mn>12</mn>
  <mline spacing="00"/>
        <mrow><mspace spacing="0"/><mn>10</mn></mrow>
        <mrow><mspace spacing="00"/><mn>9</mn></mrow>
        <mrow><mspace spacing="0"/><mline spacing="00"/></mrow>
        <mrow><mspace spacing="00"/><mn>16</mn></mrow>
        <mrow><mspace spacing="00"/><mn>15</mn></mrow>
        <mrow><mspace spacing="00"/><mline spacing="00"/></mrow>
        <mrow><mspace spacing="000"/><mn>1.0</mn></mrow>
        <mrow><mspace spacing="0000."/><mn>9</mn></mrow>
        <mrow><mspace spacing="000"/><mline spacing="0.0"/></mrow>
        <mrow><mspace spacing="0000."/><mn>1</mn></mrow>
      </mcolumn>
    </mtd>
  </mtr>
</mtable>
```

The French method for long division is

$$
\begin{array}{r|l}
1306 & 3 \\ \cline{2-2}
\underline{12} & 435,\overline{3} \\
10 & \\
\underline{9} & \\
16 & \\
\underline{15} & \\
1,0 & \\
\underline{9} & \\
1 &
\end{array}
$$

The MathML for this is:

```
<mtable>
  <mtr>
    <mtd columnalign="left">
      <menclose notation="left">
        <mcolumn justify="left">
          <mn>1306</mn>
          <mn>12</mn>
          <mline spacing="00"/>
          <mrow><mspace spacing="0"/><mn>10</mn></mrow>
          <mrow><mspace spacing="00"/><mn>9</mn></mrow>
          <mrow><mspace spacing="0"/><mline spacing="00"/></mrow>
          <mrow><mspace spacing="00"/><mn>16</mn></mrow>
          <mrow><mspace spacing="00"/><mn>15</mn></mrow>
          <mrow><mspace spacing="00"/><mline spacing="00"/></mrow>
          <mrow><mspace spacing="000"/><mn>1,0</mn></mrow>
          <mrow><mspace spacing="0000,"/><mn>9</mn></mrow>
          <mrow><mspace spacing="000"/><mline spacing="0,0"/></mrow>
          <mrow><mspace spacing="0000,"/><mn>1</mn></mrow>
        </mcolumn>
      </menclose>
    </mtd>
    <mtd columnalign="left">
      <mcolumn justify="left">
        <mn>3</mn>
        <mline spacing="000,0"/>
        <mn>435,3&#x0305;</mn>
      </mcolumn>
    </mtd>
  </mtr>
</mtable>
```

3.7.3 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:

$$0.33333\overline{3}$$

$0.\overline{142857}$

$0.\underline{142857}$

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

The MathML for these involves using `mover`, `munder`, and `mline`. The MathML for the preceeding examples above is given below.

```
<mover align="right">
  <mn> 0.3333 </mn>
  <mline spacing="3"/>
</mover>

<mover align="right">
  <mn> 0.142857 </mn>
  <mline spacing="142857"/>
</mover>

<munder align="right">
  <mn> 0.142857 </mn>
  <mline spacing="142857"/>
</munder >

<mover align="right" diff="add">
  <mn> 0.142857 </mn>
  <mrow> <mo>.</mo> <mspace spacing="4285"/> <mo>.</mo> </mrow>
</mover>
```

3.8 Semantics and Presentation

MathML uses the `semantics` element to allow specifying semantic annotations to presentation MathML elements; these can be content MathML or other notations. As such, `semantics` should be considered part of both presentation MathML and content MathML. All MathML processors should process the `semantics` element, even if they only process one of those subsets.

In semantic annotations a presentation MathML expression is typically the first child of the `semantics` element. However, it can also be given inside of an `annotation-xml` element inside the `semantics` element. If it is part of an `annotation-xml` element, then `encoding="MathML-presentation"` must be used and presentation MathML processors should use this value for the presentation.

See Section 5.2 for more details about the `semantics` and `annotation-xml` elements.

Chapter 4

Content Markup

4.1 Introduction

In MathML3, content markup is divided into two subsets '`Strict`'- and '`Pragmatic`' Content MathML. The first subset uses a minimal set of elements representing the meaning of a mathematical expression in a uniform structure, while the second one tries to strike a pragmatic balance between verbosity and formality. Both forms of content expressions are legitimate and have their role in representing mathematics. Strict Content MathML is canonical in a sense and simplifies the implementation of content MathML processors and the comparison of content expressions and Pragmatic Content MathML is much simpler and more intuitive for humans to understand, read, and write.

Strict Content MathML3 expressions can directly be given a formal semantics in terms of 'OpenMath Objects' [OpenMath2004], and we interpret Pragmatic Content MathML3 expressions by specifying equivalent Strict variants, so that they inherit their semantics.

4.2 Strict Content MathML

4.2.1 The structure of MathML3 Content Expressions

MathML content encoding is based on the concept of an expression tree built up from

- basic expressions, i.e. Numbers, Symbols, and Identifiers
- derived expressions, i.e. function applications and binding expressions, and
- attributions
- error markup

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 generally represent some kind of function application or other mathematical construction that builds up a compound object. 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 terminal nodes underneath the internal node.

This section provides the basic XML Encoding of content MathML expression trees. General usage and the mechanism used to associate mathematical meaning with symbols are provided here. [mathml3cds] provides a complete listing of the specific Content MathML symbols defined by this specification along with full reference information including attributes, syntax, and examples. It also describes the intended semantics of those symbols and suggests default renderings. The rules for using presentation markup within content markup are explained in Section 5.4.2.

4.2.2 Encoding OpenMath Objects

Strict Content MathML is designed to be and XML encoding of OpenMath Objects (see [OpenMath2004]), which constitute the semantics of strict content MathML expressions. The table below gives an element-by-element correspondence between the OpenMath XML encoding of OpenMath objects and strict content MathML.

strict Content MathML	OpenMath
`cn`	`OMI`, `OMF`
`csymbol`	`OMS`
`ci`	`OMV`
`apply`	`OMA`
`bind`	`OMBIND`
`bvar`	`OMBVAR`
`condition`	`OMC` (proposed)
`share`	`OMR`
`semantics`	`OMATTR`, `OMATP`
`annotation`, `annotation-xml`	`OMFOREIGN`
`error`	`OME`

4.2.3 Numbers

The `cn` element is the MathML element used to represent numbers. The supported types of numbers include integers, real numbers, double precision floating point numbers, rational numbers and complex numbers. Where it makes sense, the base in which the number is written can be specified. For most numeric values, the content of a `cn` element should be either `PCDATA` or other `cn` elements.

The permissible attributes on the `cn` are:

Name	Values	Default
`type`	`"integer"` | `"real"` | `"double"` | `"e-notation,"` | `"rational"` | `"complex-cartesian"` | `"complex-polar"`	real
`base`	number	10

The attribute `type` is used to specify the kind of number being represented. The pre-defined values are given in the table above. Unless otherwise specified, the default `"real"` is used.

The attribute `base` 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 `PCDATA` 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"`.

Each data type implies that the content be of a certain form, as detailed below.

integer An integer is represented by an optional sign followed by a string of one or more ‘digits’. How a ‘digit’ is interpreted depends on the `base` attribute. If `base` is present, it specifies the base for the digit encoding, and it specifies it base 10. Thus `base=’16’` specifies a hexadecimal encoding. When `base` > 10, letters are used in alphabetical order as digits. For example,

```
<cn base="16">7FE0</cn>
```

encodes the number written as 32736 in base ten. When `base` > 36, some integers cannot be represented using numbers and letters alone and it is up to the application what additional characters (if any) may be used for digits. For example,

```
<cn base="1000">10F</cn>
```

represents the number written in base 10 as 1,000,015. However, the number written in base 10 as 1,000,037 cannot be represented using letters and numbers alone when `base` is 1000.

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. If a different `base` is specified, then the digits are interpreted as being digits computed to that base (in the same was as described for type `"integer"`).

double This type is used to mark up those double-precision floating point numbers that can be represented in the IEEE 754 standard. 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 content of a `cn` element may be `PCDATA` (representing numeric values as described below), a `infinity` symbol (representing positive real infinity), a `minfinity` symbol (representing negative real infinity) or a `notanumber` element. **Editor's note:**MikoStephen is postulating an `mininfinity` symbol here, but we do not have one yeta. If the content is `PCDATA`, it is interpreted as a real number in scientific notation. The number then has one or two parts, a significand and possibly an exponent. The significand has the format of a base 10 real number, as described above. The exponent (if present) has the format of a base 10 integer as described above. If the exponent is not present, it is taken to have the value 0. The value of the number is then that of the significand times ten to the power of the exponent. A special case of `PCDATA` content is recognized. If a number of the above form has a negative sign and all digits of the significand are zero, then it is taken to be a negative zero in the sense of the IEEE 754 standard.

e-notation This type is deprecated. It is recommended to use `double` or `real` instead.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, 12.3`<sep/>`5 represents 12.3 times 10^5. The default presentation of this example is 12.3e5.

rational A rational number is given as two integers giving the numerator and denominator of a quotient. These should themselves be given as nested `cn` elements.For backward compatibility, deprecated usage allows the two integers to be given as `PCDATA` separated by `<sep/>`. If a `base` is present in this deprecated use, it specifies the base used for the digit encoding of both integers.

complex-cartesian A complex cartesian number is given as two numbers giving the real and imaginary parts. These should themselves be given as nested `cn` elements. As for rational numbers, the deprecated use of `<sep/>` is also allowed.

complex-polar A complex polar number is given as two numbers giving the magnitude and angle. These should themselves be given as nested `cn` elements. As for rational numbers, the deprecated use of `<sep/>` is also allowed.

constant This type was deprecated in MathML 2.0 and is now no longer supported. The number constants `exponentiale`, `imaginaryi`, `true`, `false`, `notanumber`, `pi`, `eulergamma`, and `infinity` should be used instead.

4.2.4 Symbols and Identifiers

The notion of constructing a general expression tree is essentially that of applying an operator to sub-objects. For example, the sum '$x+y$' can be thought of as an application of the addition operator to two arguments x and y. And the expression 'cos(π)' as the application of the cosine function to the number π.

In Content MathML, elements are used for operators and functions to capture the crucial semantic distinction between the function itself and the expression resulting from applying that function to zero or more arguments. This is addressed by making the functions self-contained objects with their own properties and providing an explicit `apply` construct corresponding to function application. We will consider the `apply` construct in the next section.

In a sum expression '$x+y$' above, x and y typically taken to be '`variables`', since they have properties, but no fixed value, whereas the addition function is a '`constant`' or '`symbol`' as it denotes a specific function, which is defined somewhere externally. (Note that 'symbol' is used here in the abstract sense and has no connection with any presentation of the construct on screen or paper).

4.2.4.1 Content Identifiers

Strict Content MathML3 uses the `ci` element (for 'content identifier') to construct a variable, or an identifier that is not a symbol. Its `PCDATA` content is interpreted as a name that identifies it. Two variables are considered equal, iff their names are in the respective scope (see Section 4.2.6 for a discussion). A `type` attribute indicates the type of object the symbol represents. Typically, `ci` represents a real scalar, but no default is specified.

Name	values	default
type	string	unspecified

4.2.4.2 Content Symbols

Due to the nature of mathematics the meaning of the mathematical expressions must be extensible. The key to extensibility is the ability of the user to define new functions and other symbols to expand the terrain of mathematical discourse. The `csymbol` element is used represent a 'symbol' in much the same way that `ci` is used to construct a variable. The difference is that `csymbol` should refer to some mathematically defined concept with an external definition referenced via the content dictionary attributes, whereas `ci` is used for identifiers that are essentially 'local' to the MathML expression.

In MathML3, external definitions are grouped in *Content Dictionaries* (structured documents for the definition of mathematical concepts; see [OpenMath2004] and [mathml3cds]).

We need three bits of information to fully identify a symbol: a *symbol name*, a *Content Dictionary name*, and (optionally) a *Content Dictionary base URI*, which we encode in the textual content (which is the symbol name) and two attributes of the `csymbol` element: `cd` and `cdbase`. The Content Dictionary is the location of the declaration of the symbol, consisting of a name and, optionally, a unique prefix called a *cdbase* which is used to disambiguate multiple Content Dictionaries of the same name. There are multiple encodings for content dictionaries, this referencing scheme does not distinguish between them. If a symbol does not have an explicit `cdbase` attribute, then it inherits its `cdbase` from the first ancestor in the XML tree with one, should such an element exist. In this document we have tended to omit the `cdbase` for brevity.

Name	values	default
cdbase	URI	inherited
cd	URI	required

Editor's note: MiKoneed to fix the default URI here

Issue (): We might make the `cd` attribute optional? Then that would refer to the current CD if we are in one, or we could make `cd` inherit like `cdbase`. That would save bandwidth

There are other properties of the symbol that are not explicit in these fields but whose values may be obtained by inspecting the Content Dictionary specified. These include the symbol definition, formal properties and examples and, optionally, a *Role* which is a restriction on where the symbol may appear in a MathML expression tree. The possible roles are described in Section 8.5.

```
<csymbol cdbase="http://www.example.com" cd="VectorCalculus">Christoffel</csymbol>
```

For backwards compatibility with MathML2 and to facilitate the use of MathML within a URI-based framework (such as RDF [rdf] or OWL [owl]), the `csymbol` content together with the values of the `cd` and `cdbase` attributes can be combined in the `definitionURL` attribute: we provide the following scheme for constructing a canonical URI for an MathML Symbol, which can be given in the `definitionURL` attribute.

$\texttt{\{URI = \}}cdbase-value\texttt{\{ + '/' + \}}cd-value\texttt{\{ + '\#' + \}}content$

In the case of the Christoffel symbol above this would be the URL

```
http://www.example.com/VectorCalculus#Christoffel
```

For backwards compatibility with MathML2, we do not require that the `definitionURL` point to a content dictionary. But if the URL in this attribute is of the form above, it will be interpreted as the canonical URL of a MathML3 symbol. So the representation above would be equivalent to the one below:

```
<csymbol definitionURL="http://www.example.com/VectorCalculus">Christoffel</csymbol>
```

Issue (): We still have to fix this. Maybe it should correspond to the final resting place for CDs.

Issue (): The URI encoding of the triplet we propose here does not work (not yet for MathMLCDs and not at all for OpenMath2 CDs). The URI reference proposed uses a bare name pointer `#Christoffel` at the end, which points to the element that has and `ID`-type attribute with value `Christoffel`, which is not present in either of these formats. Moreover, it does not scale well with extended CD formats like the OMDoc 1.8 format currently under development

Issue (): What do we want to use for referencing the CD in csymbol? I propose to add `cdbase`, `cd` attributes as in OpenMath to have maximal compatibility. This also enables negotiation over multiple CD encodings.

Resolution: We have decided to add `cdbase` and `cd` and use the `csymbol` content for the symbol name. Using the triplet means that there is an abstract CD for this.

Issue (default): For the inheritance mechanism to be complete, it would make sense to define a default cdbase attribute value, e.g. at the math element. We'd support expressions ignorant of cdbase as they all are thus far. Something such as `http://www.w3.org/Math/CDs/official` ? Moreover the MathML content dictionaries should contain such.

Issue (): What should be the value of the `encoding` attribute. I propose the MIME type. What is the mine-type for MathML content dictionaries?

Resolution: We should drop the`encoding` altogether and let the application deal with the MIME type returned by the CD hosting application. We can use MIME type negotiation to get the right one.

Issue ():do we want to deprecate it for the OM-conformant three-attribute referencing way?

Resolution: We do not deprecate anything, but this is 'Pragmatic Content MathML'quote> now. In particular, we can still use `definitionURL` for situations where we do not want to or cannot point to a Content Dictionary, but somewhere which isn't. This is a slight anomaly in the pragmatic-by-translation approach.

Issue ():do we want to keep a table of MIME types (for the encodings) and and the default extensions to make the mapping work? Is this something the OpenMath Society should do?

Resolution: This is something for the OpenMath Society, not the W3C

4.2.5 Function Application

The most fundamental way of building a compound object in mathematics is by applying a function or an operator to some arguments. MathML supplies an infrastructure to represent this in expression trees, which we will present in this section.

An `apply` element is used to build an expression tree that represents the result of applying a function or operator to its arguments. The 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.

Name	values	default
cdbase	URI	inherited

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* and *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:

```
<apply> op a b [c...] </apply>
```

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>
```

Mathematical expressions involving a mixture of operations result in nested occurrences of `apply`. For example, $a x + b$ would be encoded as

```
<apply><csymbol cd="arith1">plus</csymbol>
  <apply><csymbol cd="arith1">times</csymbol>
    <ci>a</ci>
    <ci>x</ci>
  </apply>
  <ci>b</ci>
</apply>
```

There is no need to introduce parentheses or to resort to operator precedence in order to parse the expression 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 viewed as a single coherent object.

An expression 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>
```

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 to be properly formed.

If the object being applied as a function is not already one of the elements known to be a function (such as `sin` or `plus`) then it is treated as if it were a function.

4.2.6 Bindings and Bound Variables

Some complex mathematical objects are constructed by the use of bound variables. For instance the integration variables in an integral expression is one.

4.2.6.1 Bindings

Such expressions are represented as MathML expression trees using the `bind` element. Its first child is a MathML expression that represents a binding operator (the integral operator in our example). This can be followed by a non-empty list of `bvar` elements for the bound variables, possibly augmented by the qualifier element `condition` (see Section 4.2.7. The last child is the *body* of the binding, it is another content MathML expression.

Name	values	default
cdbase	URI	inherited

4.2.6.2 Bound Variables

The `bvar` element is a special qualifier element that is used to denote the bound variable of a binding expression, e.g. in sums, products, and quantifiers or user defined functions.

Name	values	default
cdbase	URI	inherited

4.2.6.3 Examples

```
<bind>
  <csymbol cd="quant1">forall</csymbol>
  <bvar><ci>x</ci></bvar>
  <apply><csymbol cd="relation1">eq</csymbol>
    <apply><csymbol cd="arith1">minus</csymbol><ci>x</ci><ci>x</ci></apply>
    <cn>0</cn>
  </apply>
</bind>
```

```
<bind>
  <csymbol cd="calculus1">int</csymbol>
  <bvar><ci xml:id="var-x">x</ci></bvar>
  <apply><csymbol cd="arith1">power</csymbol>
    <ci definitionURL="#var-x"><mi>x</mi></ci>
    <cn>7</cn>
  </apply>
</bind>
```

Editor's note: MiKoWe need to say something about alpha-conversion here for OpenMath compatibility.

4.2.7 Qualifiers

The integrals we have seen so far have all been indefinite, i.e. the range of the bound variables range is unspecified. In many situations, we also want to specify range of bound variables, e.g. in definitive integrals. MathML3 provides the optional `condition` element as a general restriction mechanism for binding expressions.

4.2.7.1 Conditions

A `condition` element contains a single child that represents a truth condition. Compound conditions are indicated by applying operators such as `and` in the condition. Consider for instance the following representation of a definite integral.

Name	values	default
cdbase	URI	inherited

4.2.7.2 Examples

```
<bind>
  <int/>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><csymbol cd="sets">in</csymbol>
      <apply><csymbol cd="interval">interval</csymbol>
        <cn>0</cn>
<csymbol cd="nums1">infinity</csymbol>
      </apply>
    </apply>
  </condition>
  <apply><csymbol cd="transc1">sin</csymbol><ci>x</ci></apply>
</bind>
```

Here the `condition` element restricts the bound variables to range over the non-negative integers. A number of common mathematical constructions involve such restrictions, either implicit in conventional notation, such as a bound variable, or thought of as part of the operator rather than an argument, as is the case with the limits of a definite integral.

A typical use of the `condition` qualifier is to define sets by rule, rather than enumeration. The following markup, for instance, encodes the set $x \mid x < 1$:

```
<bind><csymbol cd="sets1">set</csymbol>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><csymbol cd="relations1">lt</csymbol><ci>x</ci><cn>1</cn></apply>
  </condition>
  <ci>x</ci>
</bind>
```

In the context of quantifier operators, this corresponds to the ‘such that’ construct used in mathematical expressions. The next example encodes ‘for all x in N there exist prime numbers p, q such that $p+q = 2x$’.

```
<bind><csymbol cd="quant1">forall</csymbol>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><csymbol cd="sets">in</csymbol>
      <ci>x</ci>
      <csymbol cd="setname1">naturalnumbers</csymbol>
    </apply>
  </condition>
  <bind><csymbol cd="quant1">exists</csymbol>
```

```
      <bvar><ci>p</ci></bvar>
      <bvar><ci>q</ci></bvar>
      <condition>
        <apply><csymbol cd="logic1">and</csymbol>
          <apply><csymbol cd="sets">in</csymbol><ci>p</ci><primes/></apply>
          <apply><csymbol cd="sets">in</csymbol><ci>q</ci><primes/></apply>
        </apply>
      </condition>
      <apply><csymbol cd="relation1">eq</csymbol>
         <apply><csymbol cd="arith1">plus</csymbol><ci>p</ci><ci>q</ci></apply>
         <apply><csymbol cd="arith1">times</csymbol><cn>2</cn><ci>x</ci></apply>
      </apply>
    </bind>
</bind>
```

This use extends to multivariate domains by using extra bound variables and a domain corresponding to a cartesian product as in

```
<bind><csymbol cd="calculus1">int</csymbol>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <condition>
    <apply><csymbol cd="logic1">and</csymbol>
      <apply><csymbol cd="relation1">leq</csymbol><cn>0</cn><ci>x</ci></apply>
      <apply><csymbol cd="relation1">leq</csymbol><ci>x</ci><cn>1</cn></apply>
      <apply><csymbol cd="relation1">leq</csymbol><cn>0</cn><ci>y</ci></apply>
      <apply><csymbol cd="relation1">leq</csymbol><ci>y</ci><cn>1</cn></apply>
    </apply>
  </condition>
  <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>
```

Issue ():the `OMBVAR` allows multiple bound variables, for the MathML, I am not so sure, what do we want?

Resolution: It does not matter for alignment, so we should probably leave things as they are for backwards compatibility

4.2.8 Structure Sharing

To conserve space, MathML3 expression trees can make use of structure sharing

4.2.8.1 The `share` element

This element has an `href` attribute whose value is the value of a URI referencing an `xml:id` attribute of a MathML expression tree. When building the MathML expression tree, the `share` element is replaced by a copy of the MathML expression tree referenced by the `href` attribute. Note that this copy is *structurally equal*, but not identical to the element referenced. The values of the `share` will often be relative URI references, in which case they are resolved using the base URI of the document containing the `share element`.

Name	values	default
href	URI	

Issue (): In order to get parallel markup working, we might want to introduce a sharing element for presentation MathML as well. That would also potentially give us size benefits.

Resolution: The WG decided on the Boston F2F that we do not want sharing in presentation (too complicated with all the inherited elements

For instance, the mathematical object $f(f(f(a,a),f(a,a)),f(a,a),f(a,a))$ can be encoded as either one of the following representations (and some intermediate versions as well).

```
<math>         <math>
  <apply>                         <apply>
    <ci>f</ci>                      <ci>f</ci>
    <apply>                         <apply xml:id="t1">
      <ci>f</ci>                      <ci>f</ci>
      <apply>                         <apply xml:id="t11">
        <ci>f</ci>                      <ci>f</ci>
        <ci>a</ci>                      <ci>a</ci>
        <ci>a</ci>                      <ci>a</ci>
      </apply>                        </apply>
      <apply>                         <share href="#t11"/>
        <ci>f</ci>
        <ci>a</ci>
        <ci>a</ci>
      </apply>
    </apply>                        </apply>
    <apply>                         <share href="#t1"/>
      <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>
</math>                        </math>
```

4.2.8.2 An Acyclicity Constraint

We say that an element dominates all its children and all elements they dominate. An `share` element dominates its target, i.e. the element that carries the `xml:id` attribute pointed to by the `href` attribute. For instance in the representation above the `apply` element with `xml:id="t1"` and also the second `share` dominate the `apply` element with `xml:id="t11"`.

The occurrences of the `share` element must obey the following global *acyclicity constraint*: An element may not dominate itself. For instance the following representation violates this constraint:

```
<apply xml:id="foo">
    <csymbol cd="arith1">plus</csymbol>
    <cn>1</cn>
    <apply>
       <csymbol cd="arith1">plus</csymbol>
       <cn>1</cn>
       <share href="foo"/>
    </apply>
  </apply>
```

Here, the `apply` element with `xml:id="foo"` dominates its third child, which dominates the `share` element, which dominates its target: the element with `xml:id="foo"`. So by transitivity, this element dominates itself, and by the acyclicity constraint, it is not an MathML expression tree. Even though it could be given the interpretation of the continued fraction $\frac{1}{1+\frac{1}{1+\frac{1}{1+\dots}}}$ this would correspond to an infinite tree of applications, which is not admitted by Content MathML

Note that the acyclicity constraints is not restricted to such simple cases, as the following example shows:

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

Here, the `apply` with `xml:id="bar"` dominates its third child, the `share` with `href="baz"`, which dominates its target `apply` with `xml:id="baz"`, which in turn dominates its third child, the `share` with `href="bar"`, this finally dominates its target, the original `apply` element with `xml:id="bar"`. So this pair of representations violates the acyclicity constraint.

4.2.8.3 Structure Sharing and Binding

Note that the `share` element is a *syntactic* referencing mechanism: an `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 for variable capture. Consider for instance

```
<bind xml:id="outer">
  <csymbol cd="fns1">lambda</csymbol>
  <bvar><ci>x</ci></bvar>
  <apply>
    <ci>f</ci>
    <bind xml:id="inner">
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <share xml:id="copy" href="#orig"/>
    </bind>
    <apply xml:id="orig"><ci>g</ci><ci>X</ci></apply>
  </apply>
</bind>
```

it represents the term $\lambda x.f(\lambda x.g(x),g(x))$ which has two sub-terms of the form $g(x)$, one with `xml:id="orig"` (the one explicitly represented) and one with `xml:id="copy"`, represented by the `share` element. In the original, the variable x is bound by the *outer* `bind` element, and in the copy, the variable x is bound by the *inner* `bind` element. We say that the inner `bind` has captured the variable X.

It is well-known that variable capture does not conserve semantics. For instance, we could use α-conversion to rename the inner occurrence of x into, say, y arriving at the (same) object $\lambda x.f(\lambda y.g(y),g(x))$ Using references that capture variables in this way can easily lead to representation errors, and is not recommended.

4.2.8.4 Structure Sharing and `cdbase`

Editor's note: MiKosay something about `cdbase` here.

4.2.9 Attribution via `semantics`

Content elements can be adorned with additional information via the `semantics` element, see Section 5.3 for details. As such, the `semantics` element should be considered part of both presentation MathML and content MathML. MathML3 considers a `semantics` element (strict) content MathML, if and only if its first child is (strict) content MathML. All MathML processors should process the `semantics` element, even if they only process one of those subsets.

Editor's note: MiKoGive an elaborated example from the types note here (or in the primer?), reference Section 8.4

Issue (): The functionality of `semantics` together with `annotation` is very similar to the one given by the OpenMath style `attribution` and `foreign` elements. At least if we make the `definitionURL` attribute mandatory on `annotation`, as we had planned for MathML2(2e), but forgot (the types note depends on this). The Difference then is largely in the way the key is addressed, and what we say about the semantics of attributions (does the order play a role, how about duplicates, interaction with alpha renaming,...); some of this is still not fully solved in OpenMath yet, but on the agenda. We should decide for one of the possibilities and consolidate the rest.

Resolution: We have decided to go only with semantics and upgrade it so that it is openmath-compatible.

4.2.10 In Situ Error Markup

A content error expression is made up of a symbol and a sequence of zero or more MathML expression trees. This object has no direct mathematical meaning. Errors occur as the result of some treatment 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.

Name	values	default
cdbase	URI	inherited

To encode an error caused by a division by zero, we would employ a `aritherror` Content Dictionary with a `DivisionByZero` symbol with role `error` we would use the following expression tree:

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

Note that the error should cover the smallest erroneous subexpression so `cerror` can be a subexpression of a bigger one, e.g.

```
<apply><csymbol cd="relation1">eq</csymbol>
  <cerror>
    <csymbol cd="aritherror">DivisionByZero</csymbol>
    <apply><csymbol cd="arith1">divide</csymbol><ci>x</ci><cn>0</cn></apply>
  </cerror>
  <cn>0</cn>
</apply>
```

If an application wishes to signal that the content MathML expressions it has received is invalid or is not well-formed then the offending data must be encoded as a string. For example:

```
<cerror>
  <csymbol cd="parser">invalid_XML</csymbol>
  <mtext> &lt;apply&gt;&lt;cos&gt; &lt;ci&gt;v&lt;/ci&gt; &lt;/apply&gt; </mtext>
</cerror>
```

Note that the < and > characters have been escaped as is usual in an XML document.

4.3 Pragmatic Content MathML

Strict MathML3 content markup differs from earlier versions of MathML in that it has been regularized and based on the content dictionary model introduced by OpenMath [OpenMath2004]. MathML3 also supports MathML2-like markup as a pragmatic representation that is easier to read and more intuitive for humans. 'Content MathML' without qualification consists of both types of markup together.

In the following we will discuss the general aspects of *pragmatic Content MathML3* and indicate the equivalent strict Content MathML3 expressions. Thus the 'pragmatic content MathML' representations inherit the meaning from their strict counterparts. As pragmatic Content MathML is not as regular as strict Content MathML and the mapping from the former to the latter is not regular either, the particulars will be covered in Section 4.4, where the Content MathML operators are introduced.

Editor's note: MiKoThis part of the specification is still under heavy development and should not be considered as final. In particular, the description of the pragmatic-vs-strict correspondence is still under-defined and should only be considered as an indication of the intended relation. We anticipate that we may have to give normative specification of the relation as a XSLT style sheet that converts pragmatic content MathML expressions to strict content MathML expressions. Such a style sheet is under development at "http://svn.openmath.org/OpenMath3/xsl/cmml2om.xsl (actually it transforms pragmatic content MathML to OpenMath, but this is equivalent, and can be transformed to strict content MathML via "http://svn.openmath.org/OpenMath3/xsl/om2mml.xsl.

4.3.1 Operator Elements

Pragmatic content MathML3 provides empty elements for the operators and functions of the K-14 fragment of mathematics. For instance, the empty MathML element `<plus/>` is equivalent to the element

```
<csymbol cdbase="http://w3.org/Math/CD" cd="arith1">plus</csymbol>
```

The set of elements is the same as the ones for MathML2 with few additions. In most cases, the names of the empty operator elements are the same as the symbol names defined in the MathML3 content dictionaries (see Section 4.4 for details). Note that the concepts of 'MathML symbols' (defined in Section 4.2.4) and 'operator elements' are different. In particular not all symbols defined by the MathML3 Content Dictionaries have corresponding operator elements in pragmatic Content MathML.

Issue ():do we want to deprecate the old MathML2 elements in favor of the `csymbol` variant, or is it enough just to state that they are equivalent and leave the choice to the user?

Resolution: We do not deprecate anything, but label it as "pragmatic content MathML"

Issue ():do we introduce new empty elements for the new symbols for which we introduce definitions in the CDs?

Resolution: We introduce new operator elements for the new symbols in the (MathML3) CDs, but no general mechanisms for making new operator elements for other CDs.

Issue (): In MathML2, the meaning of various operator elements could be specialized via various attributes, usually the `type` attribute. Strict Content MathML does not have this possibility

Resolution: We pass these attributes as extra arguments in the `apply` (or `bind` elements), or add new symbols for the non-default case to the respective content dictionaries.

4.3.2 Pragmatic Elements with Attributes

Following MathML2, pragmatic Content MathML3 allows to specialize the meaning of some elements via attributes, usually the `type` attribute. Strict Content MathML does not have this possibility, therefore these attributes are either passed to the symbols as extra arguments in the `apply` or `bind` elements, or MathML3 adds new symbols for the non-default case to the respective content dictionaries. These will normally not have corresponding operator elements (see above).

For instance the closure `interval` element can be given by the `closure` attribute. Thus the pragmatic Content MathML3 expression

```
<apply><interval closure="open-closed"/><cn>0</cn><cn>1</cn></apply>
```

is equivalent to the strict content MathML3 expression

```
<apply><csymbol cd="interval1">interval-oc</csymbol><cn>0</cn><cn>1</cn></apply>
```

The exact relation strict symbols and operator elements is defined in the MathML3 content dictionaries (see Chapter 8) and recapitulated in Section 4.4.

In MathML2, the `definitionURL` attribute could be used to modify the meaning of an element to allow essentially the same notation to be re-used for a discussion taking place in a different mathematical domain. This use of the attribute is deprecated in MathML3, in favor of using a `csymbol` with `cdbase` and `cd` attributes that combine to the same `definitionURL` attribute (see Section 4.2.4.2).

4.3.3 Bindings with `apply`

Pragmatic Content MathML3 allows to use the `apply` element instead of the `bind` element to conserve backwards compatibility with MathML2. The mapping to strict Content MathML applies two general principles here depending on the operator. Where there is a binding operator in the content dictionaries, we use that and only replace the `apply` tag with a `bind` tag. This is the case for instance for the quantifiers: the pragmatic expression

```
<apply>
  <forall/>
  <bvar><ci>x</ci></bvar>
  <apply><geq/><ci>x</ci><ci>x</ci></apply>
</apply>
```

is equivalent to the strict expression.

```
<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>
```

This situation also obtains for the `exists` and `lambda` symbols.

Where binding operators are not available, we just convert the expression with the bound variable into a λ-expression. Usually we have to move any qualifiers into an argument. For instance for sums:

```
<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>
```

is equivalent to the strict expression.

```
<apply>
  <sum/>
  <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><power/><ci>x</ci><ci>i</ci></apply>
  </bind>
</apply>
```

For information about which of the schemes applies to a particular symbol, see the respective instructions in Section 4.4.

4.3.4 Container Markup for Constructor Symbols

Pragmatic Content MathML3 provides an alternative representation for applications of 'constructor' symbols called 'container markup'. Constructor symbols represent operators that construct mathematical operators that construct a mathematical structure from a list of objects. This list can be given by an explicit sequence of arguments or as an expression with a bound variable. In pragmatic Content MathML3, we allow to write the argument list as children to the element instead of having to append them as to the empty operator element as children of an `apply` element.

For instance for the `set` constructor allow to write:

```
<set><ci>a</ci><ci>b</ci><ci>c</ci></set>
```

This is conisdered equivalent to the following strict Content MathML3 expression.

```
<apply><csymbol cd="set1">set</csymbol><ci>a</ci><ci>b</ci><ci>c</ci></apply>
```

But the set constructor can also take a list that is given as an expression with a bound variable in pragmatic Content MathML. Consider for instance the collection of all intervals from 0 to x. Here we do not have a systematic correspondence, since a symbol can only have one role. For the constructor symbols this is the role `application`. Thus the pragmatic Content MathML expression

```
<set>
  <bvar><ci>x</ci></bvar>
  <interval><cn>0</cn><ci>x</ci></interval>
</set>
```

has to be modeled by a

```
<apply>
  <csymbol cd="set1">suchthat</csymbol>
  <bind>
    <csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>x</ci></bvar>
    <apply>
      <csymbol cd="interval1">interval</csymbol>
      <cn>0</cn>
      <ci>x</ci>
    </apply>
  </bind>
</apply>
```

Note that even though we have not made use of this here, the bound variable can be qualified by any of the strict qualifier element `condition` or the pragmatic qualifier elements (see Section 4.3.12, Section 4.3.10 and Section 4.3.11).

Note that container markup is restricted to the MathML2 elements `set`, `interval`, `list`, `matrix`, `matrixrow`, and `vector`. For specifics of the strict-pragmatic correspondence see Section 4.4

Issue (): Do we want to prescribe one of the representations for the DOM? That would make the processing much simpler.

Resolution: We have decided to keep the MathML DOM directly in equivalent to the XML DOM of this, then this becomes a non-issue

4.3.5 Container Markup for Binding Constructors

The `lambda` element allows a kind of container markup for the `lambda` symbol from the `fns1` content dictionary. e.g.

```
<lambda><bvar><ci>x</ci></bvar><ci>x</ci></lambda>
```

but unlike the `set` element, which corresponds to a symbol with role `application`, the role of the `lambda` symbol is `binding`. Therefore the `lambda` element has to have at least one `bvar` child followed by qualifiers (see below), followed by a content MathML element. The strict Content MathML equivalent of the expression above is

```
<bind><csymbol cd="fns1">lambda</csymbol><bvar><ci>x</ci></bvar><ci>x</ci></bind>
```

4.3.6 Container Markup for Applicative Constructors

The `piecewise>`, `piece`, and `otherwise` allow container markup for the constructor symbols of the content dictionary `piece1`. Unlike the cases described above, these do not allow their arguments to be represented as expressions with bound variables, so the strict-pragmatic correspondence is very simple in this case. For instance the pragmatic Content MathML representation of the absolute value function

```
<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>
```

has the strict equivalent

```
<apply>
  <csymbol cd="piece1">piecewise</csymbol>
  <apply>
    <csymbol cd="piece1">piece</csymbol>
    <apply><csymbol cd="arith1">minus</csymbol><ci>x</ci></apply>
    <apply><csymbol cd="arith1">lt</csymbol><ci>x</ci><cn>0</cn></apply>
```

```
  </apply>
  <apply>
    <csymbol cd="piece1">piece</csymbol>
    <cn>0</cn>
    <apply><csymbol cd="arith1">eq</csymbol><ci>x</ci><cn>0</cn></apply>
  </apply>
  <apply>
    <csymbol cd="piece1">piece</csymbol>
    <ci>x</ci>
    <apply><csymbol cd="arith1">gt</csymbol><ci>x</ci><cn>0</cn></apply>
  </apply>
</apply>
```

4.3.7 Symbols and Identifiers With Presentation MathML

In Pragmatic Content MathML, the ci and csymbol elements can contain a general presentation construct (see Section 3.1.6), which is used for rendering (see Section 8.6). For example,

```
<csymbol  cd="ContDiffFuncs">
  <msup><mi>C</mi><mn>2</mn></msup>
</csymbol>
```

encodes an atomic symbol that displays visually as C^2 and that, for purposes of content, is treated as a single symbol representing the space of twice-differentiable continuous functions.

Issue (): What is the strict equivalent for the case of a csymbol with pMathML content, we do not have a good way of determining that either from the pMathML (we could take the element content stripped of elements; I am assuming this in the example below for now) or from the definitionURL. But as David convinced me, this does not work, so we still need to discuss this. In the We also need to keep the use of symbol names as fragment identifiers in mind.

A ci or csymbol element with Presentation MathML content is equivalent to a semantics construction where the first child is a ci whose content is the symbol or identifier name and whose second child is an annotation-xml element with the MathML Presentation. For example the Strict Content MathML equivalent to the example above would be

```
<semantics>
  <csymbol cd="ContDiffFuncs">C2</csymbol>
  <annotation-xml encoding="MathMLP">
    <msup><mi>C</mi><mn>2</mn></msup>
  </annotation-xml>
</semantics>
```

In this situation, the name of the symbol name (which has to be a text string) can be determined from the presentation MathML representation above by stripped off elements. But this is not possible in general . Therefore pragmatic Content MathML allows an additional name attribute on csymbol and ci which allows to specify the name. It is highly advisable to supply name attributes for symbols and identifiers that have presentation MathML content.

Alternatively, the definitionURL attribute can be used to associate a name with with a ci element. See the discussion of bound variables (Section 4.2.6) for a discussion of an important instance of this. For example,

```
<ci name="c1"><msub><mi>c</mi><mn>1</mn></msub></ci>
```

encodes an atomic symbol that displays visually as c_1 which, for purposes of content, is treated as a atomic concept representing a real number.

Instances of the bound variables are normally recognized by comparing the XML information sets of the relevant ci elements after first carrying out XML space normalization. Such identification can be made explicit by placing an xml:id on the ci element in the bvar element and referring to it using the definitionURL attribute on all other instances. An example of this approach is This xml:id based approach is especially helpful when constructions involving bound variables are nested.

It can be necessary to associate additional information with a bound variable one or more instances of it. 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 it and the additional information. Recognition of and 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 xml:id based approach outlined above may still be used.

4.3.8 Elementary MathML Types on Operator and Container Elements

The ci element uses the type attribute to specify the basic type of object that it represents. While any CDATA string is a valid type, the predefined types include "integer", "rational", "real", "complex", "complex-polar", "complex-cartesian", "constant", "function" and more generally, any of the names of the MathML container elements (e.g. vector) 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.

Editor's note: MiKo Give the Strict equivalent here by techniques from the Types Note, but be careful what we eventually do with types.

4.3.9 Domain of Application

In pragmatic Content MathML the domainofapplication element may be used in an apply element to mark up the domain over which a given function is being applied. In contrast to its use as a qualifier in the bind element, the usage in the apply element only marks the argument position. For instance, the integral of a function f over an arbitrary domain C can be represented as

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

in Pragmatic Content MathML to mark the domain for the range argument of the definite integral. This expression is considered equivalent to

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

4.3.10 Domain of Application in Bindings

The domainofapplication was intended to be an alternative to specification of range of bound variables for condition. Generally, a domain of application D can be specified by a condition element requesting that the bound variable is a member of D. For instance, we consider the Pragmatic Content MathML representation

```
<apply><int/>
  <bvar><ci>x</ci></bvar>
  <domainofapplication><ci type="set">D</ci></domainofapplication>
  <apply><ci type="function">f</ci><ci>x</ci></apply>
</apply>
```

as equivalent to the Strict Content MathML representation

```
<apply><csymbol cd="calculus1">int</csymbol>
  <bind><csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>x</ci></bvar>
    <condition>
      <apply><csymbol cd="set1">in</csymbol><ci>x</ci><ci type="set">D</ci></apply>
    </condition>
    <apply><ci type="function">f</ci><ci>x</ci></apply>
  </bind>
</apply>
```

4.3.11 degree

The degree element is a qualifier used by some MathML container elements to specify that, for example, a bound variable is repeated several times, i.e. for the for the ‘degree’ or ‘order’ of an operation. There are a number of basic mathematical constructs that come in families, such as derivatives and moments. Rather than introduce special elements for each of these families, MathML uses a single general construct, the `degree` element for this concept of ‘order’.

```
<bind><diff/>
  <bvar><ci>x</ci><degree><cn>2</cn></degree></bvar>
  <apply><power/><ci>x</ci><cn>5</cn></apply>
</bind>
```

A variable that is to be bound is placed in this container. In a derivative, it indicates which variable with respect to which a function is being differentiated. When the `bvar` element is used to qualify a derivative, the `bvar` element may contain a child `degree` element that specifies the order of the derivative with respect to that variable.

```
<apply>
  <diff/>
  <bvar>
    <ci>x</ci>
    <degree><cn>2</cn></degree>
  </bvar>
  <apply><power/><ci>x</ci><cn>4</cn></apply>
</apply>
```

it is equivalent to

```
<bind>
  <apply><diff/><cn>2</cn></apply>
  <bvar><ci>x</ci></bvar>
  <apply><power/><ci>x</ci><cn>4</cn></apply>
</bind>
```

Editor’s note: MiKospecify a complete list of containers that allow `degree` elements, so far I see `diff`, `partialdiff`, `root`, `moment`.

Note that the degree element is only allowed in the container representation. The strict representation takes the degree as a regular argument as the second child of the `apply` or `bind` element.

Editor’s note: MiKoMake sure that all `MMLdefinitions` of degree-carrying symbols get a paragraph like the one for `root`.

4.3.12 Upper and Lower Limits

The `uplimit` and `lowlimit` elements are pragmatic Content MathML qualifiers that can be used to restrict the range of a bound variable to an interval, e.g. in some integrals and sums. In strict content MathML, the `uplimit`/ `lowlimit` pairs can be expressed via the `interval` symbol from the CD `interval1`. For instance, we consider the Pragmatic Content MathML representation

```
<apply><int/>
  <bvar><ci> x </ci></bvar>
  <lowlimit><ci>a</ci></lowlimit>
  <uplimit><ci>b</ci></uplimit>
  <apply><ci>f</ci><ci>x</ci></apply>
</apply>
```

as equivalent to the following strict representation

```
<apply>
  <csymbol cd="calculus1">defint</csymbol>
  <apply><interval/><ci>a</ci><ci>b</ci></apply>
  <bind>
    <lambda/>
    <bvar><ci>x</ci></bvar>
    <apply><ci>f</ci><ci>x</ci></apply>
  </bind>
</apply>
```

If the `lowlimit` qualifier is missing, it is interpreted as negative infinity, similarly, if `uplimit` is then it is interpreted as positive infinity.

4.3.13 Lifted Associative Commutative Operators

Issue (): Pragmatic Content MathML allows the use of n-ary operators as binding operators with bound variables induced by them. For instance `union` could be used as the equivalent for the TeX `\cup` as well as `\bigcup`. While the relation between the nary and the set-based operators is deterministic, i.e. the induced big operators are fully determined by them, the concepts are quite different in nature (different notational conventions, different types, different occurrence schemata. I therefore propose to extend the MathML K-14 CDs with symbols big operators, much like we already have `sum` as the big operator for for the n-ary `plus` symbol, and `prod` for `times`. For the new symbols, I propose the naming convention of capitalizing the big operators (as an alternative, we could follow TeX and pre-pend a `big`). For example we could have `Union` as a big operator for `union`

Resolution: We have decided not to have a general rule for this correspondence, but to define it on a case-by-case basis to be specified in the CDs. Most cases will be dealt with by making them OpenMath compatible, i.e. by introducing a lambda.

Pragmatic Content MathML allows to use a associative operators to be ‘lifted’ to ‘big operators’, for instance the n-ary minimum operator to the minimum operator over sets, as the minimum of squares in this expression:

```
<apply>
  <min/>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><in/><ci>x</ci><interval><cn>-4</cn><cn>4</cn></interval></apply>
  </condition>
  <apply><power/><ci>x</ci><cn>2</cn></apply>
</apply>
```

While the relation between the nary and the set-based operators is deterministic, i.e. the induced big operators are fully determined by them, the concepts are quite different in nature (different notational conventions, different types, different occurrence schemata). Therefore the MathML3 content dictionaries provide explicit symbols for the 'big operators', much like MathML2 did with sum as the big operator for for the n-ary plus symbol, and prod for times. Concretely, these are big_union, big_intersect, big_max, big_min, big_gcd, big_lcm, big_or, big_and, and big_xor.

Editor's note: MiKoactually, there are more, e.g. cartesianproduct; make a complete list

With these, we can express all pragmatic Content MathML expressions. For instance, the minimum above can be represented in strict Content MathML as

```
<apply>
  <csymbol cd="set1">suchthat</csymbol>
  <bind>
    <csymbol cd="fns1">lambda</csymbol>
    <bvar><ci>S</ci></bvar>
    <condition>
<apply><csymbol cd="set1">in</csymbol><ci>S</ci><ci>F</ci></apply>
    </condition>
    <apply><csymbol cd="set1">setdiff</csymbol><ci>U</ci><ci>S</ci></apply>
  </bind>
</apply>
```

For the exact meaning of the new symbols, consult the content dictionaries.

Issue (): The large operators can be solved in two ways, in the way described here, by inventing large operators (and David does not like symbol names distinguished only by case; and I agree tend to agree with him). Or by extending the role of roles to allow duplicate roles per symbol, then we could re-use the symbols like we did in MathML2, but then we would have to extend OpenMath for that

Resolution: We have decided to provide big operators in the respective CDs, but they do not have empty operator elements.

4.3.14 Declare (declare)

Editor's note: MiKoThis should maybe be moved into a general section about changes or deprecated elements. Also Stan thinks the text should be improved.

MathML2 provided the declare element that allowed to bind properties like types to symbols and variables and to define abbreviations for structure sharing. This element is deprecated in MathML3. Structure sharing can obtained via the share element (see Section 4.2.8 for details).

4.4 The MathML3 Content Dictionaries and Operators

We will now give an overview over the MathML3 symbols: they are grouped into content dictionaries that broadly reflect the area of mathematics from which they come.

Editor's note: MiKo The list is generated from the MathML3 Content Dictionaries at http://svn.openmath.org/OpenMath3/cd/MathML/. These are the result of merging material from the MathML2 and OpenMath2 content dictionaries, they are under review. We have already done some work on the arith1 CD to show how the extracted material will look in the MathML3 specification. The presentation of the other CDs will improve with the content dictionary editorial process.

4.4.1 arith1

This CD defines symbols for common arithmetic functions.

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml.

4.4.1.1 lcm (<lcm/>)

Description: This n-ary operator is 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.

Example: If the following expression were evaluated at $a = 2$, $b = 4$, and $c = 6$ it would yield 12.

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

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#lcm.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementlcm in pragmatic Content MathML.

The lcm symbol can be used as a binding operator in pragmatic Content MathML. This role is taken over by the big_lcm symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <lcm/>
    <bvar><ci>x</ci></bvar>
    <ci>x</ci>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="arith1">big_lcm</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <ci>x</ci>
    </bind>
  </apply>
```

4.4.1.2 big_lcm

Description: This n-ary operator is used to construct the least common multiple over a set of expressions.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#big_lcm.

4.4.1.3 gcd (<gcd/>)

Description: This is the n-ary operator used to construct an expression which represents the greatest common divisor of its arguments. If no argument is provided, the gcd is 0. If one argument is provided, the gcd is that argument.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#gcd.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementgcd in pragmatic Content MathML.

The gcd symbol can be used as a binding operator in pragmatic Content MathML. This role is taken over by the big_gcd symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <gcd/>
    <bvar><ci>x</ci></bvar>
    <ci>x</ci>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="arith1">big_gcd</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <ci>x</ci>
    </bind>
  </apply>
```

4.4.1.4 big_gcd

Description: This n-ary operator is used to construct the greatest common divisor over a set of expressions.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#big_gcd.

4.4.1.5 plus (<plus/>)

Description: The symbol representing an n-ary commutative function plus. If no operands are provided, the expression represents the additive identity. If one operand, a, is provided the expression evaluates to "a". If two or more operands are provided, the expression represents the (semi) group element corresponding to a left associative binary pairing of the operands. The meaning of mixed operand types not covered by the signatures shown here are left up to the target system.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#plus.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementplus in pragmatic Content MathML.

As an n-ary operation the operands of plus can also be generated by allowing a function or expression vary over a domain of application though the sum element is normally used for that purpose.

4.4.1.6 `unary_minus` (`<unary_minus/>`)

Description: This symbol denotes unary minus, i.e. the additive inverse for an additive group. It constructs the additive inverse of that group element.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#unary_minus`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `unary_minus` in pragmatic Content MathML.

4.4.1.7 `minus` (`<minus/>`)

Description: The symbol representing a binary minus function, the subtraction operator for an additive group. This is equivalent to adding the additive inverse.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#minus`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`minus` in pragmatic Content MathML.

4.4.1.8 `times` (`<times/>`)

Description: The symbol representing an n-ary multiplication function.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#times`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`times` in pragmatic Content MathML.

4.4.1.9 `divide` (`<divide/>`)

Description: This symbol represents a (binary) division function denoting the first argument right-divided by the second, i.e. divide(a,b)=a*inverse(b). It is the inverse of the multiplication function defined by the symbol times in this CD.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#divide`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`divide` in pragmatic Content MathML.

4.4.1.10 `power` (`<power/>`)

Description: This symbol represents a power function. The first argument is raised to the power of the second argument. When the second argument is not an integer, powering is defined in terms of exponentials and logarithms for the complex and real numbers. This operator can represent general powering.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#power`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`power` in pragmatic Content MathML.

4.4.1.11 abs (<abs/>)

Description: A unary operator which represents the absolute value of its argument. The argument should be numerically valued. In the complex case this is often referred to as the modulus.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#abs.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementabs in pragmatic Content MathML.

4.4.1.12 root (<root/>)

Description: A binary operator which represents its first argument "lowered" to its n'th root where n is the second argument. This is the inverse of the operation represented by the power symbol defined in this CD. Care should be taken as to the precise meaning of this operator, in particular which root is represented, however it is here to represent the general notion of taking n'th roots. As inferred by the signature relevant to this symbol, the function represented by this symbol is the single valued function, the specific root returned is the one indicated by the first CMP. Note also that the converse of the second CMP is not valid in general.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#root.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementroot in pragmatic Content MathML.

The root element is used to construct roots. The kind of root to be taken is specified by a 'degree' child, 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.

Note that Pragmatic MathML3 supports a degree element in the container representation. If no degree is present, a default value of 2 is used.

4.4.1.13 sum (<sum/>)

Description: An operator taking two arguments, the first being the range of summation, e.g. an integral interval, the second being the function to be summed. Note that the sum may be over an infinite interval.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#sum.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementsum in pragmatic Content MathML.

In pragmatic Content MathmL, the sum operator may used as the first child of an apply element, which is qualified by providing a domainofapplication, an uplimit, lowlimit pair, condition element. The index for the summation is specified by a bvar element.

If no bound variables are specified then terms of the sum correspond to those produced by evaluating the function that is provided at the points of the domain, while if bound variables are present they are the index of summation and they take on the values of points in the domain. In this case the terms of the sum correspond to the values of the expression that is provided, evaluated at those points. Depending on the structure of the domain, the domain of summation can be abbreviated by using uplimit and lowlimit to specify upper and lower limits for the sum.

A sum in pragmatic Content MathML is turned into strict Content MathML by supplying a lambda binder for the expression to make it into a function. The range of integration is converted to an interval.

The pragmatic Content MathML 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>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <sum/>
    <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><power/><ci>x</ci><ci>i</ci></apply>
    </bind>
  </apply>
```

4.4.1.14 product (<product/>)

Description: An operator taking two arguments, the first being the range of multiplication e.g. an integral interval, the second being the function to be multiplied. Note that the product may be over an infinite interval.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/arith1.xhtml#product.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementproduct in pragmatic Content MathML.

In pragmatic Content MathmL, the product Operator may used as the first child of an apply element, which is qualified by providing a domainofapplication, an uplimit, lowlimit pair, condition element. The index is specified by a bvar element.

If no bound variables are specified then terms of the product correspond to those produced by evaluating the function that is provided at the points of the domain, while if bound variables are present they are the index and they take on the values of points in the domain. In this case the terms of the product correspond to the values of the expression that is provided, evaluated at those points. Depending on the structure of the domain, the domain of multiplication can be abbreviated by using uplimit and lowlimit to specify upper and lower limits for the product.

A product in pragmatic Content MathML is turned into strict Content MathML by supplying a lambda binder for the expression to make it into a function. The range of integration is converted to an interval.

The pragmatic Content MathML expression

```
<apply>
    <product/>
    <bvar><ci>i</ci></bvar>
    <apply><power/><ci>x</ci><ci>i</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <product/>
    <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><power/><ci>x</ci><ci>i</ci></apply>
    </bind>
  </apply>
```

4.4.2 fns1

This CD is intended to be 'compatible' with the corresponding elements in Content MathML. In this CD we give a set of functions concerning functions themselves. Functions can be constructed from expressions via a lambda expression. Also there are basic function functions like compose, etc.

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml.

4.4.2.1 domain (<domain/>)

Description: This symbol denotes the domain of a given function, which is the set of values it is defined over.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml#domain.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementdomain in pragmatic Content MathML.

4.4.2.2 range (<codomain/>)

Description: This symbol denotes the range of a function, that is a set that the function will map to. The single argument should be the function whos range is being queried. It should be noted that this is not necessarily equal to the image, it is merely required to contain the image.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml#range.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementcodomain in pragmatic Content MathML.

4.4.2.3 image (<image/>)

Description: This symbol denotes the image of a given function, which is the set of values the domain of the given function maps to.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml#image.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`image` in pragmatic Content MathML.

4.4.2.4 `identity` (`<ident/>`)

Description: The identity function, it takes one argument and returns the same value.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml#identity`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`ident` in pragmatic Content MathML.

4.4.2.5 `left_inverse`

Description: This symbol is used to describe the left inverse of its argument (a function). This inverse may only be partially defined because the function may not have been surjective. If the function is not surjective the left inverse function is ill-defined without further stipulations. No other assumptions are made on the semantics of this left inverse.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml#left_inverse`.

4.4.2.6 `right_inverse`

Description: This symbol is used to describe the right inverse of its argument (a function). This inverse may only be partially defined because the function may not have been surjective. If the function is not surjective the right inverse function is ill-defined without further stipulations. No other assumptions are made on the semantics of this right inverse.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml#right_inverse`.

4.4.2.7 `inverse` (`<inverse/>`)

Description: This symbol is used to describe the inverse of its argument (a function). This inverse may only be partially defined because the function may not have been surjective. If the function is not surjective the inverse function is ill-defined without further stipulations. No assumptions are made on the semantics of this inverse.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml#inverse`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`inverse` in pragmatic Content MathML.

4.4.2.8 `left_compose` (`<compose/>`)

Description: This symbol represents the function which forms the left-composition of its two (function) arguments.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml#left_compose`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`compose` in pragmatic Content MathML.

4.4.2.9 `lambda` (`<lambda>`)

Description: This symbol is used to represent anonymous functions as lambda expansions. It is used in a binder that takes two further arguments, the first of which is a list of variables, and the second of which is an expression, and it forms the function which is the lambda extraction of the expression.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/fns1.xhtml#lambda`.

This constructor symbol allows container markup with with the element `lambda` in pragmatic Content MathML.

In pragmatic Content MathML, the `condition` and `domainofapplication` elements can be used to restrict the defined function to a specific domain.

4.4.3 linalg1

Operations on Matrices (independent of the matrix representation).

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg1.xhtml`.

4.4.3.1 `vectorproduct` (`<vectorproduct/>`)

Description: This symbol represents the vector product function. It takes two three dimensional vector arguments and returns a three dimensional vector. It is defined as follows: if we write a as [a_1,a_2,a_3] and b as [b_1,b_2,b_3] then the vector product denoted a x b = [a_2b_3 - a_3b_2 , a_3b_1 - a_1b_3 , a_1b_2 - a_2b_1]. Note that the vector product is often referred to as the cross product.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg1.xhtml#vectorproduct`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `vectorproduct` in pragmatic Content MathML.

4.4.3.2 `scalarproduct` (`<scalarproduct/>`)

Description: This symbol represents the scalar product function. It takes two vector arguments and returns a scalar value. The scalar product of two vectors a, b is defined as |a| * |b| * cos(\theta), where \theta is the angle between the two vectors and |.| is a euclidean size function. Note that the scalar product is often referred to as the dot product.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg1.xhtml#scalarproduct`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `scalarproduct` in pragmatic Content MathML.

4.4.3.3 `outerproduct` (`<outerproduct/>`)

Description: This symbol represents the outer product function. It takes two vector arguments and returns a matrix. It is defined as follows: if we write the i,j'th element of the matrix to be returned as m_i,j, then: m_i,j=a_i * b_j where a_i,b_j are the i'th and j'th elements of a, b respectively.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg1.xhtml#outerproduct`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `outerproduct` in pragmatic Content MathML.

4.4.3.4 `transpose` (`<transpose/>`)

Description: This symbol represents a unary function that denotes the transpose of the given matrix or vector

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg1.xhtml#transpose`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`transpose` in pragmatic Content MathML.

4.4.3.5 `determinant` (`<determinant/>`)

Description: This symbol denotes the unary function which returns the determinant of its argument, the argument should be a square matrix.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg1.xhtml#determinant`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `determinant` in pragmatic Content MathML.

4.4.3.6 `vector_selector` (`<selector/>`)

Description: This symbol represents the function which allows individual entries to be selected from a vector, or a matrixrow. It takes two arguments. The first argument is the position in the vector (or matrixrow) of the required entry, the second argument is the vector (or matrixrow) in question. The indexing is one based, i.e. the first element is indexed by one.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg1.xhtml#vector_selector`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`selector` in pragmatic Content MathML.

4.4.3.7 `matrix_selector` (`<matrix_selector/>`)

Description: This symbol represents the function which allows individual entries to be selected from a matrix. It takes three arguments, the first is the index of the row and the second is the index of the column of the required element, the third argument is the matrix in question. The indexing is one based, i.e. the element in the top left hand corner is indexed by (1,1).

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg1.xhtml#matrix_selector`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `matrix_selector` in pragmatic Content MathML.

4.4.4 logic1

This CD holds the basic logic functions.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml`.

4.4.4.1 `equivalent` (`<equivalent/>`)

Description: This symbol is used to show that two boolean expressions are logically equivalent, that is have the same boolean value for any inputs.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#equivalent.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `equivalent` in pragmatic Content MathML.

4.4.4.2 `not` (`<not/>`)

Description: This symbol represents the logical not function which takes one boolean argument, and returns the opposite boolean value.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#not.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`not` in pragmatic Content MathML.

4.4.4.3 `and` (`<and/>`)

Description: This symbol 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 or false otherwise.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#and.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`and` in pragmatic Content MathML.

The `and` operator element can be used as a binding operator in pragmatic Content MathML. This role is taken over by the `big_and` symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <and/>
    <bvar><ci>x</ci></bvar>
    <apply><eq/><ci>x</ci><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="arith1">big_and</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="relation1">eq</csymbol><ci>x</ci><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.4.4 big_and

Description: This n-ary operator is used to construct the conjunction over a set of forumlae.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#big_and.

4.4.4.5 xor (<xor/>)

Description: This symbol represents the logical xor function which is an n-ary function taking boolean arguments and returning a boolean value. It is true if there are an odd number of true arguments or false otherwise.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#xor.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementxor in pragmatic Content MathML.

The xor operator element can be used as a binding operator in pragmatic Content MathML. This role is taken over by the big_xor symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <xor/>
    <bvar><ci>x</ci></bvar>
    <apply><eq/><ci>x</ci><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="arith1">big_xor</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="relation1">eq</csymbol><ci>x</ci><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.4.6 big_xor

Description: This n-ary operator is used to construct the exclusive disjunction over a set of forumlae.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#big_xor.

4.4.4.7 or (<or/>)

Description: This symbol represents the logical or function which is an n-ary function taking boolean arguments and returning a boolean value. It is true if any of the arguments are true or false otherwise.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#or.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementor in pragmatic Content MathML.

The `or` operator element can be used as a binding operator in pragmatic Content MathML. This role is taken over by the `big_or` symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <or/>
    <bvar><ci>x</ci></bvar>
    <apply><eq/><ci>x</ci><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="arith1">big_or</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="relation1">eq</csymbol><ci>x</ci><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.4.8 `big_or`

Description: This n-ary operator is used to construct the disjunction over a set of forumlae.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#big_or.

4.4.4.9 `implies` (`<implies/>`)

Description: This symbol represents the logical implies 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.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#implies.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`implies` in pragmatic Content MathML.

4.4.4.10 `true` (`<true/>`)

Description: This symbol represents the boolean value true.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#true.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`true` in pragmatic Content MathML.

4.4.4.11 `false` (`<false/>`)

Description: This symbol represents the boolean value false.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/logic1.xhtml#false`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`false` in pragmatic Content MathML.

4.4.5 nums1

This CD is intended to be 'compatible' with the MathML view of constructors for numbers (integers to an arbitrary base, rationals) and symbols for some common numerical constants. This CD holds a set of symbols for creating numbers, including some defined constants (i.e. nullary constructors).

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/nums1.xhtml`.

4.4.5.1 based_integer (<based_integer/>)

Description: This symbol represents the constructor function for integers, specifying the base. It takes two arguments, the first is a positive integer to denote the base to which the number is represented, the second argument is a string which contains an optional sign and the digits of the integer, using 0-9a-z (as a consequence of this no radix greater than 35 is supported). Base 16 and base 10 are already covered in the encodings of integers.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/nums1.xhtml#based_integer`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `based_integer` in pragmatic Content MathML.

4.4.5.2 rational (<rational/>)

Description: This symbol represents the constructor function for rational numbers. It takes two arguments, the first is an integer p to denote the numerator and the second a nonzero integer q to denote the denominator of the rational p/q.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/nums1.xhtml#rational`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`rational` in pragmatic Content MathML.

4.4.5.3 infinity (<infinity/>)

Description: A symbol to represent the notion of infinity.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/nums1.xhtml#infinity`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`infinity` in pragmatic Content MathML.

4.4.5.4 e (<exponentiale/>)

Description: This symbol represents the base of the natural logarithm, approximately 2.718. See Abramowitz and Stegun, Handbook of Mathematical Functions, section 4.1.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/nums1.xhtml#e`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `exponentiale` in pragmatic Content MathML.

4.4.5.5 i (<imaginaryi/>)

Description: This symbol represents the square root of -1.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/nums1.xhtml#i`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `imaginaryi` in pragmatic Content MathML.

4.4.5.6 pi (<pi/>)

Description: A symbol to convey the notion of pi, approximately 3.142. The ratio of the circumference of a circle to its diameter.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/nums1.xhtml#pi`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`pi` in pragmatic Content MathML.

4.4.5.7 gamma (<eulergamma/>)

Description: A symbol to convey the notion of the gamma constant as defined in Abramowitz and Stegun, Handbook of Mathematical Functions, section 6.1.3. It is the limit of 1 + 1/2 + 1/3 + ... + 1/m - ln m as m tends to infinity, this is approximately 0.5772 15664.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/nums1.xhtml#gamma`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `eulergamma` in pragmatic Content MathML.

4.4.5.8 NaN (<notanumber/>)

Description: A symbol to convey the notion of not-a-number. The result of an ill-posed floating computation. See IEEE standard for floating point representations.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/nums1.xhtml#NaN`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `notanumber` in pragmatic Content MathML.

4.4.6 relation1

This CD holds the common arithmetic relations. It is intended to be ‘compatible’ with the appropriate MathML elements.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/relation1.xhtml`.

4.4.6.1 `eq` (`<eq/>`)

Description: This symbol represents the binary equality function.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/relation1.xhtml#eq`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`eq` in pragmatic Content MathML.

4.4.6.2 `lt` (`<lt/>`)

Description: This symbol represents the binary less than function which returns true if the first argument is less than the second, it returns false otherwise.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/relation1.xhtml#lt`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`lt` in pragmatic Content MathML.

4.4.6.3 `gt` (`<gt/>`)

Description: This symbol represents the binary greater than function which returns true if the first argument is greater than the second, it returns false otherwise.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/relation1.xhtml#gt`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`gt` in pragmatic Content MathML.

4.4.6.4 `neq` (`<neq/>`)

Description: This symbol represents the binary inequality function.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/relation1.xhtml#neq`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`neq` in pragmatic Content MathML.

4.4.6.5 `leq` (`<leq/>`)

Description: This symbol represents the binary less than or equal to function which returns true if the first argument is less than or equal to the second, it returns false otherwise.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/relation1.xhtml#leq`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`leq` in pragmatic Content MathML.

4.4.6.6 geq (<geq/>)

Description: This symbol represents the binary greater than or equal to function which returns true if the first argument is greater than or equal to the second, it returns false otherwise.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/relation1.xhtml#geq`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`geq` in pragmatic Content MathML.

4.4.6.7 approx (<approx/>)

Description: This symbol is used to denote the approximate equality of its two arguments.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/relation1.xhtml#approx`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`approx` in pragmatic Content MathML.

4.4.7 set1

This CD defines the set functions and constructors for basic set theory. It is intended to be 'compatible' with the corresponding elements in MathML.

The symbols introduced in this content dictionary have operator elements that take a `type` attribute indicates whehter the they pertain to a set or multilset interpretation of the arguments. This attribute takes the values `"normal"` and `"multiset"`, where `"normal"` is the default value. `"multiset"` indicates that repetitions are allowed. The symbols defined in the `set1` content dictionary correspond to the set interpretation.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml`.

4.4.7.1 cartesian_product (<cartesianproduct/>)

Description: This symbol represents an n-ary construction function for constructing the Cartesian product of sets. It takes n set arguments in order to construct their Cartesian product.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#cartesian_product`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `cartesianproduct` in pragmatic Content MathML.

4.4.7.2 emptyset (<emptyset/>)

Description: This symbol is used to represent the empty set, that is the set which contains no members. It takes no parameters.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#emptyset.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`emptyset` in pragmatic Content MathML.

4.4.7.3 map (<map/>)

Description: This symbol represents a mapping function which may be used to construct sets, it takes as arguments a function from X to Y and a set over X in that order. The value that is returned is a set of values in Y. The argument list may be a set or an integer_interval.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#map.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`map` in pragmatic Content MathML.

4.4.7.4 size (<card/>)

Description: This symbol is used to denote the number of elements in a set. It is either a non-negative integer, or an infinite cardinal number. The symbol infinity may be used for an unspecified infinite cardinal.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#size.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`card` in pragmatic Content MathML.

4.4.7.5 suchthat (<suchthat/>)

Description: This symbol represents the suchthat function which may be used to construct sets, it takes two arguments. The first argument should be the set which contains the elements of the set we wish to represent, the second argument should be a predicate, that is a function from the set to the booleans which describes if an element is to be in the set returned.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#suchthat.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`suchthat` in pragmatic Content MathML.

4.4.7.6 set (<set>)

Description: This symbol represents the set construct. It is an n-ary function. The set entries are given explicitly. There is no implied ordering to the elements of a set.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#set.

This constructor symbol allows container markup with with the element `set` in pragmatic Content MathML.

4.4.7.7 intersect (<intersect/>)

Description: This symbol 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.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#intersect.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementintersect in pragmatic Content MathML.

The intersect operator element can be used as a binding operator in pragmatic Content MathML. This role is taken over by the big_intersect symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <intersect/>
    <bvar><ci>x</ci></bvar>
    <apply><interval type="integer"/><cn>0</cn><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="set1">big_intersect</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="interval1">integer_interval</csymbol><cn>0</cn><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.7.8 big_intersect

Description: This n-ary operator is used to construct the intersection over a collection of sets.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#big_intersect.

4.4.7.9 union (<union/>)

Description: This symbol 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.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#union.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementunion in pragmatic Content MathML.

The union operator element can be used as a binding operator in pragmatic Content MathML. This role is taken over by the big_union symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <union/>
    <bvar><ci>x</ci></bvar>
    <apply><interval type="integer"/><cn>0</cn><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="set1">big_union</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="interval1">integer_interval</csymbol><cn>0</cn><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.7.10 `big_union`

Description: This n-ary operator is used to construct the union over a collection of sets.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#big_union`.

4.4.7.11 `setdiff` (`<setdiff/>`)

Description: This symbol is used to denote the set difference of two sets. 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.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#setdiff`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`setdiff` in pragmatic Content MathML.

4.4.7.12 `subset` (`<subset/>`)

Description: This symbol has two (set) arguments. It is used to denote that the first set is a subset of the second.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#subset`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`subset` in pragmatic Content MathML.

4.4.7.13 `in` (`<in/>`)

Description: This symbol has two arguments, an element and a set. It is used to denote that the element is in the given set.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#in`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`in` in pragmatic Content MathML.

4.4.7.14 `notin` *(`<notin/>`)*

Description: This symbol has two arguments, an element and a set. It is used to denote that the element is not in the given set.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#notin`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`notin` in pragmatic Content MathML.

4.4.7.15 `prsubset` *(`<prsubset/>`)*

Description: This symbol has two (set) arguments. It is used to denote that the first set is a proper subset of the second, that is a subset of the second set but not actually equal to it.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#prsubset`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`prsubset` in pragmatic Content MathML.

4.4.7.16 `notsubset` *(`<notsubset/>`)*

Description: This symbol has two (set) arguments. It is used to denote that the first set is not a subset of the second.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#notsubset`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`notsubset` in pragmatic Content MathML.

4.4.7.17 `notprsubset` *(`<notprsubset/>`)*

Description: This symbol has two (set) arguments. It is used to denote that the first set is not a proper subset of the second. A proper subset of a set is a subset of the set but not actually equal to it.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/set1.xhtml#notprsubset`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `notprsubset` in pragmatic Content MathML.

4.4.8 veccalc1

This CD contains symbols to represent functions which are concerned with vector calculus.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/veccalc1.xhtml`.

4.4.8.1 `divergence` *(`<divergence/>`)*

Description: This symbol is used to represent the divergence function. It takes one argument which should be a vector of scalar valued functions, intended to represent a vector valued function and returns a scalar value. It should satisfy the defining relation: divergence(F) = \partial(F_(x_1))/\partial(x_1) + ... + \partial(F_(x_n))/\partial(x_n)

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/veccalc1.xhtml#divergence.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `divergence` in pragmatic Content MathML.

4.4.8.2 `grad` (`<grad/>`)

Description: This symbol is used to represent the grad function. It takes one argument which should be a scalar valued function and returns a vector of functions. It should satisfy the defining relation: grad(F) = (\partial(F)/\partial(x_1), ... ,\partial(F)/partial(x_n))

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/veccalc1.xhtml#grad.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`grad` in pragmatic Content MathML.

4.4.8.3 `curl` (`<curl/>`)

Description: This symbol is used to represent the curl function. 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. It should satisfy the defining relation: curl(F) = i X \partial(F)/\partial(x) + j X \partial(F)/\partial(y) + j X \partial(F)/\partial(Z) where i,j,k are the unit vectors corresponding to the x,y,z axes respectively and the multiplication X is cross multiplication.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/veccalc1.xhtml#curl.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`curl` in pragmatic Content MathML.

4.4.8.4 `Laplacian` (`<laplacian/>`)

Description: This symbol is used to represent the laplacian function. 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. It should satisfy the defining relation: laplacian(F) = \partial^2(F)/\partial(x_1)^2 + ... + \partial^2(F)/\partial(x_n)^2

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/veccalc1.xhtml#Laplacian.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`laplacian` in pragmatic Content MathML.

4.4.9 calculus1

This CD is intended to be compatible with the calculus operations in Content MathML. Integration is just for the univariate case and is either definite or indefinite.

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/calculus1.xhtml.

4.4.9.1 diff (<diff/>)

Description: This symbol is used to express ordinary differentiation of a unary function. The single argument is the unary function.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/calculus1.xhtml#diff.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementdiff in pragmatic Content MathML.

In pragmatic Content MathML the diff operator can be applied to an expression involving a single variable such as $\sin(x)$, or $\cos(x)$. or a polynomial in x. For the expression case the actual variable is designated by a bvar element that is a child of the containing apply element. To translate this usage to strict Content MathML, we add a lambda construction.

The pragmatic Content MathML expression

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

is equivalent to the strict Content MathML expression

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

The bvar element may also contain a degree element, which specifies the order of the derivative to be taken. To achieve this effect in strict Content MathML, we use the nthdiff symbol.

The pragmatic Content MathML expression

```
<apply>
    <diff/>
    <bvar><degree><cn>2</cn></degree><ci>x</ci></bvar>
    <apply><sin/><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <nthdiff/>
    <cn>2</cn>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><sin/><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.9.2 `nthdiff`

Description: This symbol is used to express the nth-iterated ordinary differentiation of a unary function. The first argument is n, and the second the unary function.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/calculus1.xhtml#nthdiff.

4.4.9.3 `partialdiff` (`<partialdiff/>`)

Description: This symbol is used to express partial differentiation of a function of more than one variable. It has two arguments, the first is a list of integers which index the variables of the function, the second is the function.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/calculus1.xhtml#partialdiff.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `partialdiff` in pragmatic Content MathML.

In pragmatic Content MathML, the `partialdiff` operator can be applied to an expression with bound variables 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. In strict Content MathML, the degrees are given as a list in the first argument of the `partialdiff` symbol.

The pragmatic Content MathML 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 equivalent to the strict Content MathML expression

```
<apply>
    <partialdiff/>
    <apply><csymbol cd="list1">list</csymbol><ci>n</ci><ci>m</ci></apply>
    <bind>
      <lambda/>
      <bvar><ci>x</ci></bvar>
      <bvar><ci>y</ci></bvar>
      <apply><sin/><apply><times/><ci>x</ci><ci>y</ci></apply></apply>
    </bind>
  </apply>
```

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. Each `degree` schema used with `partialdiff` is expected to contain a single child schema. For example,

```
<apply>
  <partialdiff/>
  <bvar>
    <degree><cn>2</cn></degree>
    <ci>x</ci>
  </bvar>
  <bvar><ci>y</ci></bvar>
  <bvar><ci>x</ci></bvar>
  <degree><cn>4</cn></degree>
  <ci type="function">f</ci>
</apply>
```

denotes the mixed partial derivative (d^4 / d^2x dy dx) f. In strict Content MathML, the overall degree cannot be given.

Editor's note: MiKowhat do do about this?

4.4.9.4 `int` (`<int/>`)

Description: This symbol is used to represent indefinite integration of unary functions. The argument is the unary function.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/calculus1.xhtml#int`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`int` in pragmatic Content MathML.

4.4.9.5 `defint` (`<defint/>`)

Description: This symbol is used to represent definite integration of unary functions. It takes two arguments; the first being the range (e.g. a set) of integration, and the second the function.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/calculus1.xhtml#defint`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`defint` in pragmatic Content MathML.

4.4.10 integer1

This CD holds a collection of basic integer functions. This CD is intended to be 'compatible' with the corresponding elements in Content MathML.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/integer1.xhtml`.

4.4.10.1 `factorof` (`<factorof/>`)

Description: This is the binary OpenMath operator that is used to indicate the mathematical relationship a "is a factor of" b, where a is the first argument and b is the second. This relationship is true if and only if b mod a = 0.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/integer1.xhtml#factorof`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`factorof` in pragmatic Content MathML.

4.4.10.2 `factorial` (`<factorial/>`)

Description: The symbol to represent a unary factorial function on non-negative integers.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/integer1.xhtml#factorial`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`factorial` in pragmatic Content MathML.

4.4.10.3 `quotient` (`<quotient/>`)

Description: The symbol to represent the integer (binary) division operator. That is, for integers a and b, quotient(a,b) denotes q such that a=b*q+r, with |r| less than |b| and a*r positive.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/integer1.xhtml#quotient`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`quotient` in pragmatic Content MathML.

4.4.10.4 `remainder` (`<rem/>`)

Description: The symbol to represent the integer remainder after (binary) division. For integers a and b, remainder(a,b) denotes r such that a=b*q+r, with |r| less than |b| and a*r positive.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/integer1.xhtml#remainder`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`rem` in pragmatic Content MathML.

4.4.11 linalg2

This CD treats matrices and vectors in a row oriented fashion (using matrixrow's).

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg2.xhtml`.

4.4.11.1 `vector` (`<vector>`)

Description: This symbol represents an n-ary function used to construct (or describe) vectors. Vectors in this CD are considered to be row vectors and must therefore be transposed to be considered as column vectors.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg2.xhtml#vector`.

This constructor symbol allows container markup with with the element `vector` in pragmatic Content MathML.

4.4.11.2 matrixrow (<matrixrow>)

Description: This symbol is an n-ary constructor used to represent rows of matrices. Its arguments should be members of a ring.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/linalg2.xhtml#matrixrow.

This constructor symbol allows container markup with with the element `matrixrow` in pragmatic Content MathML.

4.4.11.3 matrix (<matrix>)

Description: This symbol is an n-ary matrix constructor which requires matrixrow's as arguments. It is used to represent matrices.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/linalg2.xhtml#matrix.

This constructor symbol allows container markup with with the element `matrix` in pragmatic Content MathML.

4.4.12 mathmlkeys

This content dictionary provides some keys for specifying the relations in semantics elements.

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/mathmlkeys.xhtml.

4.4.12.1 equiv

Description: This key specifies that the corresponding value is equivalent to the annotated element in some unspecified way.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/mathmlkeys.xhtml#equiv.

4.4.12.2 contentequiv

Description: This key specifies that the corresponding value is the content MathML equivalent of the annotated element.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/mathmlkeys.xhtml#contentequiv.

4.4.12.3 contentequiv-strict

Description: This key specifies that the corresponding value is the equivalent in strict content MathML of the annotated element.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/mathmlkeys.xhtml#contentequiv-strict.

4.4.13 rounding1

A CD of basic rounding concepts

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/rounding1.xhtml.

4.4.13.1 `ceiling` (`<ceiling/>`)

Description: The round up (to +infinity) operation.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/rounding1.xhtml#ceiling`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`ceiling` in pragmatic Content MathML.

4.4.13.2 `floor` (`<floor/>`)

Description: The round down (to -infinity) operation.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/rounding1.xhtml#floor`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`floor` in pragmatic Content MathML.

4.4.13.3 `trunc` (`<trunc/>`)

Description: The round to zero operation.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/rounding1.xhtml#trunc`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`trunc` in pragmatic Content MathML.

4.4.13.4 `round` (`<round/>`)

Description: The round to nearest operation.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/rounding1.xhtml#round`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`round` in pragmatic Content MathML.

4.4.14 setname1

This CD defines common sets of mathematics

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/setname1.xhtml`.

4.4.14.1 `P` (`<primes/>`)

Description: This symbol represents the set of positive prime numbers.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/setname1.xhtml#P`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`primes` in pragmatic Content MathML.

4.4.14.2 N (<naturalnumbers/>)

Description: This symbol represents the set of natural numbers (including zero).

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/setname1.xhtml#N.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `naturalnumbers` in pragmatic Content MathML.

4.4.14.3 Z (<integers/>)

Description: This symbol represents the set of integers, positive, negative and zero.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/setname1.xhtml#Z.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`integers` in pragmatic Content MathML.

4.4.14.4 rationials (<rationals/>)

Description: This symbol represents the set of rational numbers.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/setname1.xhtml#rationials.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`rationals` in pragmatic Content MathML.

4.4.14.5 R (<reals/>)

Description: This symbol represents the set of real numbers.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/setname1.xhtml#R.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`reals` in pragmatic Content MathML.

4.4.14.6 complexes (<complexes/>)

Description: This symbol represents the set of complex numbers.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/setname1.xhtml#complexes.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`complexes` in pragmatic Content MathML.

4.4.15 complex1

This CD is intended to be ‘compatible’ with the MathML view of operations on and constructors for complex numbers.

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/complex1.xhtml.

4.4.15.1 `complex_cartesian` (`<complex_cartesian/>`)

Description: This symbol represents a constructor function for complex numbers specified as the Cartesian coordinates of the relevant point on the complex plane. It takes two arguments, the first is a number x to denote the real part and the second a number y to denote the imaginary part of the complex number x + i y. (Where i is the square root of -1.)

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/complex1.xhtml#complex_cartesian`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `complex_cartesian` in pragmatic Content MathML.

4.4.15.2 `real` (`<real/>`)

Description: This represents the real part of a complex number

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/complex1.xhtml#real`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`real` in pragmatic Content MathML.

4.4.15.3 `imaginary` (`<imaginary/>`)

Description: This represents the imaginary part of a complex number

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/complex1.xhtml#imaginary`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`imaginary` in pragmatic Content MathML.

4.4.15.4 `complex_polar` (`<complex_polar/>`)

Description: This symbol represents a constructor function for complex numbers specified as the polar coordinates of the relevant point on the complex plane. It takes two arguments, the first is a nonnegative number r to denote the magnitude and the second a number theta (given in radians) to denote the argument of the complex number r e^(i theta). (i and e are defined as in this CD).

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/complex1.xhtml#complex_polar`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `complex_polar` in pragmatic Content MathML.

4.4.15.5 `argument` (`<arg/>`)

Description: This symbol represents the unary function which returns the argument of a complex number, viz. the angle which a straight line drawn from the number to zero makes with the Real line (measured anti-clockwise). The argument to the symbol is the complex number whos argument is being taken.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/complex1.xhtml#argument`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arg` in pragmatic Content MathML.

4.4.15.6 conjugate (<conjugate/>)

Description: A unary operator representing the complex conjugate of its argument.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/complex1.xhtml#conjugate.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementconjugate in pragmatic Content MathML.

4.4.16 interval1

This CD holds symbols which describe both discrete and continuous 1-dimensional intervals (with open/closed end points).

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/interval1.xhtml.

4.4.16.1 integer_interval (<integer_interval/>)

Description: A symbol to denote a discrete 1 dimensional interval from the first argument to the second (inclusive), where the discretisation occurs at unit intervals. The arguments are the start and the end points of the interval in that order.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/interval1.xhtml#integer_interval.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element integer_interval in pragmatic Content MathML.

4.4.16.2 interval (<interval>)

Description: A symbol to denote a continuous 1-dimensional interval without any information about the character of the end points (used in definite integration). The arguments are the start and the end points of the interval in that order.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/interval1.xhtml#interval.

This constructor symbol allows container markup with with the element interval in pragmatic Content MathML.

In pragmatic Content MathML, the interval element takes an attribute closure, which can take on any of the values "open", "closed", "open-closed", or "closed-open", with a default value of "closed". In strict Content MathML, the interval symbol is reserved for an interval whose closure property is not specified. For with closure information, we use the interval_oo, interval_oo, interval_oo, and interval_oo symbols defined below.

The pragmatic Content MathML expression

```
<interval closure="open"/>
```

is equivalent to the strict Content MathML expression

```
<csymbol cd="interval1">interval_oo</csymbol>
```

Moreover, the interval element can be used as a container element in pragmatic Content MathML.

4.4.16.3 `interval_oo` (`<interval closure="open"/>`)

Description: A symbol to denote a continuous 1-dimensional interval with both end points excluded from the interval. The arguments are the start and the end points of the interval in that order.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/interval1.xhtml#interval_oo`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`interval` with attributes `closure="open"` in pragmatic Content MathML.

4.4.16.4 `interval_cc` (`<interval closure="closed"/>`)

Description: A symbol to denote a continuous 1-dimensional interval with both end points included in the interval. The arguments are the start and the end points of the interval in that order.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/interval1.xhtml#interval_cc`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`interval` with attributes `closure="closed"` in pragmatic Content MathML.

4.4.16.5 `interval_oc` (`<interval closure="open-closed"/>`)

Description: A symbol to denote a continuous 1-dimensional interval with the first point excluded from the interval, but the last included. The arguments are the start and the end points of the interval in that order.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/interval1.xhtml#interval_oc`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`interval` with attributes `closure="open-closed"` in pragmatic Content MathML.

4.4.16.6 `interval_co` (`<interval closure="closed-open"/>`)

Description: A symbol to denote a continuous 1-dimensional interval with the first point included in the interval, but the last excluded. The arguments are the start and the end points of the interval in that order.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/interval1.xhtml#interval_co`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`interval` with attributes `closure="closed-open"` in pragmatic Content MathML.

4.4.17 linalg3

This CD treats matrices and vectors in a column oriented fashion (using matrixcolumn's).

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg3.xhtml`.

4.4.17.1 vector (<vector.column>)

Description: This symbol represents an n-ary function used to construct (or describe) vectors. Vectors in this CD are considered to be column vectors, and must therefore be transposed to be considered as row vectors.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg3.xhtml#vector`.

This constructor symbol allows container markup with with the element `vector.column` in pragmatic Content MathML.

4.4.17.2 matrixcolumn (<matrixcolumn/>)

Description: This symbol is an n-ary constructor used to represent columns of matrices. Its arguments should be members of a ring.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg3.xhtml#matrixcolumn`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `matrixcolumn` in pragmatic Content MathML.

4.4.17.3 matrix (<matrix.column/>)

Description: This symbol is an n-ary matrix constructor which requires matrixcolumn's as arguments. It is used to represent matrices.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/linalg3.xhtml#matrix`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `matrix.column` in pragmatic Content MathML.

4.4.18 minmax1

This CD holds the definitions of min and max.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/minmax1.xhtml`.

4.4.18.1 min (<min/>)

Description: This symbol denotes the unary minimum function which takes a set as its argument and returns the minimum element in that set.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/minmax1.xhtml#min`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`min` in pragmatic Content MathML.

The `min` operator element can be used as a binding operator in pragmatic Content MathML. This role is taken over by the `big_min` symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <min/>
    <bvar><ci>x</ci></bvar>
    <apply><power/><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="arith1">big_min</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="arith1">power</csymbol><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.18.2 big_min

Description: This n-ary operator is used to construct the minimum over a set of objects.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/minmax1.xhtml#big_min.

4.4.18.3 max (<max/>)

Description: This symbol denotes the unary maximum function which takes a set as its argument and returns the maximum element in that set.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/minmax1.xhtml#max.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementmax in pragmatic Content MathML.

The max operator element can be used as a binding operator in pragmatic Content MathML. This role is taken over by the big_max symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <max/>
    <bvar><ci>x</ci></bvar>
    <apply><power/><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="arith1">big_max</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="arith1">power</csymbol><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.18.4 big_max

Description: This n-ary operator is used to construct the maximum over a set of objects.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/minmax1.xhtml#big_max.

4.4.19 piece1

This CD is intended to be compatible with the corresponding elements in Content MathML 2. In this CD we give a set of operators for piece-wise defined expressions.

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/piece1.xhtml.

4.4.19.1 piecewise (<piecewise>)

Description: This operator heads an expression that is being defined piecewise. Its arguments are n objects built with the piece constructor, optionally followed by one built with otherwise constructor.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/piece1.xhtml#piecewise.

This constructor symbol allows container markup with with the element `piecewise` in pragmatic Content MathML.

4.4.19.2 piece (<piece>)

Description: This introduces an individual component of a piecewise definition. It has precisely two arguments: the first is the value, and the second is a Boolean (meant to be a predicate) which is true if and only if this piece is to provide the value of the piecewise construct.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/piece1.xhtml#piece.

This constructor symbol allows container markup with with the element `piece` in pragmatic Content MathML.

4.4.19.3 otherwise (<otherwise>)

Description: This symbol introduces the 'default' value of a piecewise construct. If none of the previous piece constructs can provide the value, this will. It has a single child, the value.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/piece1.xhtml#otherwise.

This constructor symbol allows container markup with with the element `otherwise` in pragmatic Content MathML.

4.4.20 error1

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/error1.xhtml.

4.4.20.1 `unhandled_symbol`

Description: This symbol represents the error which is raised when an application reads a symbol which is present in the mentioned content dictionary, but which it has not implemented. When receiving such a symbol, the application should act as if it had received the OpenMath error object constructed from unhandled_symbol and the unhandled symbol as in the example below.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/error1.xhtml#unhandled_symbol`.

4.4.20.2 `unexpected_symbol`

Description: This symbol represents the error which is raised when an application reads a symbol which is not present in the mentioned content dictionary. When receiving such a symbol, the application should act as if it had received the OpenMath error object constructed from unexpected_symbol and the unexpected symbol as in the example below.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/error1.xhtml#unexpected_symbol`.

4.4.20.3 `unsupported_CD`

Description: This symbol represents the error which is raised when an application reads a symbol where the mentioned content dictionary is not present. When receiving such a symbol, the application should act as if it had received the OpenMath error object constructed from unsupported_CD and the symbol from the unsupported Content Dictionary as in the example below.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/error1.xhtml#unsupported_CD`.

4.4.21 limit1

This CD holds the basic notion of the limits of unary functions whilst its variable tend (either from above, below or both sides) to a particular value.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/limit1.xhtml`.

4.4.21.1 `limit` (`<limit/>`)

Description: This symbol is used to denote the limit of a unary function. It takes 3 arguments: the limiting value of the argument, the method of approach (either null, above, below or both_sides) and the function.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/limit1.xhtml#limit`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`limit` in pragmatic Content MathML.

4.4.21.2 `both_sides` (`<both_sides/>`)

Description: This symbol is used within a limit construct to show the limit is being approached from both sides. It takes no arguments.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/limit1.xhtml#both_sides.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element both_sides in pragmatic Content MathML.

4.4.21.3 above (<above/>)

Description: This symbol is used within a limit construct to show the limit is being approached from above. It takes no arguments.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/limit1.xhtml#above.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementabove in pragmatic Content MathML.

4.4.21.4 below (<below/>)

Description: This symbol is used within a limit construct to show the limit is being approached from below. It takes no arguments.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/limit1.xhtml#below.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementbelow in pragmatic Content MathML.

4.4.21.5 null (<null/>)

Description: This symbol is used within a limit construct to avoid specifying the method of approach to the limit. It takes no arguments.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/limit1.xhtml#null.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementnull in pragmatic Content MathML.

4.4.21.6 tendsto (<tendsto/>)

Description: This symbol is used within a limit construct to show the limit is being approached from below. It takes no arguments.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/limit1.xhtml#tendsto.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementtendsto in pragmatic Content MathML.

4.4.22 list1

This CD is intended to be ‘compatible’ with MathML list constructs.

For a full semantic description of the content dictionary see http://svn.openmath.org/OpenMath3/cd/MathML/list1.xhtml.

4.4.22.1 `map`

Description: This symbol represents a mapping function which may be used to construct lists, it takes as arguments a function from X to Y and a list over X in that order. The value that is returned is a list of values in Y. The argument list may be a set or an integer_interval.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/list1.xhtml#map`.

4.4.22.2 `suchthat`

Description: This symbol represents the suchthat function which may be used to construct lists, it takes two arguments. The first argument should be the set which contains the elements of the list, the second argument should be a predicate, that is a function from the set to the booleans which describes if an element is to be in the list returned.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/list1.xhtml#suchthat`.

4.4.22.3 `list` (`<list>`)

Description: This symbol denotes the list construct which is an n-ary function. The list entries must be given explicitly.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/list1.xhtml#list`.

This constructor symbol allows container markup with with the element `list` in pragmatic Content MathML.

4.4.23 multiset1

This CD defines the set functions and constructors for basic multiset theory. It is intended to be ‘compatible’ with the corresponding elements in MathML i.e. set operations acting on sets of type=multiset.

The symbols introduced in this content dictionary have operator elements in pragmatic Content MathML that take a `type` attribute indicates whehter the they pertain to a set or multilset interpretation of the arguments. The symbols defined in the `multiset1` content dictionary correspond to the multiset interpretatio, where repetitions of members are allowed. Therefore the value `"multiset"` is prescribed on these elements.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml`.

4.4.23.1 `size` (`<size type="multiset"/>`)

Description: This symbol is used to denote the number of elements in a multiset. It is either a non-negative integer, or an infinite cardinal number. The symbol infinity may be used for an unspecified infinite cardinal.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#size`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`size` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.2 `cartesian_product` (`<cartesianproduct type="multiset"/>`)

Description: This symbol represents an n-ary construction function for constructing the Cartesian product of multisets. It takes n multiset arguments in order to construct their Cartesian product.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#cartesian_product`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `cartesianproduct` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.3 `emptyset` (`<emptyset type="multiset"/>`)

Description: This symbol is used to represent the empty multiset, that is the multiset which contains no members. It takes no parameters.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#emptyset`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`emptyset` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.4 `multiset` (`<multiset type="multiset">`)

Description: This symbol represents the multiset construct. It is either an n-ary function, in which case the multiset entries are given explicitly, or it works on an elements construct. There is no implied ordering to the elements of a multiset.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#multiset`.

This constructor symbol allows container markup with with the element `multiset` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.5 `intersect` (`<intersect type="multiset"/>`)

Description: This symbol is used to denote the n-ary intersection of multisets. It takes multisets as arguments, and denotes the multiset that contains all the elements that occur in all of them, with multiplicity the minimum of their multiplicities in all multisets.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#intersect`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`intersect` with attributes `type="multiset"` in pragmatic Content MathML.

The `intersect` operator element can be used as a binding operator in pragmatic Content MathML. This role is taken over by the `big_intersect` symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <intersect/>
    <bvar><ci>x</ci></bvar>
    <apply><interval type="integer"/><cn>0</cn><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="arith1">big_intersect</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="interval1">integer_interval</csymbol><cn>0</cn><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.23.6 big_intersect

Description: This n-ary operator is used to construct the intersection over a collection of sets.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#big_intersect.

4.4.23.7 union (<union type="multiset"/>)

Description: This symbol is used to denote the n-ary union of multisets. It takes multisets as arguments, and denotes the multiset that contains all the elements that occur in any of them, with multiplicity the sum of all the multiplicities in the multiset arguments.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#union.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementunion with attributes type="multiset" in pragmatic Content MathML.

The union operator element can be used as a binding operator in pragmatic Content MathML. This role is taken over by the big_union symbol in strict Content MathML.

The pragmatic Content MathML expression

```
<apply>
    <union/>
    <bvar><ci>x</ci></bvar>
    <apply><interval type="integer"/><cn>0</cn><ci>x</ci></apply>
  </apply>
```

is equivalent to the strict Content MathML expression

```
<apply>
    <csymbol cd="arith1">big_union</csymbol>
    <bind>
      <csymbol cd="fns1">lambda</csymbol>
      <bvar><ci>x</ci></bvar>
      <apply><csymbol cd="interval1">integer_interval</csymbol><cn>0</cn><ci>x</ci></apply>
    </bind>
  </apply>
```

4.4.23.8 big_union

Description: This n-ary operator is used to construct the union over a collection of multisets.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#big_union.

4.4.23.9 `setdiff` (`<setdiff type="multiset"/>`)

Description: This symbol is used to denote the multiset difference of two multisets. It takes two multisets as arguments, and denotes the multiset that contains all the elements that occur in the first multiset with strictly greater multiplicity than in the second. The multiplicity in the result is the difference of the two.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#setdiff`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`setdiff` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.10 `subset` (`<subset type="multiset"/>`)

Description: This symbol has two (multiset) arguments. It is used to denote that the first set is a subset of the second, i.e. every element of the first occurs with multiplicity at least as much in the second.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#subset`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`subset` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.11 `in` (`<in type="multiset"/>`)

Description: This symbol has two arguments, an element and a multiset. It is used to denote that the element is in the given multiset.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#in`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`in` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.12 `notin` (`<notin type="multiset"/>`)

Description: This symbol has two arguments, an element and a multiset. It is used to denote that the element is not in the given multiset.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#notin`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`notin` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.13 `prsubset` (`<prsubset type="multiset"/>`)

Description: This symbol has two (multiset) arguments. It is used to denote that the first multiset is a proper subset of the second, that is a subset of the second multiset but not actually equal to it.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#prsubset`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`prsubset` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.14 `notsubset` (`<notsubset type="multiset"/>`)

Description: This symbol has two (multiset) arguments. It is used to denote that the first multiset is not a subset of the second.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#notsubset`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`notsubset` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.23.15 `notprsubset` (`<notprsubset type="multiset"/>`)

Description: This symbol has two (multiset) arguments. It is used to denote that the first multiset is not a proper subset of the second. A proper subset of a multiset is a subset of the multiset but not actually equal to it.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/multiset1.xhtml#notprsubset`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `notprsubset` with attributes `type="multiset"` in pragmatic Content MathML.

4.4.24 quant1

This CD holds the definitions of the basic universal ("for all") quantifier and existential ("there exists") quantifier. It is intended to be ‘compatible’ with the MathML elements representing these quantifiers.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/quant1.xhtml`.

4.4.24.1 `forall` (`<forall/>`)

Description: This symbol represents the universal ("for all") quantifier which takes two arguments. It must be placed within an OMBIND element. The first argument is the bound variables (placed within an OMBVAR element), and the second is an expression.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/quant1.xhtml#forall`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`forall` in pragmatic Content MathML.

4.4.24.2 `exists` (`<exists/>`)

Description: This symbol represents the existential ("there exists") quantifier which takes two arguments. It must be placed within an OMBIND element. The first argument is the bound variables (placed within an OMBVAR element), and the second is an expression.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/quant1.xhtml#exists`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`exists` in pragmatic Content MathML.

4.4.25 s_dist1

This CD holds the definitions of the basic statistical functions used on random variables. It is intended to be 'compatible' with the MathML elements representing statistical functions.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/s_dist1.xhtml`.

4.4.25.1 mean (<mean.dist/>)

Description: This symbol represents a unary function denoting the mean of a distribution. The argument is a univariate function to describe the distribution. That is, if f is the function describing the distribution. The mean is the expression integrate(x*f(x)) w.r.t. x over the range (-infinity,infinity).

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_dist1.xhtml#mean`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`mean.dist` in pragmatic Content MathML.

4.4.25.2 sdev (<sdev.dist/>)

Description: This symbol represents a unary function denoting the standard deviation of a distribution. The argument is a univariate function to describe the distribution. The standard deviation of a distribution is the arithmetical mean of the squares of the deviation of the distribution from the mean.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_dist1.xhtml#sdev`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`sdev.dist` in pragmatic Content MathML.

4.4.25.3 variance (<variance.dist/>)

Description: This symbol represents a unary function denoting the variance of a distribution. The argument is a function to describe the distribution. That is if f is the function which describes the distribution. The variance of a distribution is the square of the standard deviation of the distribution.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_dist1.xhtml#variance`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `variance.dist` in pragmatic Content MathML.

4.4.25.4 moment (<moment.dist/>)

Description: This symbol represents a ternary function to denote the i'th moment of a distribution. The first argument should be the degree of the moment (that is, for the i'th moment the first argument should be i), the second argument is the value about which the moment is to be taken and the third argument is a univariate function to describe the distribution. That is, if f is the function which describe the distribution. The i'th moment of f about a is the integral of (x-a)^i*f(x) with respect to x, over the interval (-infinity,infinity).

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_dist1.xhtml#moment`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element `moment.dist` in pragmatic Content MathML.

4.4.26 s_data1

This CD holds the definitions of the basic statistical functions used on sample data. It is intended to be 'compatible' with the MathML elements representing statistical functions, though it does not cover the concept of random variable which is mentioned in MathML.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/s_data1.xhtml`.

4.4.26.1 `mean` *(`<mean/>`)*

Description: This symbol represents an n-ary function denoting the mean of its arguments. That is, their sum divided by their number.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_data1.xhtml#mean`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`mean` in pragmatic Content MathML.

4.4.26.2 `sdev` *(`<sdev/>`)*

Description: This symbol represents a function requiring two or more arguments, denoting the sample standard deviation of its arguments. That is, the square root of (the sum of the squares of the deviations from the mean of the arguments, divided by the number of arguments). See CRC Standard Mathematical Tables and Formulae, editor: Dan Zwillinger, CRC Press Inc., 1996, (7.7.11) section 7.7.1.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_data1.xhtml#sdev`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`sdev` in pragmatic Content MathML.

4.4.26.3 `variance` *(`<variance/>`)*

Description: This symbol represents a function requiring two or more arguments, denoting the variance of its arguments. That is, the square of the standard deviation.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_data1.xhtml#variance`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`variance` in pragmatic Content MathML.

4.4.26.4 `mode` *(`<mode/>`)*

Description: This symbol represents an n-ary function denoting the mode of its arguments. That is the value which occurs with the greatest frequency.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_data1.xhtml#mode`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`mode` in pragmatic Content MathML.

4.4.26.5 `median` *(`<median/>`)*

Description: This symbol represents an n-ary function denoting the median of its arguments. That is, if the data were placed in ascending order then it denotes the middle one (in the case of an odd amount of data) or the average of the middle two (in the case of an even amount of data).

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_data1.xhtml#median`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`median` in pragmatic Content MathML.

4.4.26.6 `moment` *(`<moment/>`)*

Description: This symbol is used to denote the i'th moment of a set of data. The first argument should be the degree of the moment (that is, for the i'th moment the first argument should be i), the second argument should be the point about which the moment is being taken and the rest of the arguments are treated as the data. For n data values x_1, x_2, ..., x_n the i'th moment about c is (1/n) ((x_1-c)^i + (x_2-c)^i + ... + (x_n-c)^i). See CRC Standard Mathematical Tables and Formulae, editor: Dan Zwillinger, CRC Press Inc., 1996, section 7.7.1.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/s_data1.xhtml#moment`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`moment` in pragmatic Content MathML.

4.4.27 transc1

This CD holds the definitions of many transcendental functions. They are defined as in Abromowitz and Stegun (ninth printing on), with precise reductions to logs in the case of inverse functions. Note that, if signed zeros are supported, some strict inequalities have to become weak . It is intended to be 'compatible' with the MathML elements denoting trancendental functions. Some additional functions are in the CD transc2.

For a full semantic description of the content dictionary see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml`.

4.4.27.1 `log` *(`<log/>`)*

Description: This symbol represents a binary log function; the first argument is the base, to which the second argument is log'ed. It is defined in Abramowitz and Stegun, Handbook of Mathematical Functions, section 4.1

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#log`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`log` in pragmatic Content MathML.

4.4.27.2 `ln` *(`<ln/>`)*

Description: This symbol represents the ln function (natural logarithm) as described in Abramowitz and Stegun, section 4.1. It takes one argument. Note the description in the CMP/FMP of the branch cut. If signed zeros are in use, the inequality needs to be non-strict.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#ln`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`ln` in pragmatic Content MathML.

4.4.27.3 `exp` (`<exp/>`)

Description: This symbol represents the exponentiation function as described in Abramowitz and Stegun, section 4.2. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#exp`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`exp` in pragmatic Content MathML.

4.4.27.4 `sin` (`<sin/>`)

Description: This symbol represents the sin function as described in Abramowitz and Stegun, section 4.3. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#sin`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`sin` in pragmatic Content MathML.

4.4.27.5 `cos` (`<cos/>`)

Description: This symbol represents the cos function as described in Abramowitz and Stegun, section 4.3. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#cos`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`cos` in pragmatic Content MathML.

4.4.27.6 `tan` (`<tan/>`)

Description: This symbol represents the tan function as described in Abramowitz and Stegun, section 4.3. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#tan`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`tan` in pragmatic Content MathML.

4.4.27.7 `sec` (`<sec/>`)

Description: This symbol represents the sec function as described in Abramowitz and Stegun, section 4.3. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#sec`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`sec` in pragmatic Content MathML.

4.4.27.8 csc (<csc/>)

Description: This symbol represents the csc function as described in Abramowitz and Stegun, section 4.3. It takes one argument.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#csc.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementcsc in pragmatic Content MathML.

4.4.27.9 cot (<cot/>)

Description: This symbol represents the cot function as described in Abramowitz and Stegun, section 4.3. It takes one argument.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#cot.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementcot in pragmatic Content MathML.

4.4.27.10 sinh (<sinh/>)

Description: This symbol represents the sinh function as described in Abramowitz and Stegun, section 4.5. It takes one argument.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#sinh.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementsinh in pragmatic Content MathML.

4.4.27.11 cosh (<cosh/>)

Description: This symbol represents the cosh function as described in Abramowitz and Stegun, section 4.5. It takes one argument.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#cosh.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementcosh in pragmatic Content MathML.

4.4.27.12 tanh (<tanh/>)

Description: This symbol represents the tanh function as described in Abramowitz and Stegun, section 4.5. It takes one argument.

For a full semantic definition and more examples see http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#tanh.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator elementtanh in pragmatic Content MathML.

4.4.27.13 `sech` (`<sech/>`)

Description: This symbol represents the sech function as described in Abramowitz and Stegun, section 4.5. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#sech`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`sech` in pragmatic Content MathML.

4.4.27.14 `csch` (`<csch/>`)

Description: This symbol represents the csch function as described in Abramowitz and Stegun, section 4.5. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#csch`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`csch` in pragmatic Content MathML.

4.4.27.15 `coth` (`<coth/>`)

Description: This symbol represents the coth function as described in Abramowitz and Stegun, section 4.5. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#coth`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`coth` in pragmatic Content MathML.

4.4.27.16 `arcsin` (`<arcsin/>`)

Description: This symbol represents the arcsin function. This is the inverse of the sin function as described in Abramowitz and Stegun, section 4.4. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arcsin`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arcsin` in pragmatic Content MathML.

4.4.27.17 `arccos` (`<arccos/>`)

Description: This symbol represents the arccos function. This is the inverse of the cos function as described in Abramowitz and Stegun, section 4.4. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arccos`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arccos` in pragmatic Content MathML.

4.4.27.18 `arctan` (`<arctan/>`)

Description: This symbol represents the arctan function. This is the inverse of the tan function as described in Abramowitz and Stegun, section 4.4. It takes one argument.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arctan`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arctan` in pragmatic Content MathML.

4.4.27.19 `arcsec` (`<arcsec/>`)

Description: This symbol represents the arcsec function as described in Abramowitz and Stegun, section 4.4.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arcsec`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arcsec` in pragmatic Content MathML.

4.4.27.20 `arccsc` (`<arccsc/>`)

Description: This symbol represents the arccsc function as described in Abramowitz and Stegun, section 4.4.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arccsc`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arccsc` in pragmatic Content MathML.

4.4.27.21 `arccot` (`<arccot/>`)

Description: This symbol represents the arccot function as described in Abramowitz and Stegun, section 4.4.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arccot`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arccot` in pragmatic Content MathML.

4.4.27.22 `arcsinh` (`<arcsinh/>`)

Description: This symbol represents the arcsinh function as described in Abramowitz and Stegun, section 4.6.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arcsinh`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arcsinh` in pragmatic Content MathML.

4.4.27.23 `arccosh` (`<arccosh/>`)

Description: This symbol represents the arccosh function as described in Abramowitz and Stegun, section 4.6.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arccosh`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arccosh` in pragmatic Content MathML.

4.4.27.24 `arctanh` (`<arctanh/>`)

Description: This symbol represents the arctanh function as described in Abramowitz and Stegun, section 4.6.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arctanh`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arctanh` in pragmatic Content MathML.

4.4.27.25 `arcsech` (`<arcsech/>`)

Description: This symbol represents the arcsech function as described in Abramowitz and Stegun, section 4.6.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arcsech`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arcsech` in pragmatic Content MathML.

4.4.27.26 `arccsch` (`<arccsch/>`)

Description: This symbol represents the arccsch function as described in Abramowitz and Stegun, section 4.6.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arccsch`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arccsch` in pragmatic Content MathML.

4.4.27.27 `arccoth` (`<arccoth/>`)

Description: This symbol represents the arccoth function as described in Abramowitz and Stegun, section 4.6.

For a full semantic definition and more examples see `http://svn.openmath.org/OpenMath3/cd/MathML/transc1.xhtml#arccoth`.

Usage in pragmatic Content MathML: This symbol can be represented as the (empty) operator element`arccoth` in pragmatic Content MathML.

Chapter 5

Combining Presentation and Content Markup

Presentation markup and content markup can be combined in three ways. The first method, known as *mixed markup* is to intersperse content and presentation elements in what is essentially a single tree. The second method, known as *parallel markup* is to provide *both* explicit presentation markup and explicit content markup in a pair of trees. The third method is to provide *notation* elements as is done in Section 8.6 which enable processors to render content markup to presentation markup. This chapter describes how to use the first two methods to combine content and presentation markup, and how they may be used in conjunction with style sheets and other tools. Please refer to Section 8.6 for the specification of notation documents.

5.1 Motivation

Chapters 3 and 4 describe two kinds of markup that encode mathematical material in documents.

Presentation markup encodes the *notational structure* of an expression. It encodes the visual appearance of an expression in a way that facilitates rendering to various media. Thus, the same presentation markup can be rendered with relative ease on screen in wide or narrow windows, in ASCII or graphics, in print, or in sensible spoken language. Presentation markup supports these various renderings because it encodes notational information such as structured grouping of subexpressions, identification of mathematical symbols such as identifiers and operators, and disambiguation of invisible symbols such as for multiplication and function application.

Presentation markup is not directly concerned with the functional structure or mathematical semantics of an expression. In many situations, the notational structure and the functional structure are closely related, so a sophisticated processing application may be able to heuristically infer the functional structure from the notational structure, provided sufficient context is known. In practice, however, the inference of functional structure or mathematical meaning from mathematical notation must often be left to the reader.

While presentation markup is sufficient for a wide range of applications, employing presentation markup alone may limit the ability of some applications such as symbolic calculation systems to re-use the markup in another context.

Content markup encodes the *functional structure* of an expression. It encodes this structure in a sufficiently regular way that facilitates the assignment of mathematical semantics to an expression by application programs. Though the details of mapping from functional application structure to mathematical semantics can be extremely complex, in practice, there is wide agreement about the conventional meaning of many basic mathematical constructs. Consequently, much of the meaning of a content expression is easily accessible to a processing application, independent of where or how it is displayed to the reader. In many cases, content markup could be cut from a Web browser and pasted into a mathematical software tool with confidence that it will compute sensible values.

Since content markup is *not* directly concerned with how an expression is displayed, a renderer must infer how an expression should be presented to a reader. A sufficiently sophisticated renderer may provide transformational style sheets and other mechanisms that allow a user to interact with mathematical documents via their own personalized notational preferences. These preferences may then interact with the notational preferences expressed by the content author in a sensible way, which may require user intervention at some level.

While content markup is sufficient for a wide range of computational applications, employing content markup alone may limit the ability of the author to control precisely how an expression is rendered.

Both content and presentation markup are necessary to provide the full expressive capability one may require from in a mathematical markup language. In some cases, the same visual notation is used to represent several completely different mathematical concepts. For example, the notation x^i may be intended (in polynomial algebra) as the i-th power of the variable x, or (in tensor calculus) as the i-th component of a vector x. In other cases, the same mathematical concept may be displayed in one of various notations. For instance, the factorial of a number might be expressed with an exclamation mark, a Gamma function, or a Pochhammer symbol.

Thus, the same notation may represent several mathematical ideas, and conversely, the same mathematical idea often has several notations. Both content and presentation markup are needed to provide authors the ability to control visual notation in a form that is natural to a human user, and to encode functional structure in a form that is predictable for a computational system.

In general, when it is important to control exactly how an expression is rendered, presentation markup will be more satisfactory. When it is important that the meaning of an expression can be interpreted dependably and automatically, then content markup will be more satisfactory.

5.2 Semantic Annotations for Alternate Representations

Editor's note: Paul+SamThis section could be almost mute about the attribution key thus making it a simple rephrase of MathML2's semantic element and pushign the cd-name-cdbase requirements to the next one

An important concern of MathML is to associate specific semantics with a particular presentation, or additional presentation information with a content construct.

A *semantic annotation* decorates a MathML expression with a sequence of one or more pairs made up of a symbol (see), the 'attribute' or 'key', and an associated object, the 'value of the attribute'.

A semantic annotation is built up by the `semantics` element, which takes as the first child the MathML expression to be annotated. Subsequent children are `annotation-xml` (for values that are XML-encoded) and `annotation` (for values that are not) children that represent the attribute/value pairs. The key symbol allows to specify the relation of the annotation to the annotated element, it is referenced either by the `cdbase`, `cd`, `name` triplet or the `definitionURL` attribute (see). If none of these attributes are specified, the symbol is assumed to be the symbol `alternate-representation` from the `attribution-keys` content dictionary.

Editor's note: MiKoreference

For example in the MathML representation

```
<semantics>
  <mrow>
    <mrow>
      <mo>sin</mo>
      <mfenced open="(" close=")"><mi>x</mi></mfenced>
    </mrow>
    <mo>+</mo>
    <mn>5</mn>
  </mrow>
```

```
  <annotation-xml cd="mathml" name="contentequiv" encoding="MathML-Content">
    <apply>
      <csymbol cd="algebra-logic" name="plus"/>
      <apply><sin/><ci>x</ci></apply>
      <cn>5</cn>
    </apply>
  </annotation-xml>
  <annotation cd="maple" name="nativerep" encoding="text/maple">sin(x) + 5</annotation>
  <annotation cd="mathematica" name="nativerep" encoding="Mathematica">Sin[x] + 5</annotation>
  <annotation cd="TeX" name="plainTeXrep" encoding="TeX"> \sin x + 5</annotation>
  <annotation-xml cd="openmath" name="XMLencoding" encoding="OpenMath">
    <OMA xmlns="http://www.openmath.org/OpenMath">
      <OMA>
        <OMS cd="arith1" name="plus"/>
        <OMA><OMS cd="transc1" name="sin"/><OMV name="x"/></OMA>
      <OMI>5</OMI>
    </OMA>
  </annotation-xml>
</semantics>
```

binds together various representations of the sum of the sinus function applied to a variable x and the number 5. Essentially, we annotate the presentation element in the first child of the `semantics` element with various content-oriented representations. Each `annotation` and `annotation-xml` element specifies the nature of the annotation by referencing a key symbol in an appropriate content dictionary. For instance, the first `annotation-xml` element references the key symbol `"contentequiv"` from the `attribution-keys` content dictionary that specifies that the content MathML expression it provides is mathematically equivalent to the annotated presentation MathML expression.

Editor's note: PaulTODO: Indicate predefined values of encoding (MathML-Presentation and MathML-Content) as in Section 7.1.3.

The `annotation` element contains arbitrary parsed character data. If it contains the XML reserved characters `&`, `<`, `>`, `'`, `"`, then they must XML-escaped as `&`, `<`, `>`, `'`, `"` or the content must be enclosed in a `CDATA` section. Using a decoder for the encoding specified by the `encoding` attribute, the content is interpreted as a value for the key (symbol) is referenced either by the `cdbase`, `cd`, `name` triplet or the `definitionURL` attribute (see). It is recommended that its MIME type is used as the value of the `encoding` attribute. For example

```
<annotation-xml encoding="text/latex">
      <![CDATA[\documentclass{article}
      \begin{document}
      \title{E}
      \maketitle
      The base of the natural logarithms, approximately 2.71828.
      \end{document}]]>
    </annotation-xml>
```

5.2.1 The `annotation-xml` element

The `annotation-xml` element is analogous to the `annotation` element, except that the content can be an arbitrary XML sub-tree with a single root element. It is recommended that, where the contents of the foreign object are in an XML dialect, the namespace of the XML dialect is used as the value. For instance

```
<annotation-xml encoding="http://www.w3.org/1999/xhtml">
  <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>
```

5.2.2 Annotation references

In some cases the alternative children of a semantics element are not required for default behavior, but may be useful to specialized processors. For example, the presentation of a markup instance within a browser may not require additional annotations, but they may be needed when the markup is exported into another application. To enable the availability of several annotation formats in a more efficient manner, empty annotation and annotation-xml elements may be used to provide encoding and href attributes that specify a location for the expanded markup form for the annotation. Processing agents that anticipate that consumers of exported markup may not be able to retrieve the expanded form of such annotations should replace the annotation reference with the expanded form, by requesting the content from the indicated URL. We refer to Section 7.2 for more information about processing such elements within transfer paradigms such as copy-and-paste and drag-and-drop. A very simple example usage of this attribute could be:

```
<semantics>
  <mfrac><mi>a</mi><mrow><mi>a</mi><mo>+</mo><mi>b</mi></mrow></mfrac>
  <annotation encoding="image/png" href="333/formula56.png"/>
  <annotation encoding="text/maple" href="333/formula56.ms"/>
  </semantics>
```

5.3 Semantic Annotations beyond Alternate Representations

An *attribution* decorates a content MathML expression with a sequence of one or more semantic annotations. MathML uses the semantics, annotation-xml, and annotation elements introduced above for this.

An attribution acts as either adornment annotation or as semantical annotation. When the key has role "attribution", then replacement of the attributed object by the object itself is not harmful and preserves the semantics. When the key has role "semantic-attribution" then the attributed object is modified by the attribution and cannot be viewed as semantically equivalent to the stripped object. If the attribute lacks the role specification then attribution is acting as adornment annotation.

An example of the use of an adornment attribution would be to indicate the color in which an content representation object *A* should be displayed, for example

Editor's note: MiKoneed a much better example here, this one interferes with the ones above.

```
<semantics>
  A
  <annotation-xml cd="display" name="color" encoding="cMathML">
    red
  </annotation>
</semantics>
```

Note *red* are arbitrary representations whereas the key is a symbol.

An example of the use of a semantic attribution would be to indicate the type of an object. For example the object

```
<semantics>
  A
  <annotation-xml cd="mathmltypes" name="type" encoding="cMathML">
```

```
    t
  </annotation>
</semantics>
```

represents the judgment stating that object A has type t. Note that both A and t are arbitrary cMathML expressions.

Composition of semantic annotation, as in

```
<semantics>
  <semantics>AA_1  A_k</semantics>
  A_k+1  ...  A_n
 </semantics>
```

where the A_i are `annotation` or `annotation-xml` elements is equivalent to a single attribution, that is the semantic annotation

```
<semantics>
  A
  A_1  ...  A_n
</semantics>
```

The operation that produces an object with a single layer of semantic annotations is called *flattening*. Multiple annotations with the same key symbol are allowed. While the order of the given attributes does not imply any notion of priority, potentially it could be significant.

5.4 Mixed Markup

5.4.1 Reasons to Mix Markup

In many situations, an author or program may generate either presentation or content markup exclusively. For example, a program that translates legacy documents would likely generate pure presentation markup. Similarly, an educational software package might generate only content markup for evaluation in a computer algebra system. However, in many other situations, there are advantages to mixing both presentation and content markup within a single expression.

If an author is primarily concerned with presentation, interspersing some content markup may produce more accessible, more re-usable results with the content expressions rendered appropriately. For example, an author writing about linear algebra might write:

```
<mrow>
  <apply><power/><ci>x</ci><cn>2</cn></apply>
  <mo>+</mo>
  <msup><mi>v</mi><mn>2</mn></msup>
</mrow>
```

where v is a vector, the superscript denotes a vector component, and x is a real variable. Because of the linear algebra context, a visually impaired reader may direct a voice synthesis program to render superscripts as vector components. The explicit encoding of the power as content markup would then yield a much better voice rendering than would likely happen by default.

If an author is primarily concerned with content, there are two reasons to intersperse presentation markup. First, the use of presentation markup provides a way to modify or refine how a content expression is rendered. For example, one may write:

```
<apply>
  <in/>
  <ci><mi mathvariant="bold">v</mi></ci>
```

```
  <ci>S</ci>
</apply>
```

In this case, the use of embedded presentation markup allows the author to specify that v should be rendered in boldface. In the same way, it is sometimes the case that a completely different notation is desired for a content expression. For example, here we express a fact about factorials, $n = n!/(n\text{-}1)!$, using the ascending factorial notation:

```
<apply>
  <equivalent/>
  <ci>n</ci>
  <apply>
     <divide/>
     <semantics>
        <apply>
           <factorial/>
           <ci>n</ci>
        </apply>
        <annotation-xml encoding="MathML-Presentation">
           <msup>
              <mn>1</mn>
              <mover accent="true">
                 <mi>n</mi>
                 <mo>&OverBar;</mo>
              </mover>
           </msup>
        </annotation-xml>
     </semantics>
     <semantics>
        <apply>
           <factorial/>
           <apply><minus/><ci>n</ci><cn>1</cn></apply>
        </apply>
        <annotation-xml encoding="MathML-Presentation">
           <msup>
              <mn>1</mn>
              <mover accent="true">
                 <mrow><mi>n</mi><mo>-</mo><mn>1</mn></mrow>
                 <mo>&OverBar;</mo>
              </mover>
           </msup>
        </annotation-xml>
     </semantics>
   </apply>
</apply>
```

This content expression might render using the given notation as: $n \equiv \frac{1^{\overline{n}}}{1^{\overline{n-1}}}$

A second reason to use presentation markup within content markup is that there is a continually growing list of areas of discourse that do not have pre-defined content elements that encode their objects and operators. As a consequence, any system of content markup inevitably requires an extension mechanism to combine visual notation with function structure. MathML content markup specifies several ways of attaching an external semantic definitions to content objects. It is necessary, however, to use MathML presentation markup to specify how such user-defined semantic extensions should be rendered.

For example, the ‘rank’ operator from linear algebra is not included as a pre-defined MathML content element. Thus, to express the statement rank($u^T v$)=1 we may use a semantics element to bind a semantic definition to the symbol *rank*.

```
<apply>
  <eq/>
  <apply>
    <semantics>
      <mi>rank</mi>
      <annotation-xml encoding="OpenMath">
        <OMS name="rank" cd="linalg4" xmlns="http://www.openmath.org/OpenMath"/>
      </annotation-xml>
    </semantics>
    <apply>
      <times/>
      <apply> <transpose/> <ci>u</ci> </apply>
      <ci>v</ci>
    </apply>
  </apply>
  <cn>1</cn>
</apply>
```

Here, the semantics of rank have been given using a symbol from an OpenMath [OpenMath2000] content dictionary (CD).

5.4.2 Presentation Markup in Content Markup

The use of presentation markup within content markup is limited to situations that do not effect the ability of content markup to unambiguously encode mathematical meaning. 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 a 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 6) 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.

5.4.3 Content Markup in Presentation Markup

The guiding principle for embedding content markup within presentation expressions is that the resulting expression should still have an unambiguous rendering. In general, this means that embedded content expressions must be semantically meaningful, since rendering of content markup depends on its meaning.

The following content elements may not appear as an immediate child of a presentation element: `annotation`, `annotation-xml`, `bvar`, `condition`, `degree`, `logbase`, `lowlimit`, `uplimit`.

Within presentation markup, content markup may not appear within presentation token elements.

5.5 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.

5.5.1 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, for example, the presentation markup, and the other child element provides the content markup.

The following example encodes the boolean arithmetic expression $(a+b)(c+d)$ in this way.

```
<semantics>
  <mrow>
    <mrow><mo>(</mo><mi>a</mi> <mo>+</mo> <mi>b</mi><mo>)</mo></mrow>
    <mo>&InvisibleTimes;</mo>
    <mrow><mo>(</mo><mi>c</mi> <mo>+</mo> <mi>d</mi><mo>)</mo></mrow>
  </mrow>
  <annotation-xml encoding="MathML-Content">
    <apply><and/>
      <apply><xor/><ci>a</ci> <ci>b</ci></apply>
      <apply><xor/><ci>c</ci> <ci>d</ci></apply>
    </apply>
  </annotation-xml>
</semantics>
```

Note that the above markup annotates the presentation markup as the first child element, with the content markup as part of the `annotation-xml` element. An equivalent form could be given that annotates the content markup as the first child element, with the presentation markup as part of the `annotation-xml` element.

Top-level parallel markup should be strived for by applications that are able to since it provides recipients with an easier processing than partial semantic annoations.

5.5.2 Parallel Markup via Cross-References: `xml:id` and `xref`

To accommodate applications that must process sub-expressions of large objects, MathML can use cross-references between the branches of a `semantics` element to identify corresponding sub-structures. This application of the semantics elements and id-marking should be viewed as best practice to enable recipients to "select" arbitrary sub-expressions in all forms of the semantic-annotations alternatives.

These cross-references use xml:id and xref attributes within the branches of a containing semantics element. These 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 xml:id="E">
    <mrow xml:id="E.1">
      <mo xml:id="E.1.1">(</mo>
      <mi xml:id="E.1.2">a</mi>
      <mo xml:id="E.1.3">+</mo>
      <mi xml:id="E.1.4">b</mi>
      <mo xml:id="E.1.5">)</mo>
    </mrow>
    <mo xml:id="E.2">&InvisibleTimes;</mo>
    <mrow xml:id="E.3">
      <mo xml:id="E.3.1">(</mo>
      <mi xml:id="E.3.2">c</mi>
      <mo xml:id="E.3.3">+</mo>
      <mi xml:id="E.3.4">d</mi>
      <mo xml: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 xml:id attribute and a corresponding xref appearing within the same semantics element create a correspondence between sub-expressions.

All of the xml:id attributes referenced by any xref must be in the *same* branch of an enclosing semantics element. This constraint guarantees that these correspondences do not create unintentional cycles. (Note that this restriction does *not* exclude the use of xml:id attributes within the other branches of the enclosing semantics element. It does, however, exclude references to these other xml:id attributes originating in the same semantics element.)

There is no restriction on which branch of the semantics element may contain the destination xml: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 `xml: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 `xml:id` attributes.

In absence of other criteria, the first branch of the `semantics` element is a sensible choice to contain the `xml:id` attributes. Applications that add or remove annotations will then not have to re-assign attributes to the `semantics` trees.

In general, the use of `xml: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.

Top level markup with cross-references applies to any XML-encoded branch of the semantic annotations as is shown by the following example where the boolean expression of the previous section can be annotated with OpenMath, and cross-linked as follows:

```
<semantics>
  <mrow id="E">
    <mrow id="E.1">
      <mo id="E.1.1">(</mo>
      <mi id="E.1.2">a</mi>
      <mo id="E.1.3">+</mo>
      <mi id="E.1.4">b</mi>
      <mo id="E.1.5">)</mo>
    </mrow>
    <mo id="E.2">&InvisibleTimes;</mo>
    <mrow id="E.3">
      <mo id="E.3.1">(</mo>
      <mi id="E.3.2">c</mi>
      <mo id="E.3.3">+</mo>
      <mi id="E.3.4">d</mi>
      <mo id="E.3.5">)</mo>
    </mrow>
  </mrow>

  <annotation-xml encoding="MathML-Content">
    <apply xref="E">
      <and xref="E.2"/>
      <apply xref="E.1">
        <xor xref="E.1.3"/><ci xref="E.1.2">a</ci><ci xref="E.1.4">b</ci>
      </apply>
      <apply xref="E.3">
        <xor xref="E.3.3"/><ci xref="E.3.2">c</ci><ci xref="E.3.4">d</ci>
      </apply>
    </apply>
  </annotation-xml>

  <annotation-xml encoding="OpenMath"
                  xmlns:om="http://www.openmath.org/OpenMath">
```

```
    <om:OMA href="E">
      <om:OMS name="and" cd="logic1" href="E.2"/>

      <om:OMA href="E.1">
        <om:OMS name="xor" cd="logic1" href="E.1.3"/>
        <om:OMV name="a" href="E.1.2"/>
        <om:OMV name="b" href="E.1.4"/>
      </om:OMA>

      <om:OMA href="E.3">
        <om:OMS name="xor" cd="logic1" href="E.3.3"/>
        <om:OMV name="c" href="E.3.2"/>
        <om:OMV name="d" href="E.3.4"/>
      </om:OMA>
    </om:OMA>
  </annotation-xml>
</semantics>
```

Here `OMA`, `OMS` and `OMV` are elements defined in the OpenMath standard for representing application, symbol and variable, respectively. The references from the OpenMath annotation are given by the `href` attributes.

Chapter 6

Characters, Entities and Fonts

6.1 Introduction

Issue (): Many of the tables in chapter 6 need to be updated and regenerated. In this draft references to tables in chapter 6 link to the published MathML2 Recommendation, and are marked [MathML2]

Resolution: Separate xml-entity-names WD

Notation and symbols have proved very important for mathematics. Mathematics has grown in part because its notation continually changes toward being succinct and suggestive. There have been many new signs developed for use in mathematical notation, and mathematicians have not held back from making use of many symbols originally introduced elsewhere. The result is that mathematics makes use of a very large collection of symbols. It is difficult to write mathematics fluently if these characters are not available for use. It is difficult to read mathematics if corresponding glyphs are not available for presentation on specific display devices.

The W3C Math Working Group therefore took on directly the task of specifying part of the full mechanism needed to proceed from notation to final presentation, and has collaborated with the STIX Fonts Project and Unicode Technical Committee (UTC) in undertaking specification of the rest.

This chapter of the MathML specification contains a listing of character names for use with MathML, recommendations for their use, and warnings to pay attention to the correct form of the corresponding code points given in the UCS (Universal Character Set) as codified in Unicode and ISO 10646 [Unicode] and the Unicode Web site. For simplicity we refer to this character set by the short name Unicode. Though Unicode changes from time to time so that it is specified exactly by using version numbers, unless this brings clarity on some point we do not use them. MathML 2.0 (Second Edition) is based on Unicode 4.0, and MathML 3.0 on Unicode 5.1.)

While a long process of review and adoption by UTC and ISO/IEC of the characters of special interest to mathematics and MathML is now complete, more characters may be added in the future. To ensure any possible corrections to relevant standards are taken into account, and for the latest character tables and font information, see the W3C Entities page and the Unicode site, notably Unicode Work in Progress and Unicode Technical Report #25 "Unicode Support for Mathematics".

A MathML token element (see Section 3.2, Section 4.2.3, Section 4.2.4) takes as content a sequence of *MathML Characters*. MathML Characters are defined to be either Unicode characters legal in XML documents or `mglyph` elements. The latter are used to represent characters that do not have a Unicode encoding, as described in Section 3.2.9. Because the Unicode UCS provided approximately one thousand special alphabetic characters for the use of mathematics with Unicode 3.1, and over 900 further special symbols in Unicode 3.2, the need for `mglyph` should be rare.

6.2 Unicode Character Data

Any character allowed by XML may be used in MathML in an XML document. The legal characters have the hexadecimal code numbers 09 (tab = U+0009), 0A (line feed = U+000A), 0D (carriage return = U+000D), 20-

D7FF (U+0020..U+D7FF), E000-FFFD (U+E000..U+FFFD), and 10000-10FFFF (U+010000..U+10FFFF). The notation, just introduced in parentheses, beginning with U+ is that recommended by Unicode for referring to Unicode characters [see [Unicode], page xxviii]. The exclusions above code number D7FF are of the blocks used in surrogate pairs, and the two characters guaranteed not to be Unicode characters at all. U+FFFE is excluded to allow determination of byte order in certain encodings.

There are essentially three different ways of encoding character data.

- Using characters directly: For example, an A may be entered as 'A' from a keyboard (character U+0041). This option is only available if the character encoding specified for the XML document includes the character. Most commonly used encodings will have 'A' in the ASCII position. In many encodings, characters may need more than one byte. Note that if the document is, for example, encoded in Latin-1 (ISO-8859-1) then *only* the characters in that encoding are available directly. Using UTF-8 or UTF-16, the only two encodings that all XML processors are required to accept, mathematical symbols can be encoded as character data.
- Using numeric XML character references: Using this notation, 'A' may be represented as A (decimal) or A (hex). Note that the numbers always refer to the Unicode encoding (and not to the character encoding used in the XML file). By using character references it is always possible to access the entire Unicode range. For a general XML vocabulary, there is a disadvantage to this approach: character references may not be used in XML element or attribute names. However, this is not an issue for MathML, as all element names in MathML are restricted to ASCII characters.
- Using entity references: The MathML DTD defines internal entities that expand to character data. Thus for example the entity reference é may be used rather than the character reference "é or, if, for example, the document is encoded in ISO-8859-1, the character \'e. An XML fragment that uses an entity reference which is not defined in a DTD is not well-formed; therefore it will be rejected by an XML parser. For this reason *every* fragment using entity references *must* use a DOCTYPE declaration which specifies the MathML DTD, or a DTD that at least declares any entity reference used in the MathML instance. The need to use a DOCTYPE complicates inclusion of MathML in some documents. However, entity references are very useful for small illustrative examples, and are used in most examples in this document.

6.3 Entity Declarations

Earlier versions of this MathML specification included detailed listings of the entity definitions to be used with the MathML DTD. These entity definitions are of more general use, and have now been separated into a separate document, [Entities]. That document describes several entity sets, not all of them are used in the MathML DTD, although an XML document that includes MathML may reference any entity definitions. The standard MathML DTD references the following entity sets:

- isobox Box and Line Drawing
- isocyr1 Russian Cyrillic
- isocyr2 Non-Russian Cyrillic
- isodia Diacritical Marks
- isolat1 Added Latin 1
- isolat2 Added Latin 2
- isonum Numeric and Special Graphic
- isopub Publishing
- isoamsa Added Math Symbols: Arrow Relations
- isoamsb Added Math Symbols: Binary Operators
- isoamsc Added Math Symbols: Delimiters
- isoamsn Added Math Symbols: Negated Relations

- isoamso Added Math Symbols: Ordinary
- isoamsr Added Math Symbols: Relations
- isogrk3 Greek Symbols
- isomfrk Math Alphabets: Fraktur
- isomopf Math Alphabets: Open Face
- isomscr Math Alphabets: Script
- isotech General Technical
- mmlextra Additional MathML Symbols
- mmlalias MathML Aliases

6.4 Special Characters Not in Unicode

For special purposes, one may need to use a character which is not in Unicode. In these cases one may use the `mglyph` element for direct access to a glyph as an image, or (in some systems) from a font that uses a non-uniocde encoding. All MathML token elements that accept character data also accept an `mglyph` in their content. Beware, however, that use of `mglyph` to access a font is deprecated and the mechanism may not work in all systems. The `mglyph` element should always supply an alternatve representation in its `alt` attribute.

6.5 Mathematical Alphanumeric Symbols

A noticeable feature of mathematical and scientific writing is the use of single letters to denote variables and constants in a given context. The increasing complexity of science has led to the use of certain common alphabet and font variations to provide enough special symbols of this letter-like type. These denotations are in fact *not* letters that may be used to make up words with recognized meanings, but individual carriers of semantics themselves. Writing a string of such symbols is usually interpreted in terms of some composition law, for instance, multiplication. Many letter-like symbols may be quickly interpreted by specialists in a given area as of a certain mathematical type: for instance, bold symbols, whether based on Latin or Greek letters, as vectors in physics or engineering, or fraktur symbols as Lie algebras in part of pure mathematics. To this end the STIX Fonts Project defined a set of mathematical characters all of which are included in Unicode 5.0.

The additional Mathematical Alphanumeric Symbols provided in Unicode 3.1 have code points U+1D400..U+1D7FF in *Plane 1*, that is, in the first plane with Unicode values higher than 2^{16}. This plane of characters is also known as the Secondary Multilingual Plane (SMP), in contrast to the Basic Multilingual Plane (BMP) which was originally the entire extent of Unicode. Support for Plane 1 characters in currently deployed software is not always reliable, but it should be possible in multilingual operating systems, since Plane 2 has many Chinese characters that must be displayable in East Asian locales.

As discussed in Section 3.2.2, MathML offers an alternative mechanism to specify mathematical alphabetic characters. This alternative spans the gap between the specification of Unicode 3.1 and its associated deployment in software and fonts. Namely, one uses the `mathvariant` attribute on the surrounding token element, which will most commonly be `mi`. In this section we detail the correspondence that a MathML processor should apply between certain characters in *Plane 0* (BMP) of Unicode, modified by the `mathvariant` attribute, and the Plane 1 Mathematical Alphanumeric Symbol characters.

The basic idea of the correspondence is fairly simple. For example, a Mathematical Fraktur alphabet is in Plane 1, and the code point for Mathematical Fraktur A is U+1D504. Thus using these characters, a typical example might be

```
<mi>&#x1D504;</mi>
```

However, an alternative, equivalent markup would be to use the standard A and modify the identifier using the `mathvariant` attribute, as follows:

```
<mi mathvariant="fraktur">A</mi>
```

The exact correspondence between a mathematical alphabetic character and an unstyled character is complicated by the fact that certain characters that were already present in Unicode are not in the 'expected' sequence.

Mathematical Alphanumeric Symbol characters should not be used for styled text. For example, Mathematical Fraktur A must not be used to just select a blackletter font for an uppercase A. Doing this sort of thing would create problems for searching, restyling (e.g. for accessibility), and many other kinds of processing.

6.6 Non-Marking Characters

Some characters, although important for the quality of print or alternative rendering, do not have glyph marks that correspond directly to them. They are called here non-marking characters. Their roles are discussed in Chapter 3 and Chapter 4.

In MathML 2 control of page composition, such as line-breaking, is effected by the use of the proper attributes on the `mspace` element.

The characters below are not simple spacers. They are especially important new additions to the UCS because they provide textual clues which can increase the quality of print rendering, permit correct audio rendering, and allow the unique recovery of mathematical semantics from text which is visually ambiguous.

Unicode codepoint	Unicode name	Description
02061	FUNCTION APPLICATION	character showing function application in presentation tagging (Section 3.2.5
02062	INVISIBLE TIMES	marks multiplication when it is understood without a mark (Section 3.2.5
02063	INVISIBLE SEPARATOR	used as a separator, e.g., in indices (Section 3.2.5
02064*	INVISIBLE PLUS	marks addition, especialy in constructs such a $1\frac{1}{2}$ (Section 3.2.5

*Character U+2064 has been accepted by the UTC and ISO for inclusion into the next revision of Unicode, 5.1

Chapter 7

MathML interactions with the Wide World

Because MathML is, typically, embedded in a wider context, it is important to describe the conditions that processors should acknowledge in order to recognize XML fragments as MathML. This chapter describes the fundamental mechanisms to recognize and transfer MathML markup fragments within a larger environment such as an XML document or a desktop file-system, it raises the issues of combining external markup within MathML, then indicates how cascading style sheets can be used within MathML.

This chapter applies to both content and presentation MathML and indicates a particular processing model to the `semantics`, `annotation` and `annotation-xml` elements defined in Section 5.2.

7.1 Invoking MathML Processors: namespace, extensions, and mime-types

7.1.1 Recognizing MathML in an XML Model

Within an XML document supporting namespaces (TODO: cite xmlns and xml specs), the preferred method to recognize MathML markup is by the identification of the `math` element in the appropriate namespace, i.e. that of URI `http://www.w3.org/1998/Math/MathML`.

This is the recommended method to embed MathML within [XHTML] documents. Some user-agents' setup may require supplementary information to be available, such as the MicroSoft behaviour specification (TODO: quote) used in the MathType browser-extension (TODO:quote).

Markup-language specifications that wish to embed MathML may provide special conditions independent of this recommendation. The conditions should be equivalent and the elements' local-names should remain the same.

7.1.2 Resource Types for MathML Documents

Although rendering MathML expressions often occurs in place in a Web browser, other MathML processing functions take place more naturally in other applications. Particularly common tasks include opening a MathML expression in an equation editor or computer algebra system. It is important therefore to specify the encoding-names that MathML fragments should be called with:

MIME types [RFC2045], [RFC2046] offer a strategy that can be used in current user agents to invoke a MathML processor. This is primarily useful when referencing separate files containing MathML markup from an `embed` or `object` element, or within a desktop environment. (TODO: check that this still applies)

[RFC3023] assigns MathML the MIME type `application/mathml+xml` which is the official mime-type. The W3C Math Working Group recommends the standard file extension `.mml` within a registry associating file formats to file-extension. In MathML 1.0, `text/mathml` was given as the suggested MIME type. This has been superceded by RFC3023. In the next section, alternate encoding names are provided for the purposes of desktop transfers.

7.1.3 Names of MathML Encodings

MathML contains two distinct vocabularies: one for encoding mathematical semantics called Chapter 4 and one for encoding visual presentation called Chapter 3. Some MathML-aware applications import and export only one of these vocabularies, while other may be capable of producing and consuming both. Consequently, we propose three distinct MathML encoding names:

Flavor Name	Description	Deprecated
`MathML-Content`	Instance contains content MathML markup only	`MathML Content`, `Content MathML`, `cMathML`
`MathML-Presentation`	Instance contains presentation MathML markup only	`MathML Presentation`, `Presentation-MathML`, `pMathML`
`MathML`	Any well-formed MathML instance presentation markup, content markup, or a mixture of the two is allowed	

Any application producing one of the encodings above should ensure to output the values of the first column but should accept encoding names of the deprecated column.

7.2 Transferring MathML in Desktop Environments

MathML expressions are often exchanged between applications using the familiar copy-and-paste or drag-and-drop paradigms. This section provides recommended ways to process MathML while applying these paradigms.

Applying them will *transfer* MathML fragments between the contexts of two applications by making them available in several flavors, often called *clipboard formats* or *data flavors*. The copy-and-paste paradigm lets application *place* content in a central *clipboard*, one data-stream per *clipboard format*; consuming applications negotiate by choose to read the data of the format they elect. The drag-and-drop pardigm lets application offer content by declaring the available formats and potential recipients accept or reject a drop based on this list; the drop action then lets the receiving application request the delivery of the format in the indicated format. The list of flavors is generally ordered, going from the most wishable to the least wishable flavor.

Current desktop platforms offer both of these transfer paradigms using similar transfer architectures. In this section we specify what applications should provide as transfer-flavors, how they should be named, and how they should handle the special `semantics`, `annotation`, and `annotation-xml` elements.

To summarize the two negotiation mechanisms, we shall, here, be talking of *flavors*, each having a *name* (a character string) and a *content* (a stream of binary data), which are *exported*.

7.2.1 Basic Transfer Flavors' Names and Contents

Note that `MathML-Content`, `MathML-Presentation` and `MathML` are the exact strings that should be used to describe the flavors corresponding to the encodings in Section 7.1.3. On operating systems that allow such, applications should register such names (e.g. Windows' RegisterClipboardFormat).

When transferring MathML, for example when placing it within a clipboard, an application MUST ensure the content is a well-formed XML instance of a MathML schema. Specifically:

1. The instance MUST begin with a XML processing instruction, e.g. <?xml version="1.0">
2. The instance MUST contain exactly one root `math` element.
3. Since MathML is frequently embedded within other XML document types, the instance MUST declare the MathML namespace on the root `math` element. In addition, the instance SHOULD use a `schemaLocation` attribute on the `math` element to indicate the location of MathML schema documents

against which the instance is valid. Note that the presence of the `schemaLocation` attribute does not require a consumer of the MathML instance to obtain or use the cited schema documents.

4. The instance MUST use numeric character references (e.g. α) rather than character entity names (e.g. α) for greater interoperability.
5. The character encoding for the instance MUST be either specified in the XML header, UTF-16, or UTF-8. UTF-16-encoded data MUST begin with a byte-order mark (BOM). If no BOM or encoding is given, the character encoding will be assumed to be UTF-8.

7.2.2 Recommended Behaviours when Transferring

Applications that transfer MathML SHOULD adhere to the following conventions:

1. Applications that have pure presentation markup and/or pure content markup versions of an expression SHOULD offer as many of these two flavors as are available.
2. When both presentation and content are exported, recipients should consider it equivalent to a single MathML instance in which presentation and content are combined at the top level using MathML's `semantics` element (see Section 5.5.1). (TODO: issue: in DnD you can't read several, at least in java) The order between flavors determines whether presentation wraps content, or vice-versa. Usually, MathML-Presentation should be offered first so that it wraps the MathML-Content.
3. When an application has a mixed presentation and content version in addition to pure presentation and/or content versions, it should export the mixed versionafter the pure presentation and/or content markup versions, and mark it as the generic `MathML` flavor.
4. When an application cannot produce pure presentation and/or content markup versions, or cannot determine whether MathML data is pure presentation or content markup (e.g. data being passed through from a third application,) it should export only one version marked as the generic `MathML` flavor.
5. An application that only has pure presentation and/or content markup versions of an expression available SHOULD NOT export a second copy of the data marked as the generic `MathML` flavor.
6. When an application exports a MathML fragment whose root element is a `semantics` element, it SHOULD offer, after the flavors above, a flavor for each `annotation` or `annotation-xml` element: the flavor should be given by the `encoding` attribute value, and the content should be the child text in UTF-8 (if the `annotation` element contains only textual data), a valid XML fragment (if the `annotation-xml` element contains children), or the data resulting of requesting the URL given by the `href` attribute.
7. As a final fallback applications SHOULD export a version of the data in plain-text flavor (such as CF_UNICODETEXT, UnicodeText, NSStringPboardType, text/plain, ...). When an application has multiple versions of an expression available, it may choose the version to export as text at its discretion. Since some older MathML-aware programs expect MathML instances transferred as text to begin with a `math` element, the text version should generally omit the XML processing instruction, DOCTYPE declaration and other XML prolog material before the `math` element. Similarly, the BOM should be omitted for Unicode text encoded as UTF-16. Note, the Unicode text version of the data should always be the last flavor exported, following the principle that exported flavors should be ordered with the most specific flavor first and the least specific flavor last.

7.2.3 Discussion

For purposes of determining whether a MathML instance is pure content markup or pure presentation markup, the `math` element and the `semantics`, `annotation` and `annotation-xml` elements should be regarded as belonging to both the presentation and content markup vocabularies. This is obvious for the root `math` element which is required for all MathML expressions. However, the `semantics` element and its child annotation elements comprise an arbitrary annotation mechanism within MathML, and are not tied to either presentation or content markup. Consequently, applications consuming MathML should always process these four elements even if the application only implements one of the two vocabularies.

It is worth noting that the above recommendations allow agents producing MathML to provide binary data for the clipboard, for example as an image or an application-specific format. The sole method to do so is to reference the binary data by the `href` attribute since XML child-text does not allow arbitrary byte-streams.

While the above recommendations are intended to improve interoperability between MathML-aware applications utilizing the transfer flavors, it should be noted that they do not guarantee interoperablility. For example, references to external resources (e.g. stylesheets, etc.) in MathML data can also cause interoperability problems if the consumer of the data is unable to locate them, just as can happen when cutting and pasting HTML or many other data types. Applications that make use of references to external resources are encouraged to make users aware of potential problems and provide alternate ways for obtaining the referenced resources. In general, consumers of MathML data containing references they cannot resolve or do not understand should ignore them.

7.2.4 Examples

7.2.4.1 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:

Flavour Name	Flavor Content
`MathML`	
`Unicode Text`	

7.2.4.2 Example 2

An equation editor is able to generate pure presentation markup, valid with respect to MathML 2.0, 2nd Edition. Consequently, it exports the following flavors:

Flavour Name	Flavor Content
`MathML-Presentation`	
`Tiff`	(a rendering sample)
`Unicode Text`	

7.2.4.3 Example 3

A schema-based content management system contains multiple MathML representations of a collection of mathematical expressions, including mixed markup from authors, pure content markup for interfacing to symbolic computation engines, and pure presentation markup for print publication. Due to the system's use of schemas, markup is stored with a namespace prefix. The system therefore can transfer the following data:

Flavour Name	Flavor Content
`MathML-Presentation`	
`MathML-Content`	
`MathML`	
`TeX`	
`Unicode Text`	

7.2.4.4 Example 4

A similar content management system is web-based and delivers MathML representations of mathematiacly expressions. The system is able to produce presentation MathML, content MathML, TeX and pictures in PNG format. In web-pages being browsed, it could produce a MathML fragment such as the following:

```
<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
  <mml:semantics>
    <mml:mrow>...</mml:mrow>
    <mml:annotation-xml encoding="MathML content">...</mml:annotation-xml>
    <mml:annotation encoding="TeX">{1 \over x}</mml:annotation>
    <mml:annotation encoding="image/png" href="formula3848.png"/>
  </mml:semantics>
</mml:math>
```

A web-browser that receives such a fragment and tries to export it as part of a drag-and-drop action, can offer the following flavors:

Flavour Name	Flavor Content
`Presentation-MathML`	
`Content-MathML`	
`MathML`	
`TeX`	
`image/png`	(the content of the picture file, requested from formula3848.png
`Unicode Text`	

7.3 Combining MathML and Other Formats

Since MathML is most often generated by authoring tools, it is particularly important that opening a MathML expression in an editor should be easy to do and to implement. In many cases, it will be desirable for an authoring tool to record some information about its internal state along with a MathML expression, so that an author can pick up editing where he or she left off. The following markup is proposed:

1. For any extra information that is expected to be *semantically equivalent* MathML-3 proposes the usage of the `semantics` element presented in Section 5.2.
2. For any extra information that cannot be declared as such, and is, expectedly, private to the application. MathML-3 suggests to use the `maction`, see Section 3.6.1.

7.3.1 Mixing MathML and HTML

In order to fully integrate MathML into XHTML, it should be possible not only to embed MathML in XHTML, as described in Section 7.1.1, but also to embed XHTML in MathML. However, the problem of supporting XHTML in MathML presents many difficulties. Therefore, at present, the MathML specification does not permit any XHTML elements within a MathML expression, although this may be subject to change in a future revision of MathML.

In most cases, XHTML elements (headings, paragraphs, lists, etc.) either do not apply in mathematical contexts, or MathML already provides equivalent or better functionality specifically tailored to mathematical content (tables, mathematics style changes, etc.). However, there are two notable exceptions, the XHTML anchor and image elements. For this functionality, MathML relies on the general XML linking and graphics mechanisms being developed by other W3C Activities.

7.3.2 Linking

Issue (and-marking-ids): We wish to stop using xlink for links since it seems unimplemented and add the necessary attributes at presentation elements.

MathML has no element that corresponds to the XHTML anchor element *a*. In XHTML, anchors are used both to make links, and to provide locations to which a link can be made. MathML, as an XML application, defines links by the use of the mechanism described in the W3C Recommendation "XML Linking Language" [XLink].

A MathML element is designated as a link by the presence of the attribute `xlink:href`. To use the attribute `xlink:href`, it is also necessary to declare the appropriate namespace. Thus, a typical MathML link might look like:

```
<mrow xmlns:xlink="http://www.w3.org/1999/xlink"
      xlink:href="sample.xml">
  ...
</mrow>
```

MathML designates that almost all elements can be used as XML linking elements. The only elements that cannot serve as linking elements are those which exist primarily to disambiguate other MathML constructs and in general do not correspond to any part of a typical visual rendering. The full list of exceptional elements that cannot be used as linking elements is given in the table below.

MathML elements that cannot be linking elements

`mprescripts`	`none`
`malignmark`	`maligngroup`

Note that the XML Linking [XLink] and XML Pointer Language [XPointer] specifications also define how to link *into* a MathML expressions. Be aware, however, that such links may or may not be properly interpreted in current software.

7.3.3 Images

The `img` element has no MathML equivalent. The decision to omit a general mechanism for image inclusion from MathML was based on several factors. However, the main reason for not providing an image facility is that MathML takes great pains to make the notational structure and mathematical content it encodes easily available to processors, whereas information contained in images is only available to a human reader looking at a visual representation. Thus, for example, in the MathML paradigm, it would be preferable to introduce new glyphs via the `mglyph` element which at a minimum identifies them as glyphs, rather than simply including them as images.

7.3.4 MathML and Graphical Markup

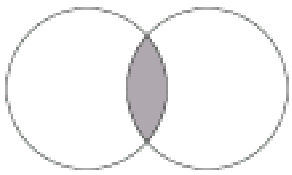
Apart from the introduction of new glyphs, many of the situations where one might be inclined to use an image amount to displaying labeled diagrams. For example, knot diagrams, Venn diagrams, Dynkin diagrams, Feynman diagrams and commutative diagrams all fall into this category. As such, their content would be better encoded via some combination of structured graphics and MathML markup. However, at the time of this writing, it is beyond the scope of the W3C Math Activity to define a markup language to encode such a general concept as 'labeled diagrams.' (See http://www.w3.org/Math for current W3C activity in mathematics and http://www.w3.org/Graphics for the W3C graphics activity.)

One mechanism for embedding additional graphical content is via the `semantics` element, as in the following example:

```
<semantics>
  <apply>
    <intersect/>
```

```
    <ci>A</ci>
    <ci>B</ci>
  </apply>
  <annotation-xml encoding="SVG1.1">
    <svg xmlns="http://www.w3.org/2000/svg"  viewBox="0 0 290 180">
      <clipPath id="a">
      <circle cy="90" cx="100" r="60"/>
      </clipPath>
      <circle fill="#AAAAAA" cy="90" cx="190"
              r="60" style="clip-path:url(#a)"/>
      <circle stroke="black" fill="none" cy="90" cx="100" r="60"/>
      <circle stroke="black" fill="none" cy="90" cx="190" r="60"/>
    </svg>
  </annotation-xml>
  <annotation-xml encoding="application/xhtml+xml">
    <img xmlns="http://www.w3.org/1999/xhtml" src="intersect.gif" alt="A intersect B"/>
  </annotation-xml>
</semantics>
```

Here, the `annotation-xml` elements are used to indicate alternative representations of the Content-MathML depiction of the intersection of two sets. The first one is in the 'Scalable Vector Graphics' format [SVG1.1] (see [XHTML-MathML-SVG] for the definition of an XHTML profile integrating MathML and SVG), the second one uses the XHTML `img` element embedded as an XHTML fragment. In this situation, a MathML processor can use any of these representations for display, perhaps producing a graphical format such as the image below.



Note that the semantics representation of this example is given in the Conten-MathML markup, as the first child of the `semantics` element. In this regard, it is the representation most analogous to the `alt` attribute of the `img` element in XHTML, and would likely be the best choice for non-visual rendering.

7.4 Using CSS with MathML

When MathML is rendered in an environment that supports [CSS2], controlling mathematics style properties with a CSS stylesheet is obviously desirable. MathML 2.0 has significantly redesigned the way presentation element style properties are organized to facilitate better interaction between MathML renderers and CSS style mechanisms. It introduces four new *mathematics style* attributes with logical values. Roughly speaking, these attributes can be viewed as the proper selectors for CSS rules that affect MathML.

Controlling mathematics styling is not as simple as it might first appear because mathematics styling and text styling are quite different in character. In text, meaning is primarily carried by the relative positioning of characters next to one another to form words. Thus, although the font used to render text may impart nuances to the meaning, transforming the typographic properties of the individual characters leaves the meaning of text basically intact. By contrast, in mathematical expressions, individual characters in specific typefaces tend to function as atomic symbols. Thus, in the same equation, a bold italic 'x' and a normal italic 'x' are almost always intended to be two distinct symbols that mean different things. In traditional usage, there are eight basic typographical categories of symbols. These categories are described by mathematics style attributes, primarily the `mathvariant` attribute.

Text and mathematics layout also obviously differ in that mathematics uses 2-dimensional layout. As a result, many of the style parameters that affect mathematics layout have no textual analogs. Even in cases where there are analogous properties, the sensible values for these properties may not correspond. For example, traditional mathematical typography usually uses italic fonts for single character identifiers, and upright fonts for multicharacter identifier. In text, italicization does not usually depend on the number of letters in a word. Thus although a font-slant property makes sense for both mathematics and text, the natural default values are quite different.

Because of the difference between text and mathematics styling, only the styling aspects that do not affect layout are good candidates for CSS control. MathML 3.0 captures the most important properties with the new mathematics style attributes, and users should try to use them whenever possible over more direct, but less robust, approaches. A sample CSS stylesheet illustrating the use of the mathematical style attributes is available in Appendix C. Users should not count on MathML implementations to implement any other properties than those in the Font, Colors, and Outlines families of properties described in [CSS2] and implementations should only implement these properties within MathML-elements. Note that these prohibitions do not apply to CSS stylesheets that implement the MathML-CSS profile. (TODO: quote).

TODO: add equivalence statements and conflict resolution and stress that CSS changes should not be considered meaningful.

Generally speaking, the model for CSS interaction with the math style attributes runs as follows. A CSS style sheet might provide a style rule such as:

```
math *.[mathsize="small"] {
  font-size: 80%
}
```

This rule sets the CSS font-size properties for all children of the `math` element that have the `mathsize` attribute set to small. A MathML renderer would then query the style engine for the CSS environment, and use the values returned as input to its own layout algorithms. MathML does not specify the mechanism by which style information is inherited from the environment. However, some suggested rendering rules for the interaction between properties of the ambient style environment and MathML-specific rendering rules are discussed in Section 3.2.2, and more generally throughout Chapter 3.

It should be stressed, however, that some caution is required in writing CSS stylesheets for MathML. Because changing typographic properties of mathematics symbols can change the meaning of an equation, stylesheet should be written in a way such that changes to document-wide typographic styles do not affect embedded MathML expressions. By using the MathML 2.0 mathematics style attributes as selectors for CSS rules, this danger is minimized.

Another pitfall to be avoided is using CSS to provide typographic style information necessary to the proper understanding of an expression. Expressions dependent on CSS for meaning will not be portable to non-CSS environments such as computer algebra systems. By using the logical values of the new MathML 3.0 mathematics style attributes as selectors for CSS rules, it can be assured that style information necessary to the sense of an expression is encoded directly in the MathML.

MathML 3.0 does not specify how a user agent should process style information, because there are many non-CSS MathML environments, and because different users agents and renderers have widely varying degrees of access to CSS information. In general, however, developers are urged to provide as much CSS support for MathML as possible.

Chapter 8

MathML3 Content Dictionaries

8.1 Introduction

The primary role of MathML content elements is to provide a mechanism for recording that a particular notational structure has a particular mathematical meaning. To this end, every content element must have a mathematical definition associated with it in some form. These definitions are provided in the form of *content dictionaries*, XML files of a certain structure (see Section 8.2).

The concept of a content dictionary has initially been introduced by the OpenMath1 format [OpenMath2000], and has been stabilized and generalized to *abstract content dictionaries* in the OpenMath2 standard [OpenMath2004], keeping a variant of OpenMath1 CDs as a reference encoding.

MathML 3 introduces a content dictionary format that is designed to support the MathML language, while meeting the requirements of OpenMath abstract CDs. We will introduce the format in the rest of the section and give an overview over the MathML3 content dictionaries for the K-14 fragment of Mathematics which is part of the MathML3 recommendation.

Editor's note: MiKoreference the final resting place or joint OM/W3C document here.

We will now detail the MathML3 Content Dictionary format, on an abstract level and discuss the special case of the MathML Recommendation CDs. Note that the latter are not the only possible ones, any individual or group can set up and publish CDs for the purposes of communication.

8.2 MathML3 Content Dictionaries Infrastructure and Metadata

MathML uses the namespace URI `http://www.w3.org/ns/mathml-cd` for the XML encoding of MathML content dictionaries. In the examples below, we will use the namespace prefix `mcd` for visual disambiguation and assume that it is bound to the URI above by the context of the example.

Issue (): Do we want a separate namespace for MathML CDs? David is suggesting that `http://www.w3.org/ns/mathml-cd` could be obtained without director's approval. In the following I am assuming that we will do this, even we do not have a formal decision on this in the group.

Resolution: The suggestion is carried.

A *MathML Content Dictionary* consists of a set of *symbol declarations* (see Section 8.3) together with administrative information about them. A MathML content dictionary is represented by the `mcd:mcd` element. The first child of the `mcd:mcd` element is a `mcd:description` element that contains a description of the collection of symbols defined by the CD. The content of the `mcd:description` is ????.

Issue (): We need to fix a content model for text fields in MCDs. This should probably be some fragment of XHTML+MathML, most probably the inline content model

Further administrative information about the CD as a whole is given by the following required attributes of the `mcd:mcd` element.

1. The *CD name* is given in the `xml:id` attribute.
2. The *revision date*, i.e. the date of the last change to the Content Dictionary is specified in the `revision-date`. Dates in MathML CDs are stored in the ISO-compliant format YYYY-MM-DD, e.g. 1966-02-03. For the MathML specification CDs the revision date is the date of the publication of the respective MathML recommendation.
3. The *review date*, i.e. a date until which the content dictionary is guaranteed to remain unchanged is specified in the `review-date` attribute.
   Issue (): There is not really a sensible review date for MathML3 spec CDs. We have no idea, when the spec will be revised. Unless we want to make them less normative than the MathML3 spec, and set up a revision process, then we will have to make this optional (and remove the requirement in OpenMath3)
   Resolution: We will drive for liberalizing the OpenMath Process
4. The CD *version number* consists of a major and minor part, it is specified in the `version` attribute. For the MathML specification CDs, this is the version number of the respective MathML recommendation.
5. The *status* of the CD is given in the `status` attribute. Its value is one of
   - `"official"`: i.e. approved by W3C as part of the MathML specification
   - `experimental`: under development and thus liable to change;
   - `private`: used by a private group of users;
   - `obsolete`: an obsolete Content Dictionary kept only for archival purposes.

   Issue (): OpenMath2 standard only allows `official` status for CDs approved by the OM Society. This seems overly proprietary. The MathML CDs should also be official. Change in OpenMath 3?
   Resolution: This is really an OpenMath Problem, which is recorded there. We should not worry here.
6. The *CD base* is a URI which, when combined with the CD name, forms a unique identifier for the Content Dictionary. It may or may not refer to an actual location from which it can be retrieved. The CD base is specified by the `cdbase` attribute.

8.3 Symbol Declarations

Editor's note: MiKo the material here still contains material that is just copied form OM-style abstract content, it will be reworked into a description of the MathML3 CD format soon. dictionaries.

MathML Content Dictionaries use the `mcd:MMLdefinition` element for symbol declarations. This element carries a mandatory `name` attribute that specifies the *name* of the declared symbol. Its value is an XML1.1 name [xml11]. The *role* of the symbol is specified in the optional `role` attribute (see Section 8.5 for details and values. The syntactic and semantic properties symbol are given by the following specialized elements in the body of the `mcd:MMLdefinition` element:

1. A short *description* of the symbol. It can be accompanied by a `discussion`, which can be as formal or informal as the author likes. These are given as `description` and `discussion` elements whose content are ???.
2. A mandatory *default rendering* specification given in the form of a `mcd:notation` element (see Section 8.6 for details).
3. Zero or more *attribute specifications* for the case, where the MathML token takes attributes.
   Editor's note: MiKospecify this
4. Zero or more *signature declarations* for type information (see Section 8.4 for details).
5. Zero or more *commented mathematical properties* which are mathematical properties of the symbol expressed in a human-readable way.
6. Zero or more *properties* which are mathematical properties of the symbol. A property can be expressed in natural language and as a MathML expression tree in the same property. The former is directly aimed at human readers, and the latter could be used for validation or evaluation in mathematical software systems. property may be given an optional *kind* attribute. An author of a Content Dictionary may use

this to indicate whether, for example, the property provides an algorithm for evaluation of the concept it is associated with. At present no fixed scheme is mandated for how this information should be encoded or used by an application.

7. Zero or more mathematical *examples* which are intended to demonstrate the use of the symbol within a content MathML expression tree.

8.4 Type Declarations

Editor's note: MiKocopy parts of the types note here, develop signature declarations for all symbols in the CDs, and make `mathmltypes` and `STS` CDs.

8.5 Symbol Roles

We say that a symbol is used to *construct* an MathML expression tree if it is the first child of an `apply`, `bind` or `error` element. The *role* of a symbol is a restriction on how it may be used to construct a compound expression tree and, in the case of the key in an attribution object, a clarification of how that attribution should be interpreted. The possible roles are:

- *binder* The symbol may appear as the first child of a `bind` element.
- *application* The symbol may appear as the first child of an `apply` element.
- *constant* The symbol cannot be used to construct a compound expression tree.
- *error* The symbol may appear as the first child of a `error` element.

Issue (): Those are the roles in OpenMath. Do we need more in MathML? We could have one for constructor (so that we know that it is a container element in legacy markup)... That could later be mapped to application. But maybe this would be better done by the `classification` element.

A symbol cannot have more than one role and cannot be used to construct a compound expression tree object in a way which requires a different role (using the definition of construct given earlier in this section). This means that one cannot use a symbol which binds some variables to construct, say, an application object. However it does not prevent the use of that symbol as an *argument* in an application object (where by argument we mean a child with index greater than 1).

If no role is indicated then the symbol can be used anywhere. Note that this is not the same as saying that the symbol's role is *constant*.

8.6 Rendering of Content Elements

Issue (): How do we want to specify renderings. There are a couple of options:

- as an image: universal, but not helpful for implementors
- as presentation MathML
- multi-format

in the latter cases: what do we do for role:application? There are solutions from ORCCA or OMDoc

Resolution: We will use the format specified below

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. Therefore it is important that content MathML have a native infrastructure for specifying notations for content symbols. These specifications can be used either directly as a parameter to a *generic rendering process* that can thus adapt to extensible mathematical vocabularies or to inform the implementation of *specialized rendering processes* for restricted vocabularies.

As notations are tied to content symbols, content dictionaries seem like a natural place: a generic rendering process can look up notation definitions in the CD specified in the `csymbol` element; a specialized rendering procedure can delimit it's area of applicability by the CDs it covers (e.e. the MathML3 CDs). As mathematical notation is highly variable even for fundamental concepts, CDs can only contain (sets of) of *default notation specifications* that can be overridden e.g. by user preferences. The MathML3 specification does not specify any mechanism for building generic or specialized rendering processors, or for selecting the relevant notations in a given context. Instead we specify a function from content MathML expressions to presentation MathML expressions that takes a list of notation specifications as an input. We will call this function the *MathML3 rendering function*, even though strictly speaking its values are presentation MathML expressions that have to be rendered by a MathML-aware processor to be rendered for visual or aural consumption.

Note that the mechanism of notation specifications can directly be transferred for generating renderings or representations in other target formats. In fact it can even be reversed to be a source of information for interpretation procedures.

Issue (): Should we specify an attribute `format` for the `mcd:rendering` elements and fix/default it to `"pMathML"`? That would make it simpler to extend the specification for other formats, I suggest that we should do this.

Resolution: The WG does not think this is a good idea, we should promote pMathML not choice.

In the rest of the section we will specify the format of notation specifications and then define the MathML3 rendering function based on this.

8.6.1 Notation Specifications

MathML specifies notations using a template-based mechanism. In essence, a notation specification is a pair consisting of schematic content MathML expression (called the *prototype*) together with a schematic presentation MathML expression (called the *rendering*). Schematic MathML expressions are expressions that can contain *metavariables*, and thus stand for a set of MathML expressions. A prototype/rendering pair directly specifies a correspondence between a set of content MathML expressions (those that match the prototype) and presentation MathML expressions (the result of instantiating the metavariables in the rendering with the pMathML expressions corresponding to the values of the matcher). Consider, for instance the following prototype/rendering pair:

```
<apply>
  <csymbol cd="algebra_logic" name="minus"/>
  <mcd:expr name="left"/>
  <mcd:expr name="right"/>
</apply>
```

```
<mrow>
  <mcd:render name="left"/>
       <mo>-</mo>
       <mcd:render name="right"/>
</mrow>
```

This specifies that any application expression whose first child is a ‘minus’ symbol will correspond to an `mrow` with an infix ‘-’ operator. The meta-variables in the prototype are represented by `expr` elements and by `mcd:render` elements in the rendering, they correspond if they share the value of the `name` attribute. Thus the content MathML expression

```
<apply><csymbol cd="algebra_logic" name="minus"/><cn>2</cn><cn>1</cn></apply>
```

corresponds to the presentation MathML expression

```
<mrow><mn>2</cn><mo>-</moL<mn>1</mn></mrow>
```

assuming a suitable correspondence for numbers.

MathML3 uses four elements for representing metavariables, i.e. named variables that stand for arbitrary MathML expression trees or tree lists: `expr`, `exprlist`, `mcd:render`, and `iterate`. The first two are called *content* *metavariables* and the latter two *rendering metavariables*. The `expr` and `mcd:render` elements are called *element metavariables* and stand for single MathML expressions, while the `exprlist` `iterate` elements are called *sequence metavariables* and stand for possibly empty sequences of MathML expressions. Content metavariables stand for (lists of) cMathML expressions whereas rendering metavariables stand for (lists of) pMathML expressions. Finally, we call a MathML expression *schematic*, if it contains one or more metavariables.

All metavariables carry the required `name` attribute, whose value is an XML name. The `expr` el is empty, and the `exprlist` can contain an arbitrary schematic cMathML expression. The `mcd:rendering` element contains an arbitrary sequence of schematic pMathML expressions. The `iterate` element carries the optional `reverse` attribute, whose values can be `"yes"` and `"no"`. The body of the `iterate` element consists of a `separator` element arbitrary sequence of schematic pMathML expressions. Finally, the content of the `separator` element is a arbitrary sequence of schematic pMathML expressions.

Prototypes and renderings are represented by the `mcd:prototype` and `mcd:rendering` elements in notation specification. The former contains schematic content MathML expression with input metavariables, and the latter contains a schematic presentation MathML expression with rendering metavariables. A prototype may not contain two sibling `exprlist` elements and no two content metavariables may have the same `name` attribute. The `mcd:prototype` element carries the optional `priority` attribute whose value is a natural number; if the attribute is missing, the *priority* of the prototype defaults to 0. The `mcd:rendering` element carries the optional `xml:lang`, `context`, and `variant` attributes. The values of the `xml:lang` attribute are ISO 639 language specifiers. The values of the `context` and `variant` attribute are currently not specified and left to applications.

Intuitively, a cMathML expression E *matches* a prototype P, iff there is a mapping σ from metavariables in P to (lists of) cMathML expressions, such that if σ is applied to P, then the result is E. The formal definition of matching schematic content MathML expressions below is somewhat more complicated, since we need to take sequence metavariables into account, and the `semantics` elements have a built-in notion of flattening. For given E and P, we know that if E matches P, then σ is unique; it is called the *matcher* for E and P.

Let σ be a mapping from metavariable names to lists of MathML expressions. We say that a sequence E of cMathML expressions *matches* a sequence of prototypes P via σ, iff one of the following holds

1. P is a single `expr` metavariable with name n and $\sigma(n) = E$
2. P and E are flattened `semantics` elements, P' and E', their first children and the $Q = Q(1)...Q(n)$ and $C = C(1)...C(n+k)$ are the sequences of those `annotation-xml` children of P and E whose keys have the role `"semantic-annotation"`, sorted lexicographically via their `cdbase` `cd`, and `name` attributes. Then E matches P via σ, iff $Q(i)$ match $C(i+k)$ for $1 \leqslant i \leqslant n$, and `<semantics>`$E'C(1)...C(k-1)$ `</semantics>` matches P' σ.
3. P and E are single elements that have the same name, and the attributes of P is a subset of those of E and their values coincide, and the sequence of children of E matches that of P via σ.
4. P and E are sequences of the form $P(1)...P(n)$ and $E(1)...E(m)$ and
   (a) none of the Pi is a sequence metavariable and $n = m$ and
   (b) all the $E(i)$ match $P(i)$ via σ.
5. P and E are sequences of the form $P(1)...P(n)$ and $E(1)...E(m)$ and
   (a) $P(j)$ is a `exprlist` element with name ν for some $1 \leqslant j \leqslant n$ (thus none of the $P(i)$ are by the no-sibling-constraint above).
   (b) $E(i)$ match $P(i)$ via σ for $i < j$.
   (c) $E(n-i)$ match $P(m-i)$ via σ for $1 \leqslant i < n-j$.
   (d) $\sigma(\nu) = E(j)...E(m-j)$
   (e) $P(j)$ has children $C = C(1)...C(l)$ and $m-n = kl$
   (f) $\sigma(\nu)$ matches the sequence given by the k-fold concatenation of C with itself via σ.

For given prototype P and expression E, we know that there is at most one mapping σ such that E matches P via σ.

Issue (): What is our notion of equality here? XML equality? Infoset equality? Do we want to specify this at all?

Resolution: We will have to define our own

A notation element usually occurs as part of a MMLdefinition element for its top symbol (i.e. the highest token or csymbol in the pattern expression trees). If it appears in some other context (e.g. as part of a user's notation preferences file), then it should reference the top symbol via the cd, name, and possibly cdbase attributes to ensure that it can be found by the rendering process.

For representation conciseness and manageability, MathML3 groups prototypes and renderings in notation elements. We call a list of mcd:rendering elements in a notation element a *rendering block*, iff it is delimited by mcd:prototype elements or the end tag of the notation element itself. No two mcd:rendering elements in a rendering block may have the same values for all three of the xml:lang, context, and variant attributes. We say that a cMathML expression *selects* the rendering block following the prototype with the highest priority among those that it matches.

Multiple mcd:prototype elements can be used e.g. to make a notation specification applicable to both strict and pragmatic cMathML expressions e.g. by adding a prototype involving the element <plus/> element. Multiple mcd:rendering elements in a rendering block can be used to model different notations or language conventions; they are differentiated by their xml:lang, context, and variant attributes. The xml:lang and context attributes allows to select a rendering variant via a global context. The former by language context to enable multilingual rendering. The selection mechanism of the context attribute is currently unspecified and left to applications. The variant attribute allows to select a notation variant locally on a cMathML element via its variant attribute. If a content element carries a class attribute and matches one of the prototypes of a notation element, then the mcd:rendering element whose variant attribute value has the same value will be chosen for rendering. (see Section 8.6.4)

The intended meaning of a notation element is that a cMathML expression corresponds to that rendering in the selected rendering block that is suitable in the current context.

Issue (selector): We could have a possibility to re-use the selector language of CSS 2.1, which is big but well accepted, or we could have a possibility to restrict to style class-names (as described above). Both would offer some predictability for authors.

A simple example is provided below, it describes the notations for the open real interval in English and other languages:

```
<notation cd="basic_content_elements" name="interval-oo">
  <mcd:prototype>
    <apply><interval type="open"/>
      <mcd:expr name="a"/>
      <mcd:expr name="b"/>
    </apply>
  </mcd:prototype>
  <mcd:prototype>
    <apply>
      <csymbol cd="basic_content_elements" name="interval-oo"/>
      <mcd:expr name="a"/>
      <mcd:expr name="b"/>
    </apply>
  </mcd:prototype>
  <mcd:rendering xml:lang="en">
    <math>
      <mrow>
        <mo>(</mo><mcd:render name="a"/><mo>,</mo><mcd:render name="b"/><mo>)</mo>
```

```
      </mrow>
    </math>
  </mcd:rendering>
  <mcd:rendering>
    <math>
      <mrow>
        <mo>]</mo><mcd:render name="a"/><mo>,</mo><mcd:render name="b"/><mo>[</mo>
      </mrow>
    </math>
  </mcd:rendering>
</notation>
```

The cMathML expression

```
<apply>
  <interval type="open"/>
  <apply><minus/><ci>n</ci><ci>ε</ci></apply>
  <apply><plus/><ci>n</ci><ci>ε</ci></apply>
</math>
```

matches the first prototype in the notation specification above. The matcher σ maps <arg name="a"/> to <apply><minus/><ci>n</ci><ci>ε</ci></apply> and <arg name="b"/> to <apply><minus/><ci>n</ci><ci>ε</ci></apply>. In a French language environment, the second mcd:rendering is selected, since the English notation in the first target does not apply. Therefore the cMathML expression renders to

```
<mrow><mo>]</mo><mrow>a-ε</mrow><mo>,</mo><mrow>a+ε</mrow><mo>[</mo></mrow>
```

assuming suitable notation specifications for plus and minus elements.

Multiple notation specifications per symbol are explicitly allowed, they can be used e.g. when writing a notation specification for the derivative which would be presented differently if applied on a simple function or a function defined using the lambda binding which indicates explicitly the variable of derivation. Thus the following notation specification with two rendering blocks is used:

```
<notation cd="calculus_veccalc" name="diff">
  <mcd:prototype>
    <apply>
      <csymbol cd="calculus_veccalc" name="diff"/>
      <bind>
        <csymbol name="basic_content_elements" name="lambda"/>
        <bvar><mcd:expr name="x"/></<bvar>
        <mcd:expr name="f"/>
      </bind>
    </apply>
  </mcd:prototype>
  <mcd:rendering>
    <mfrac>
      <mrow><mo>d</mo><mcd:render name="f"/></mrow>
      <mrow><mod>d</mod><mcd:render name="x"/></mrow>
    </mfrac>
  </mcd:rendering>
  <mcd:prototype>
    <apply>
      <csymbol cd="calculus_veccalc" name="diff"/>
```

```
      <mcd:expr name="f"/>
    </apply>
  </mcd:prototype>
  <mcd:rendering><mrow><mcd:render name="f"/><mo>'</mo></mrow></mcd:rendering>
</notation>
```

Many operations take an arbitrary number of arguments. These are modeled using schematic MathML expressions containing the `exprlist` metavariable, which matches a list of cMathML expressions in a prototype. In a `mcd:rendering` element, an `exprlist` element can contain arbitrary pMathML content. Consider for instance the notation specification for addition:

```
<notation cd="algebra_logic" name="plus">
  <mcd:prototype>
    <apply>
      <plus/>
      <mcd:exprlist name="summands">
<mcd:expr name="sumand"/>
      </mcd:exprlist>
    </apply>
  </mcd:prototype>
  <mcd:rendering>
    <mcd:iterate name="summands">
      <separator><mo>+</mo></separator>
      <render name="summand"/>
    </iterate>
  </mcd:rendering>
</notation>
```

If we apply this notation specification to the cMathML expression

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

then the metavariable `summands` matches the list `<ci>a</ci><ci>b</ci><ci>c</ci>`. In this situation, rendering of the corresponding `exprlist` is the list of rendered elements interleaved by the content of the `exprlist`. In our case we obtain

```
<mi>a</mi><mo>+</mo><mi>b</mi><mo>+</mo><mi>c</mi>
```

Editor's note: MiKoThe following examples might be material for the primer though

To further fortify our intuition, let us consider a complex example. We want to render a content representation of a multiple integral expression e.g.:

```
<bind>
  <apply><int/><ci>P</ci><ci>Q</ci><ci>R</ci></apply>
  <bvar><ci>x</ci><ci>y</ci><ci>z</ci></bvar>
  <condition>
    <apply><and/>
      <apply><gt/><ci>x</ci><cn>1</cn></apply>
      <apply><gt/><ci>y</ci><cn>0</cn></apply>
      <apply><gt/><ci>z</ci><cn>-1</cn></apply>
    </apply>
  </condition>
  <apply><plus/>
    <ci>x</ci>
    <apply><times/>
      <cn>2</cn>
```

```
      <apply><power/><ci>y</ci><cn>2</cn></apply>
    </apply>
    <apply><times/>
      <cn>3</cn>
      <apply><power/><ci>z</ci><cn>3</cn></apply>
    </apply>
  </apply>
</bind>
```

The intended rendering

```
<mrow>
   <mtable rowalign="center">
     <mtr><mo>&int;&int;&int;</mo></mtr>
     <mtr><mi>P</mi><mi>Q</mi><mi>R</mi></mtr>
     <mrow>
       <mrow><mi>x</mi><mo>&gt;</mo><mn>1</mn></mrow>
       <mo>&and;</mo>
       <mrow><mi>y</mi><mo>&gt;</mo><mn>0</mn></mrow>
       <mo>&and;</mo>
       <mrow><mi>z</mi><mo>&gt;</mo><mn>-1</mn></mrow>
    </mrow>
  </mtable>
  <mrow>
    <mi>x</mi>
    <mo>+</mo>
    <mrow><mn>2</mn><mo>&InvisibleTimes;</mo><msup><mi>y</mi><mn>2</mn></msup></mrow>
    <mo>+</mo>
    <mrow><mn>3</mn><mo>&InvisibleTimes;</mo><msup><mi>z</mi><mn>3</mn></msup></mrow>
  </mrow>
  <mrow><mo>d</mo><mi>x</mi><mo>d</mo><mi>y</mi><mo>d</mo><mi>z</mi></mrow>
</mrow>
```

can be obtained with the following notation specification:

```
<notation for="int">
  <mcd:prototype>
    <bind>
      <apply><int/>
<mcd:exprlist name="sets"><mcd:expr name="set"/></mcd:exprlist>
      </apply>
      <bvar>
<mcd:exprlist name="bvars"><mcd:expr name="bvar"/></mcd:exprlist>
      </bvar>
      <condition><mcd:expr="cond"/></condition>
      <mcd:expr name="scope"/>
    </bind>
  </mcd:prototype>
  <mcd:rendering>
    <mrow>
      <mtable rowalign="center">
        <mtr><mo><iterate name="sets"><separator>&int;</separator></iterate></mo></mtr>
        <mtr>
```

```
  <iterate name="sets">
    <separator><mo>&times;</mo></separator>
    <mcd:render name="set"/>
  </iterate>
</mtr>
        <mcd:render name="cond"/>
      </mtable>
      <mcd:render name="scope"/>
      <iterate name="bvars"><mo>d</mo><mcd:render name="bvar"/></iterate>
    </mrow>
  </mcd:rendering>
</notation>
```

Note that the prototype picks up the structural backbone of the integral expression in the `mcd:prototype` element, and introduces metavariables for the variable parts of integral expressions. While matching the example, the meta variable `sets` is bound to the list *P*, *Q*, *R* of sets that are integrated over. This list is used twice in the rendering: once for generating the three integral glyphs in the integration operator (here we do not have a `mcd:render` element so the sets themselves are not rendered) and once in the cartesian product under the integration operator. The list of bound variables is picked up in the end of the rendering to generate the *dxdydz* postfix.

The final example shows the use of non-empty `exprlist` metavariables

```
<mcd:notation>
  <mcd:prototype>
    <bind>
      <apply mcd:cr="multint"><int/>
        <mcd:exprlist name="domains">
          <apply>
            <interval mcd:cr="int"/>
            <mcd:expr name="lower_bound"/>
            <mcd:expr name="upper_bound"/>
          </apply>
        </mcd:exprlist>
      </apply>
      <bvars>
<mcd:exprlist name="variables"><mcd:expr name="var"/></mcd:exprlist>
      </bvars>
      <condition><mcd:expr name="condition"/></condition>
      <mcd:expr name="body"/>
    </bind>
  </mcd:prototype>
  <mcd:rendering>
    <mrow>
      <msub mcd:cr="multint">
        <mrow>
          <mcd:iterate name="domains">
            <msubsup>
              <mo mcd:cr="int">&int;</mo>
             <mcd:render name="lower_bound"/>
           <mcd:render name="upper_bound"/>
            </msubsup>
          </mcd:iterate>
```

```
</mrow>
        <mcd:render name="condition"/>
      </msub>
      <mcd:render name="body"/>
      <mcd:iterate name="variables" reverse="yes">
        <mo>d</mo>
        <mcd:render name="var"/>
      </mcd:iterate>
    </mrow>
  </mcd:rendering>
<mcd:notation>
```

This notation renders the cMathML expression

```
<bind>
  <apply><int/>
    <apply><interval/><ci>a</ci><ci>b</ci></apply>
    <apply><interval/><ci>c</ci><ci>d</ci></apply>
  </apply>
  <bvars><ci>x</ci><ci>y</ci></bvars>
  <condition><apply><gt/><ci>x</ci><ci>y</ci></apply></condition>
  <apply><ci>f</ci><ci>x</ci><ci>y</ci></apply>
</bind>
```

as

```
<mrow>
  <msub>
    <mrow>
      <msubsup><mo>&int;</mo><mi>a</mi><mi>b</mi></msubsup>
      <msubsup><mo>&int;</mo><mi>c</mi><mi>d</mi></msubsup>
    </mrow>
    <mrow><mi>x</mi><mo>&gt;</mo><mi>y</mi></mrow>
  </msub>
  <mrow><mi>f</mi><mo>(</mo><mi>x</mi><mi>y</mi><mo>)</mo></mrow>
  <mo>d</mo><mi>y</mi>
  <mo>d</mo><mi>x</mi>
</mrow>
```

Note that the matcher here consists of two mappings, one binds the top-level metavariables `domain`, `variables`, `condition`, and `body`, and the second one is indexed by the sequence variable `domain` and binds the metavariables `lower_bound` and `upper_bound` in its scope. The latter two can only be used in the rendering inside the `iterate` element that renders `domains`.

8.6.2 Precedence-based Elisions

In content MathML expressions, the function/argument relation is fully specified by the prefix notation in expression trees. In traditional mathematical notation, operator placement is much less restricted and brackets are used sparingly to disambiguate the functional structure. However, the use of brackets is usually restricted to cases, where the functional structure cannot be derived from the two-dimensional structure of the notations and conventions about binding strength of operators. This distribution of brackets has to be re-created to obtain high-quality renderings for cMathML expressions, and therefore the binding strengths have to be modeled in notation specifications.

Issue (naming): The precedence currently used here may clash with the precedence rules explained within the mo description in presentation-mathml We should clear things: if it is compatible, if it is the same, if it extends it.

Resolution: We have decided that the two concepts are different concepts, and that we should not worry

To account for this MathML3 add an `precedence` attributes to the `mcd:rendering` element (to specify the the *operator precedence*) and the `mcd:render` and `iterate` elements (to specify the the *argument precedence*). The value of this attribute is an integer or $+\infty$ or $-\infty$. If the `precedence` attribute is not present on a `mcd:rendering` element, its operator precedence the default value 0. If the `precedence` attribute is not present on a `mcd:render` or `iterate` element the respective argument precedence is the same as the operator precedence.

The operator precedence allows rendering agents to decide on fencing the constructed rendering. Intuitively operators with larger precedence bind more strongly, so need not be fenced. Correspondingly, the rendering of an expression is enclosed in fences, iff the operator precedence is greater than the current precedence. For the next level of rendering this is set to the respective argument precedence of the rendering metavariable that triggers the recursive rendering. For top-level formulae renderings the current precedence has the default value 0. This ensures that outer brackets are usually elided, since most operators have positive operator precedence.

In MathML, fences are considered as a special case of semantic components that are subject to *elision*, i.e. components that can be left out of the presentation in certain situations. Even though brackets are the prime examples other components of mathematical formulae, e.g. bases of logarithms, are also commonly elided, if they can be derived from the context with little effort. In MathML, a component of a rendering is marked to be elidable by adding the `mcd:egroup` attribute. This attribute specifies the *elision group*. MathML reserves the values `"fence"` and `"separator"` for fences and separators in correpsondence to the attributes `fence` and `separator` (see Section 3.2.5). In a situation, where the current precedence is c and the operator precedence of the fence (i.e. the value of the `precedence` attribute on the `notation` element that contains it) is o, then the fence is given a *visibility level* of $o-c$ (we take $\infty-\infty$ to be ∞ to make fences visible for safety in this degenerate case). Rendering applications can specify other elision groups and give elidable components an explicit visibility level as integers using the `mcd:elevel` attribute. If none is given it defaults to the value 0.

Elision can take various forms in print and digital media. In static media like traditional print on paper or the PostScript format, we have to fix the elision level, and can decide at presentation time which elidable tokens will be printed and which will not. In this case, the presentation algorithm will take visibility thresholds $T(g)$ for every elision group g as a user parameter and then elide (i.e. not render) all tokens in visibility group g with level $l > T(g)$.

In an output format that is capable of interactively changing its appearance, e.g. dynamic XHTML+MathML (i.e. XHTML with embedded Presentation MathML formulae, which can be manipulated via JavaScript in browsers), an application can export the the information about elision groups and levels to the target format, and can then dynamically change the visibility thresholds by user interaction.

For example, the following `notation` element could be used for the factorial operator:

```
<notation cd="algebra-logic" name="factorial">
  <mcd:prototype>
    <apply><factorial/><mcd:expr name="arg"/></apply>
  </mcd:prototype>
  <mcd:rendering precedence="500">
    <mrow>
      <mo mcd:egroup="fence">(</mo>
        <mcd:render name="arg"/>
      <mo mcd:egroup="fence">)</mo>
      <mo>!</mo>
    </mrow>
  </mcd:rendering>
</notation>
```

In a situation with a current precedence (c) of 800 and a threshold $T(fence) = 0$ the following cMathML expression

```
<apply>
  <csymbol cd="algebra-logic" name="#factorial"/>
  <apply><csymbol cd="algebra-logic" name="plus"/><ci>x</ci><ci>y</ci></apply>
</apply>
```

would be rendered by a static renderer as

```
<mrow>
  <mo>(</mo>
    <mrow><mi>x</mi><mo>+</mo><mi>y</mi></mrow>
  <mo>)</mo>
  <mo>!</mo>
</mrow>
```

assuming an operator precedence of 100 for the addition operator (the argument precedences do not matter here, since identifiers are never fenced): $500 - 800 = -300 < 0$, so the outer fences are elided, and $500 - 100 = 400 > 0$, so the factorial fences are rendered. In a situation with a smaller current precedence e.g. 400, the outer fences would be rendered as well.

A dynamic renderer would pass the computed visibility levels to the rendering. In our situation with a current precedence of 800 we would obtain

```
<mo mcd:egroup="fence" mcd:elevel="-300">(</mo>
<mrow>
  <mo mcd:egroup="fence" mcd:elevel="400">(</mo>
    <mrow><mi>x</mi><mo>+</mo><mi>y</mi></mrow>
  <mo mcd:egroup="fence" mcd:elevel="400">)</mo>
  <mo>!</mo>
</mrow>
<mo mcd:egroup="fence" mcd:elevel="-300">)</mo>
```

Thus for a fencing threshold of 0 we would get the same result. With a user-given threshold larger than 400 (only components of a high visibility level are not elided) all fences would be elided, and with a threshold smaller than -300, all brackets would be made visible.

The requirement to have operator and argument precedences is probably most clearly seen in the case of binary associative operators we would use the following notation specification:

```
<notation cd="binary" name="plus">
  <mcd:prototype>
    <apply><csymbol cd="binary" name="plus"/>
      <mcd:expr name="left"/>
      <mcd:expr name="right"/>
    </apply>
  </mcd:prototype>
  <mcd:rendering  precedence="500">
    <mrow mcd:egroup="fence">
      <mo mcd:egroup="fence">(</mo>
        <mcd:render name="left" precedence="501"/>
<mo>+</mo>
        <mcd:render name="right" precedence="501"/>
      <mo mcd:egroup="fence">)</mo>
    </mrow mcd:egroup="fence">
  </mcd:rendering>
```

```
</notation>
```

In fact the content MathML expression

```
 <apply><csymbol cd="binary" name="plus"/>
  <apply><csymbol cd="binary" name="plus"/>
    <ci>x</ci>
    <ci>y</ci>
    </apply>
  <apply><csymbol cd="binary" name="plus"/>
    <ci>z</ci>
    <ci>w</ci>
  </apply>
</apply>
```

would be rendered as

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

Left- or right-associative binary operators (i.e. operators like the function space constructor that only elide parentheses in one of their arguments) are simply constructed by decreasing only one of their argument precedences.

8.6.3 Rendering Cross-References for Parallel Markup

The metavariable correspondence in `notation` elements is a good source for crossreferences in parallel markup with crossreferences (see Section 5.5.2): If the rendering application constructs parallel markup, then it can cross-reference the top-level elements in corresponding `prototype` and `rendering` elements, as well as the metavariables that have the name value of the `name` element. If this is not enough, then additional crossreferencing opportunities can be specified by adding `mcd:cr` attributes with equal values to corresponding elements in the prototypes and renderings in a notation declaration. For instance, in above example we add cross-references for the multiple integral operator and for the integrals operators it is made up of. Note that `mcd:cr` attributes can be added to all MathML elements, but their values have to be unique in any `prototype` and `rendering` elements.

8.6.4 General Rules

In this section we will specify the default rendering of cMathML expressions that do not match any of the notation definitions given to the rendering function. The default renderings of pragmatic content MathML are given by the default rendering of the corresponding strict ones. We will go over the cases for strict MathML expressions in turn:

8.6.4.1 Numbers

The default rendering of a simple `cn`-tagged object is the same as for the presentation element `mn` with some provision for overriding the presentation of the `PCDATA` by providing explicit `mn` tags. This is described in detail in Section 4.2.3.

8.6.4.2 Symbols and Identifiers

If the content of a `ci` or `csymbol` element is tagged using presentation tags, that presentation is used. If no such tagging is supplied then the `PCDATA` 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.

8.6.4.3 Applications

If F is the rendering of f and Ai those of ai, then the default rendering of an application element of the form

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

is

```
<mrow>
    F
    <mo mcd:egroup="fence" fence="true">(</mo>
    A1
    <mo mcd:egroup="separator" separator="true">,</mo>
    ...
    <mo mcd:egroup="separator" separator="true">,</mo>
    An
    <mo  mcd:egroup="fence" fence="true">)</mo>
</mrow>
```

This general rule could also be expressed by the following notation definition:

```
<mcd:notation>
 <mcd:prototype>
    <apply>
       <mcd:expr name="fun"/>
       <mcd:exprlist name="args"><mcd:expr name="arg"/></mcd:exprlist>
    </apply>
 <mcd:prototype>
 <mcd:rendering>
    <mrow>
       <mcd:render name="fun"/>
       <mo fence="true">(</mo>
       <mcd:iterate name="args"/>
         <mcd:separator><mo separator="true">,</mo></mcd:separator>
 <mcd:render name="arg"/>
       </mcd:iterate>
       <mo fence="true">)</mo>
   </mrow>
 <mcd:rendering>
</mcd:notation>
```

8.6.4.4 Binders

If b, c, xi, c, and s render to B, C, Xi, C, and S, then the default rendering of a binding element of the form

```
<bind>b<bvar>x1  ...  xn</bvar>C  S</bind>
```

is

```
<mrow>
  B
  x1
  <mo mcd:egroup="separator" separator="true">,</mo>
  ...
  <mo mcd:egroup="separator" separator="true">,</mo>
  xn
  <mo mcd:egroup="separator" separator="true">:</mo>
  C
  <mo mcd:egroup="separator" separator="true">.</mo>
  S
</mrow>
```

If the `condition` is not present, then the fragment `<mo mcd:egroup="separator" separator="true">:</mo>`*C* is omitted from the rendering.

This general rule could also be expressed by the following notation definition:

```
<mcd:notation>
 <mcd:prototype>
    <bind>
       <mcd:expr name="bex"/>
       <bvar><exprlist name="bvars"><expr name="bvar"/></mcd:exprlist>
       <condition><mcd:expr name="condition"/></condition>
       <mcd:expr name="body"/>
    </bind>
 <mcd:prototype>
 <mcd:rendering>
   <mrow>
     <mcd:render name="bex"/>
     <mcd:iterate name="bvars"/>
       <mcd:separator><mo mcd:egroup="separator" separator="true">,</mo></mcd:separator>
       <mcd:render name="bvar"/>
     </mcd:iterate>
     <mo mcd:egroup="separator" separator="true">:</mo>
     <mcd:render name="condition"/>
     <mo mcd:egroup="separator" separator="true">.</mo>
     <mcd:render name="body"/>
   </mrow>
 <mcd:rendering>
 <mcd:prototype>

       <mcd:expr name="bex"/>
       <bvar><exprlist name="bvars"><expr name="bvar"/></mcd:exprlist>
       <mcd:expr name="body"/>
    </bind>
 <mcd:prototype>
 <mcd:rendering>
   <mrow>
     <mcd:render name="bex"/>
     <mcd:iterate name="bvars"/>
       <mcd:separator><mo mcd:egroup="separator" separator="true">,</mo></mcd:separator>
```

```
        <mcd:render name="bvar"/>
      </mcd:iterate>
      <mo mcd:egroup="separator" separator="true">.</mo>
      <mcd:render name="body"/>
    </mrow>
  <mcd:rendering>
</mcd:notation>
```

8.6.4.5 Attributions

The default rendering of a `semantics` element is the default rendering of its first child: the `annotation` and `annotation-xml` are not rendered. When a MathML-presentation annotation is provided, a MathML renderer may optionally use this information to render the MathML construct. This would typically be the case when the first child is a MathML content construct and the annotation is provided to give a preferred rendering differing from the default for the content elements.

8.6.4.6 Structure Sharing

The default rendering of a `share` is that of the MathML element pointed to by the URI in the `href` attribute.

8.6.4.7 Rest

Editor's note: MiKodo all the rest

8.6.5 Limitations and Extensions of Notation Documents

The elements proposed in this section provide a basis for exchangeable notation-documents which can be processed by rendering agents for the conversion of content-elements, to presentation MathML.

There is a great wealth of conversion tools from content to presentation. Compared to hand-written XSLT stylesheets, such as ctop.xsl (TODO: quote), the expression matching is quite poor and the programming facilities are almost nonexistent (thus it seems not possible, yet, to specify that the 7 in $\frac{\partial^7}{\partial_x^3 \partial^4} f$ is computed automatically), the approach of notation-documents is more declarative and opens the door to exchangeability, moreover, it has the potential to respect user and language-dependent notations.

For the many more dynamic rendering agents, which includes all content-oriented input-editors, notation-documents may be a good way to render dynamically symbols *just found on the web*.

Appendix A

Parsing MathML

A.1 Use of MathML as Well-Formed XML

Issue ():DTD and W3C XML Schema need updating to MathML3

Issue (): Should we add a (normative?) Relax NG schema.

Resolution: We make it normative

A MathML document must be a well-formed XML document using elements in the MathML namespace as defined by this specification, however it is not required that the document refer to any specific Document Type Definition (DTD) or schema that specifies MathML. It is sometimes advantagous *not* to specify such a language definition as these files are large, often much larger than the MathML expression and unless they have been previously cached by the MathML application, the time taken to fetch the DTD or schema may have an appreciable effect on the processing of the MathML document.

Note also that if no DTD is specified with a DOCTYPE declaration, that entity references (for example to refer to MathML characters by name) may not be used. The document should be encoded in an encoding (for example UTF-8) in which all needed characters may be encoded as character data, or characters may be referenced using numeric character references, for example ∫ rather than ∫

If a MathML fragment is parsed without a DTD, in other words as a well-formed XML fragment, it is the responsibility of the processing application to treat the white space characters occurring outside of token elements as not significant.

However, in many circumstances, especially while producing or editing MathML, it is useful to use a language definition, to constrain the editing process or to check the correctness of generated files. The following section, Section A.2, discusses the RelaxNG Schema for MathML3 [RelaxNG], which forms a normative part of the specification. Following that, Section A.4, and Section A.3 discuss alternative languages definition using the document type definitions (DTD) and the W3C XML schema language, [XMLSchemas], both of which are derived from the normative RelaxNG schema automatically. One should note that the schema definitions of the language is currently stricter than the DTD version. That is, a schema validating processor will declare invalid documents that are declared valid by a (DTD) validating XML parser. This is partly due to the fact that the XML schema language may express additional constraints not expressable in the DTD, and partly due to the fact that for reasons of compatibility with earlier releases, the DTD is intentionally forgiving in some places and does not enforce constraints that are specified in the text of this specification.

A.2 Using the RelaxNG Schema for MathML3

MathML documents should be validated using the RelaxNG Schema for MathML, either in the XML encoding (`http://www.w3.org/Math/RelaxNG/mathml3/mathml3.rng`) or in compact notation (`http://www.w3.org/Math/RelaxNG/mathml3/mathml3.rnc`) which is also shown below.

In contrast to DTDs there is no in-document method to associate a RelaxNG schema with a document.

Editor's note: MiKoI have included the schema verbatim for reference, a better version should be generated somehow

We provide five RelaxNG schemata for sub-languages of MathML3:

- The grammar for Presentation MathML without content elements mixed in
- The grammar for strict Content MathML3
- The grammar for pragmatic Content MathML3 without presentation MathML in token elements
- The grammar for full MathML without deprecated parts
- The grammar for full MathML with deprecated parts

we will present them in detail in the next sections below. As the compact notation for RelaxNG grammars is more readable, we will use this format here.

Note that the RelaxNG grammars here are considerably more strict than the MathML2 DTDs (even in strict mode).

A.2.1 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 1998-2007 World Wide Web Consortium
#        (Massachusetts Institute of Technology, Institut National de
#         Recherche en Informatique et en Automatique, Keio University).
#         All Rights Reserved.
#
#     Permission to use, copy, modify and distribute the RelaxNG schema for MathML3
#     and its accompanying documentation for any purpose and without fee is
#     hereby granted in perpetuity, provided that the above copyright notice
#     and this paragraph appear in all copies.  The copyright holders make
#     no representation about the suitability of the Schema for any purpose.
#
#     This file contains the grammar rules for pure presentation MathML3, i.e. without
#     content MathML mixed in.
#     It is provided "as is" without expressed or implied warranty.
#
#     Revision:   $Id: mathml3-presentation.rnc,v 1.1 2008/03/27 13:35:25 mkohlhas2 Exp $
#     Author: Michael Kohlhase http://kwarc.info/kohlhase

default namespace m = "http://www.w3.org/1998/Math/MathML"

include "mathml3-common.rnc"

math.content |= ContInPres

MathML.Common.attrib |= attribute class {xsd:NMTOKENS}?,attribute style {xsd:string}?


Browser-interface.attrib = attribute baseline {xsd:string}?,
                           attribute overflow {"scroll" | "elide" | "truncate" | "scale"}?,
```

```
                           attribute altimg {xsd:anyURI}?,
                           attribute alttext {xsd:string}?,
           attribute type {xsd:string}?,
   attribute name {xsd:string}?,
   attribute height {xsd:string}?,
   attribute width {xsd:string}?

math.attlist |= Browser-interface.attrib,attribute display {"block" | "inline"}?

simple-size = "small" | "normal" | "big"

centering.values = "left" | "center" | "right"

named-space = "veryverythinmathspace" | "verythinmathspace" | "thinmathspace" |
              "mediummathspace" |
              "thickmathspace" | "verythickmathspace" | "veryverythickmathspace"
thickness = "thin" | "medium" | "thick"

# number with units used to specified lengths

length-with-unit =
    xsd:string #{pattern="(-?([0-9]+|[0-9]*\.[0-9]+)(em|ex|px|in|cm|mm|pt|pc|%))|0"}
length-with-optional-unit =
   xsd:string #{pattern="-?([0-9]+|[0-9]*\.[0-9]+)(em|ex|px|in|cm|mm|pt|pc|%)?"}

# This is just "infinity" that can be used as a length
infinity = "infinity"

# colors defined as RGB
RGB-color = xsd:string {pattern="#(([0-9]|[a-f]){3}|([0-9]|[a-f]){6})"}

# The mathematics style attributes. These attributes are valid on all
#     presentation token elements except "mspace" and "mglyph", and on no
#     other elements except "mstyle".

Token-style.attrib = attribute mathvariant
       {"normal" | "bold" | "italic" | "bold-italic" | "double-struck" |
                        "bold-fraktur" | "script" | "bold-script" | "fraktur" |
  "sans-serif" | "bold-sans-serif" | "sans-serif-italic" |
"sans-serif-bold-italic" | "monospace"}?,
                     attribute mathsize {simple-size | length-with-unit}?,

                     attribute mathcolor {xsd:string}?,
         attribute mathbackground {xsd:string}?

truefalse = "true" | "false"

Operator.attrib =
# this attribute value is normally inferred from the position of
# the operator in its "<mrow">
```

```
   attribute form {"prefix" | "infix" | "postfix"}?,
   # set by dictionary, else it is "thickmathspace"
   attribute lspace {length-with-unit | named-space}?,
   # set by dictionary, else it is "thickmathspace"
   attribute rspace {length-with-unit | named-space}?,
   # set by dictionnary, else it is "false"
   attribute fence {truefalse}?,
   # set by dictionnary, else it is "false"
   attribute separator {truefalse}?,
   # set by dictionnary, else it is "false"
   attribute stretchy {truefalse}?,
   # set by dictionnary, else it is "true"
   attribute symmetric {truefalse}?,
   # set by dictionnary, else it is "false"
   attribute movablelimits {truefalse}?,
   # set by dictionnary, else it is "false"
   attribute accent {truefalse}?,
   # set by dictionnary, else it is "false"
   attribute largeop {truefalse}?,
   attribute minsize {length-with-unit | named-space}?,
   attribute maxsize {length-with-unit | named-space | infinity | xsd:float}?


mglyph = element {mglyph} {MathML.Common.attrib,
                     attribute alt {xsd:string}?,
                     (attribute src {xsd:anyURI}| attribute fontfamily {xsd:string}),
     attribute width {xsd:string}?,
     attribute height {xsd:string}?,
     attribute baseline {xsd:string}?,
     attribute index {xsd:positiveInteger}?}


linethickness.attrib = attribute linethickness {length-with-optional-unit|thickness}
mline = element {mline} {MathML.Common.attrib,
      linethickness.attrib?,
      attribute spacing {xsd:string}?,
      attribute length {length-with-unit | named-space}?}

Glyph-alignmark = malignmark|mglyph

mi = element {mi} {MathML.Common.attrib,Token-style.attrib,(Glyph-alignmark|text)*}

mo = element {mo} {MathML.Common.attrib,Operator.attrib,Token-style.attrib,
                 (text|Glyph-alignmark)*}

mn = element {mn} {MathML.Common.attrib,Token-style.attrib,(text|Glyph-alignmark)*}

mtext = element {mtext} {MathML.Common.attrib,Token-style.attrib,(text|Glyph-alignmark)*}

ms = element {ms} {MathML.Common.attrib,Token-style.attrib,
```

```
                 attribute lquote {xsd:string}?,
 attribute rquote {xsd:string}?,
 (text|Glyph-alignmark)*}

# And the group of any token
Pres-token = mi | mo | mn | mtext | ms

msub = element {msub} {MathML.Common.attrib,
                  attribute subscriptshift {length-with-unit}?,
                  ContInPres,ContInPres}

msup = element {msup} {MathML.Common.attrib,
                  attribute supscriptshift {length-with-unit}?,
                  ContInPres,ContInPres}

msubsup = element {msubsup} {MathML.Common.attrib,
                     attribute subscriptshift {length-with-unit}?,
                     attribute supscriptshift {length-with-unit}?,
                     ContInPres,ContInPres,ContInPres}

munder = element {munder} {MathML.Common.attrib,
                         attribute accentunder {truefalse}?,
                         ContInPres,ContInPres}

mover = element {mover} {MathML.Common.attrib,
                       attribute accent {truefalse}?,
                       ContInPres,ContInPres}

munderover = element {munderover} {MathML.Common.attrib,
                                 attribute accentunder {truefalse}?,
                                 attribute accent {truefalse}?,
                                 ContInPres,ContInPres,ContInPres}

PresExp-or-none = ContInPres | none
mmultiscripts = element {mmultiscripts}{MathML.Common.attrib,
                              ContInPres,
      (PresExp-or-none,PresExp-or-none)*,
      (mprescripts,(PresExp-or-none,PresExp-or-none)*)?}
none = element {none} {empty}
mprescripts = element {mprescripts} {empty}

Pres-script = msub|msup|msubsup|munder|mover|munderover|mmultiscripts
linebreak-values = "auto" | "newline" | "indentingnewline" | "nobreak" | "goodbreak" | "badbreak
mspace = element {mspace} {MathML.Common.attrib,
                         attribute width {length-with-unit | named-space}?,
             attribute height {length-with-unit}?,
             attribute depth {length-with-unit}?,
                     attribute linebreak {linebreak-values}?}

mrow = element {mrow} {MathML.Common.attrib,ContInPres*}
```

```
mfrac = element {mfrac} {MathML.Common.attrib,
                       attribute bevelled {truefalse}?,
                       attribute denomalign {centering.values}?,
       attribute numalign {centering.values}?,
       linethickness.attrib?,
       ContInPres,ContInPres}
msqrt = element {msqrt} {MathML.Common.attrib,ContInPres*}

mroot = element {mroot} {MathML.Common.attrib,ContInPres,ContInPres}

mpadded-space = xsd:string {pattern="(\+|-)?([0-9]+|[0-9]*\.[0-9]+)(((%?)*(width|lspace|height|c



mpadded-width-space = xsd:string {pattern="((\+|-)?([0-9]+|[0-9]*\.[0-9]+)(((%?) *(width|lspace|

mpadded = element {mpadded} {MathML.Common.attrib,
                   attribute width {mpadded-width-space}?,
      attribute lspace {mpadded-space}?,
      attribute height {mpadded-space}?,
      attribute depth {mpadded-space}?,
      ContInPres*}

mphantom = element {mphantom}.attlist {MathML.Common.attrib,ContInPres*}

mfenced = element {mfenced} {MathML.Common.attrib,
                           attribute open {xsd:string}?,
                      attribute close {xsd:string}?,
      attribute separators {xsd:string}?,
   ContInPres*}

notation-values = "actuarial"|"longdiv"|"radical"|
                              "box"|"roundedbox"|"circle"|
                              "left"|"right"|"top"|"bottom"|
                              "updiagonalstrike"|"downdiagonalstrike"|
                              "verticalstrike"|"horizontalstrike"
menclose = element {menclose} {MathML.Common.attrib,
                          attribute notation {notation-values}?,
  ContInPres*}

# And the group of everything
Pres-layout = mrow|mfrac|msqrt|mroot|mpadded|mphantom|mfenced|menclose

Table-alignment.attrib = attribute rowalign
       {xsd:string {pattern="(top|bottom|center|baseline|axis)(top|bottom|center|baseline|axis)*
        attribute columnalign {xsd:string {pattern="(left|center|right)( (left|center|right))*"}
        attribute groupalign {xsd:string}?

mtr.content = mtd
```

```
mtr = element {mtr} {Table-alignment.attrib, MathML.Common.attrib,(mtr.content)+}

mlabeledtr = element {mlabeledtr} {Table-alignment.attrib,MathML.Common.attrib,(mtr.content)*}

mtd = element {mtd} {MathML.Common.attrib,
                    Table-alignment.attrib,
                    attribute columnspan {xsd:positiveInteger}?,
      attribute rowspan {xsd:positiveInteger}?,
   ContInPres*}

mtable.content = mtr|mlabeledtr
mtable = element {mtable} {Table-alignment.attrib,
                          attribute align {xsd:string}?,
 attribute alignmentscope {xsd:string {pattern="(true|false)( true| false)*"}}?,
 attribute columnwidth {xsd:string}?,
    attribute width {xsd:string}?,
    attribute rowspacing {xsd:string}?,
    attribute columnspacing {xsd:string}?,
    attribute rowlines {xsd:string}?,
    attribute columnlines {xsd:string}?,
    attribute frame {"none" | "solid" | "dashed"}?,
    attribute framespacing {xsd:string}?,
    attribute equalrows {truefalse}?,
    attribute equalcolumns {truefalse}?,
    attribute displaystyle {truefalse}?,
 attribute side {"left"|"right"|"leftoverlap"|"rightoverlap"}?,
    attribute minlabelspacing {length-with-unit}?,
    MathML.Common.attrib,
 (mtable.content)*}

malignmgroup = element {maligngroup} {MathML.Common.attrib,
     attribute groupalign {"left" | "center" | "right" | "decimalpoint"}?}

malignmark = element {malignmark} {MathML.Common.attrib,attribute edge {"left" | "right"}?}

Pres-table = mtable|maligngroup|malignmark

mcolumn = element {mcolumn} {MathML.Common.attrib,
     attribute align {"left" | "right"}}

mstyle = element {mstyle} {MathML.Common.attrib,
                          attribute scriptlevel {xsd:integer}?,
                          attribute displaystyle {truefalse}?,
 attribute scriptsizemultiplier {xsd:decimal}?,
    attribute scriptminsize {length-with-unit}?,
    attribute color {xsd:string}?,
    attribute background {xsd:string}?,
    attribute veryverythinmathspace {length-with-unit}?,
    attribute verythinmathspace {length-with-unit}?,
 attribute thinmathspace {length-with-unit}?,
```

```
                            attribute mediummathspace {length-with-unit}?,
                            attribute thickmathspace {length-with-unit}?,
                            attribute verythickmathspace {length-with-unit}?,
                            attribute veryverythickmathspace {length-with-unit}?,
                            linethickness.attrib?,
     Operator.attrib,Token-style.attrib,
  ContInPres*}

merror = element {merror} {MathML.Common.attrib,ContInPres*}

maction = element {maction} {MathML.Common.attrib,
   attribute actiontype {xsd:string}?,
                      attribute selection {xsd:positiveInteger}?,
      ContInPres*}

semantics-pmml = element {semantics} {semantics.attribs,PresExp, semantics-annotation*}

PresExp = Pres-token | Pres-layout | Pres-script | Pres-table
      |  mspace | mline | mcolumn |  maction | merror | mstyle
      | semantics-pmml

ContInPres = PresExp
```

Issue (): David wrote in an e-mail: `length-with-unit` doesn't allow white space (anywhere) which (if any) of the following do we want to allow " 2em ", "2 em", "- 2 em". Also it insists on starting with a digit or -, but do we want to allow ".5em" "-.5em"However we do claim css compatibility here which may suggest some answers to the above `http://www.w3.org/TR/CSS21/syndata.html#length-units`.css allows an optional leading + as well `+2em` css requires number to "immediately" follow any sign and the unit to "immediately" follow the number, which I think means no intervening white space. css <number> are allowed to start with a `.` so `.5em` is allowed. css insists on a digit following a `.` so `5.em` is not allowed.Once we have firm answers to the above it should be easy to drop the regexp back in, and make the text match.I think we should not allow white space except at beginning and end but allow a leading + (a change from mathml2) and allow no digits before the `.`, but insist on digits after a `.` which would be `[\-\+]?([0-9]+(\.[0-9]+)?|\.[0-9]+)(em|ex|px|in|cm|mm|pt|pc|%))|0` as written this doesn't allow `"  2em  "` but I think we can set white space trim properties to apply before the regex is checked (I'll check)

A.2.2 The Grammar for Strict Content MathML3

The grammar for Strict Content MathML3 can be found at http://www.w3.org/Math/RelaxNG/mathml3/mathml3-strict.rnc.

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2007 World Wide Web Consortium
#        (Massachusetts Institute of Technology, Institut National de
#         Recherche en Informatique et en Automatique, Keio University).
#         All Rights Reserved.
```

```
#
#     Permission to use, copy, modify and distribute the RelaxNG schema for MathML3
#     and its accompanying documentation for any purpose and without fee is
#     hereby granted in perpetuity, provided that the above copyright notice
#     and this paragraph appear in all copies.  The copyright holders make
#     no representation about the suitability of the Schema for any purpose.
#
#     This file contains the grammar rules for strict content MathML3
#     It is provided "as is" without expressed or implied warranty.
#
#     Revision:   $Id: mathml3-strict.rnc,v 1.1 2008/03/27 13:35:25 mkohlhas2 Exp $
#     Author: Michael Kohlhase http://kwarc.info/kohlhase

#  This is the RelaxNG schema module for the strict content part of MathML.

default namespace m = "http://www.w3.org/1998/Math/MathML"

include "mathml3-common.rnc"

math.content |= ContExp


opel.content = text

# we want to extend this in pragmatic CMathML, so we introduce abbrevs here.

cn.content = text
cn.type.vals  = "e-notation"|"integer"|"rational"|"real" |
                         "complex-cartesian"|"complex-polar"


cn = element {cn} {#attribute base {xsd:positiveInteger [1,...,36]},
                 attribute type {cn.type.vals}?,
    Definition.attrib,
    MathML.Common.attrib,
 (cn.content)*}

ci = element {ci} {attribute type {xsd:string}?,
                 attribute nargs {xsd:string}?,
 attribute occurrence {xsd:string}?,
                 Definition.attrib,
    MathML.Common.attrib,
 opel.content,
 name.attrib?}

cdname.attrib = attribute cd {xsd:NCName}

csymbol       = element {csymbol} {MathML.Common.attrib,
                         Definition.attrib,cdname.attrib?,cdbase.attrib?,
 opel.content}
```

```
# the content of the apply element, leave it empty and extend it later
apply = element {apply} {MathML.Common.attrib,cdbase.attrib?,apply.content}
apply-head = apply|bind|ci|csymbol|semantics-apply
apply.content = apply-head,ContExp*
semantics-apply = element {semantics} {semantics.attribs,apply-head, semantics-annotation*}

qualifier = condition

# the content of the bind element, leave it empty and extend it later
bind = element {bind} {MathML.Common.attrib,cdbase.attrib?,bind.content}
bind-head = apply|csymbol|semantics-bind
bind.content = bind-head,bvar*,qualifier?,ContExp
semantics-bind   = element {semantics} {semantics.attribs,bind-head, semantics-annotation*}

bvar = element {bvar} {MathML.Common.attrib,cdbase.attrib?,bvar-head}
bvar-head = ci|semantics-bvar
semantics-bvar   = element {semantics} {semantics.attribs,bvar-head, semantics-annotation*}

condition = element {condition} {Definition.attrib,cdbase.attrib?,ContExp}

share = element {share} {MathML.Common.attrib,attribute href {xsd:anyURI}}

# the content of the cerror element, leave it empty and extend it later
cerror = element {cerror} {MathML.Common.attrib,cdbase.attrib?,cerror.content}
cerror-head = csymbol|apply|semantics-cerror
cerror.content = cerror-head,ContExp*
semantics-cerror = element {semantics} {semantics.attribs,cerror-head, semantics-annotation*}

semantics-cmml = element {semantics} {semantics.attribs,ContExp, semantics-annotation*}

ContExp = cn| ci | csymbol | apply | bind | share | cerror | semantics-cmml
```

A.2.3 The Grammar for Pragmatic MathML

The grammar for pragmatic MathML3 can be found at http://www.w3.org/Math/RelaxNG/mathml3/mathml3-pragmatic.rnc.

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2007 World Wide Web Consortium
#        (Massachusetts Institute of Technology, Institut National de
#         Recherche en Informatique et en Automatique, Keio University).
#         All Rights Reserved.
#
#     Permission to use, copy, modify and distribute the RelaxNG schema for MathML3
```

```
#     and its accompanying documentation for any purpose and without fee is
#     hereby granted in perpetuity, provided that the above copyright notice
#     and this paragraph appear in all copies.  The copyright holders make
#     no representation about the suitability of the Schema for any purpose.
#
#     This file contains the grammar rules for pragmatic content MathML3
#     It is provided "as is" without expressed or implied warranty.
#
#     Revision:   $Id: mathml3-pragmatic.rnc,v 1.1 2008/03/27 13:35:25 mkohlhas2 Exp $
#     Author: Michael Kohlhase http://kwarc.info/kohlhase
#
#     This is the RelaxNG schema module for the pragmatic content part of
#     MathML (but without the presentation in token elements).

default namespace m = "http://www.w3.org/1998/Math/MathML"

include "mathml3-strict.rnc"

## the content of "cn" may have <sep> elements in it
sep = element {sep} {empty}
cn.content |= sep
cn.type.vals |= "constant"

# allow degree in bvar
degree = element {degree} {MathML.Common.attrib,ContExp+}
bvar-head |= (degree?,ci)|(ci,degree?)
# allow degree to modify <root/>
apply.content |= root_arith1_elt,degree,ContExp*


domainofapplication = element {domainofapplication} {Definition.attrib,MathML.Common.attrib,cdba

lowlimit = element {lowlimit} {Definition.attrib,MathML.Common.attrib,cdbase.attrib?,ContExp+}
uplimit = element {uplimit} {Definition.attrib,MathML.Common.attrib,cdbase.attrib?,ContExp+}

## allow the non-strict qualifiers
qualifier |= domainofapplication|(uplimit,lowlimit?)|(lowlimit,uplimit?)|degree

## we collect the operator elements by role
opel.constant = notAllowed
opel.binder = notAllowed
opel.application = notAllowed
opel.semantic-attribution = notAllowed
opel.attribution = notAllowed
opel.error = notAllowed

opels = opel.constant | opel.binder | opel.application |
        opel.semantic-attribution | opel.attribution |
opel.error
container = notAllowed
```

```
## the values of the MathML type attributes;
MathMLType |= "real" | "complex" | "function" | "algebraic" | "integer"

## include the relevant content dictionaries
include "mathml3-cds-pragmatic.rnc"

## we instantiate the strict content model by structure checking
apply-binder-head = semantics-apply-binder|opel.binder
apply.content |= apply-binder-head,bvar+,qualifier?,ContExp
semantics-apply-binder = element {semantics} {semantics.attribs,apply-binder-head, semantics-ann

apply-head |= opel.application
bind-head |= opel.binder
cerror-head |= opel.error

## allow all functions, constants, and containers to be content expressions on their own
ContExp |= opel.constant|opel.application|container

#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2007 World Wide Web Consortium
#        (Massachusetts Institute of Technology, Institut National de
#         Recherche en Informatique et en Automatique, Keio University).
#         All Rights Reserved.
#
#     Permission to use, copy, modify and distribute the RelaxNG schema for MathML3
#     and its accompanying documentation for any purpose and without fee is
#     hereby granted in perpetuity, provided that the above copyright notice
#     and this paragraph appear in all copies.  The copyright holders make
#     no representation about the suitability of the Schema for any purpose.
#
#     This file contains the grammar rules for pragmatic content MathML3
#     It is provided "as is" without expressed or implied warranty.
#
#     Revision:   $Id: mathml3-pragmatic.rnc,v 1.1 2008/03/27 13:35:25 mkohlhas2 Exp $
#     Author: Michael Kohlhase http://kwarc.info/kohlhase
#
#     This is the RelaxNG schema module for the pragmatic content part of
#     MathML (but without the presentation in token elements).

default namespace m = "http://www.w3.org/1998/Math/MathML"

include "mathml3-strict.rnc"

## the content of "cn" may have <sep> elements in it
sep = element {sep} {empty}
cn.content |= sep
```

```
cn.type.vals |= "constant"

# allow degree in bvar
degree = element {degree} {MathML.Common.attrib,ContExp+}
bvar-head |= (degree?,ci)|(ci,degree?)
# allow degree to modify <root/>
apply.content |= root_arith1_elt,degree,ContExp*


domainofapplication = element {domainofapplication} {Definition.attrib,MathML.Common.attrib,cdba

lowlimit = element {lowlimit} {Definition.attrib,MathML.Common.attrib,cdbase.attrib?,ContExp+}
uplimit = element {uplimit} {Definition.attrib,MathML.Common.attrib,cdbase.attrib?,ContExp+}

## allow the non-strict qualifiers
qualifier |= domainofapplication|(uplimit,lowlimit?)|(lowlimit,uplimit?)|degree

## we collect the operator elements by role
opel.constant = notAllowed
opel.binder = notAllowed
opel.application = notAllowed
opel.semantic-attribution = notAllowed
opel.attribution = notAllowed
opel.error = notAllowed

opels = opel.constant | opel.binder | opel.application |
        opel.semantic-attribution | opel.attribution |
opel.error
container = notAllowed

## the values of the MathML type attributes;
MathMLType |= "real" | "complex" | "function" | "algebraic" | "integer"

## include the relevant content dictionaries
include "mathml3-cds-pragmatic.rnc"

## we instantiate the strict content model by structure checking
apply-binder-head = semantics-apply-binder|opel.binder
apply.content |= apply-binder-head,bvar+,qualifier?,ContExp
semantics-apply-binder = element {semantics} {semantics.attribs,apply-binder-head, semantics-ann

apply-head |= opel.application
bind-head |= opel.binder
cerror-head |= opel.error

## allow all functions, constants, and containers to be content expressions on their own
ContExp |= opel.constant|opel.application|container
```

This grammar focuses on the pragmatic extensions in , , , , and .

Editor's note: MiKocheck this again

The pragmatic extensions in , , , , , rely on information that is specified in the MathML content dictionaries. This is handled in the schema http://www.w3.org/Math/RelaxNG/mathml3/mathml3-cds-pragmatic.rnc.

Finally, the pragmatic extensions given in are not covered in this schema, but will be left for full MathML in the next section.

A.2.4 Full MathML

The RelaxNG schema for full MathML without deprecated functionality builds on the schemata for presentation MathML in and pragmatic Content MathML in , mixing the content models as described in . 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 1998-2007 World Wide Web Consortium
#        (Massachusetts Institute of Technology, Institut National de
#         Recherche en Informatique et en Automatique, Keio University).
#         All Rights Reserved.
#
#     Permission to use, copy, modify and distribute the RelaxNG schema for MathML3
#     and its accompanying documentation for any purpose and without fee is
#     hereby granted in perpetuity, provided that the above copyright notice
#     and this paragraph appear in all copies.  The copyright holders make
#     no representation about the suitability of the Schema for any purpose.
#
#     This file contains the grammar driver for MathML3
#     It is provided "as is" without expressed or implied warranty.
#
#     Revision:   $Id: mathml3.rnc,v 1.1 2008/03/27 13:35:26 mkohlhas2 Exp $
#     Author: Michael Kohlhase http://kwarc.info/kohlhase

default namespace m = "http://www.w3.org/1998/Math/MathML"

include "mathml3-common.rnc"

## Content Expressions now allow pMathML in ci and csymbol
ContExp = grammar {include "mathml3-pragmatic.rnc" {start=ContExp opel.content = text|parent Pre

## Presentation Expressions allow Content Expressions mixed in everywhere
PresExp = grammar {include "mathml3-presentation.rnc" {start=PresExp ContInPres=PresExp|parent C

## the math element can contain one content element or several presentation elements
math.content|=ContExp|PresExp+
```

A.2.5 Full MathML with Deprecated Elements

The grammar for the elements deprecated in MathML3 can be found at `http://www.w3.org/Math/RelaxNG/mathml3/mathml3-deprecated.rnc`.

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2007 World Wide Web Consortium
#        (Massachusetts Institute of Technology, Institut National de
#         Recherche en Informatique et en Automatique, Keio University).
#         All Rights Reserved.
#
#     Permission to use, copy, modify and distribute the RelaxNG schema for MathML3
#     and its accompanying documentation for any purpose and without fee is
#     hereby granted in perpetuity, provided that the above copyright notice
#     and this paragraph appear in all copies.  The copyright holders make
#     no representation about the suitability of the Schema for any purpose.
#
#     This file contains the grammar driver for MathML3
#     It is provided "as is" without expressed or implied warranty.
#
#     Revision:   $Id: mathml3-deprecated.rnc,v 1.1 2008/03/27 17:39:14 dcarlis Exp $
#     Author: Michael Kohlhase http://kwarc.info/kohlhase

default namespace m = "http://www.w3.org/1998/Math/MathML"

include "mathml3.rnc"

Token-style.attrib |=
  attribute fontsize {xsd:string}? |
  attribute fontstyle {xsd:string}? |
  attribute color {xsd:string}? |
  attribute fontfamily {xsd:string}? |
  attribute fontweight {xsd:string}?

#Deprecated Content Elements
dep-content =
  element {reln} {ContExp*}|
  element {fn} {ContExp}

ContExp |= dep-content

apply-head |= dep-content

declare = element {declare} {attribute type {xsd:string}?,
                             attribute scope {xsd:string}?,
                             attribute nargs {xsd:nonNegativeInteger}?,
                             attribute occurrence {"prefix"|"infix"|"function-model"}?,
                             Definition.attrib,cdbase.attrib?,
```

```
                           ContExp+}
ContExp |= declare
```

A.2.6 Generated Grammar for Arity & Type Checking

In Section A.2.3 we have seen an example of a grammar that is generated from information present in the MathML content dictionaries. If we make use of the type information that comes with the CDs.

Editor's note: MiKomaybe we should have a note about generating Grammars from the CDs. I will have to generate arity and type checking files to mix in and import them here. Maybe this should not be treated in a normative appendix?

A.2.7 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 simple as

```
# A RelaxNG Schema for  XHTML+MathML
include "xhtml.rnc"
math = external "mathml3.rnc"
Inline.class |= math
Block.class |= math
```

assuming that we have access to a modular RelaxNG schema for xhtml that uses `Inline.class` and `Block.class` to collect the the content models for inline and block-level elements.

Editor's note: Mikocheck this and reference an external schema

Specilizing the MathML3 schema so that we can check the content of `annotation-xml` elements is similarly simple:

```
# A RelaxNG Schema for MathML with OpenMath3 annotations
omobj = external "openmath3.rnc"
include "mathml3.rnc" {anotation-xml.model = omobj}
```

For details about RelaxNG grammars and modularization see [RelaxNG] or [RelaxNGBook].

Editor's note: Mikocheck this and reference an external schema; I think we can even tie the OpenMath model to the value `OpenMath` in the `encoding` attribute.

A.3 Using the MathML DTD

Editor's note: DavidDTD to be generated from Relax NG

A.4 Using the MathML XML Schema

Editor's note: DavidXSD schema to be generated from Relax NG

Appendix B

Operator Dictionary (Non-Normative)

Issue (): The current appendix describes MathML2, it may need to be updated in later drafts.

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.

B.1 Format of operator dictionary entries

The operators are divided into groups, which are separated by blank lines in the listing below. The grouping, and the order of the groups, is significant for the proper grouping of sub-expressions using <mrow> (Section 3.3.1); 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 format of the table entries is: the <mo> element content between double quotes (start and end tags not shown), followed by the attribute list in XML format, starting with the form attribute, followed by the default rendering attributes which should be used for mo elements with the given content and form attribute.

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.

Note that the characters & and < are represented in the following table entries by the entity references & and < respectively, as would be necessary if they appeared in the content of an actual mo element (or any other MathML or XML element).

For example, the first entry,

```
"(" form="prefix" fence="true" stretchy="true" lspace="0em" rspace="0em"
```

could be expressed as an mo element by:

```
<mo form="prefix" fence="true" stretchy="true" lspace="0em" rspace="0em"> ( </mo>
```

(note the lack of double quotes around the content, and the whitespace added around the content for readability, which is optional in MathML).

This entry means that, for MathML renderers which use this suggested operator dictionary, giving the element <mo form="prefix"> (</mo> alone, or simply <mo> (</mo> in a position for which form="prefix" would be inferred (see below), is equivalent to giving the element with all attributes as shown above.

B.2 Indexing of operator dictionary

Note that the dictionary is indexed not just by the element content, but by the element content and `form` attribute value, together. Operators with more than one possible form have more than one entry. The MathML specification describes how the renderer chooses ('infers') which form to use when no `form` attribute is given; see Section 3.2.5.7.

Having made that choice, or with the `form` attribute explicitly specified in the `<mo>` element's start tag, the MathML renderer uses the remaining attributes from the dictionary entry for the appropriate single form of that operator, ignoring the entries for the other possible forms.

B.3 Choice of entity names

Extended characters in MathML (and in the operator dictionary below) are represented by XML-style entity references using the syntax `&character-name;` the complete list of characters and character names is given in Chapter 6. Many characters can be referred to by more than one name; often, memorable names composed of full words have been provided in MathML, as well as one or more names used in other standards, such as Unicode. The characters in the operators in this dictionary are generally listed under their full-word names when these exist. For example, the integral operator is named below by the one-character sequence `∫`, but could equally well be named `∫`. The choice of name for a given character in MathML has no effect on its rendering.

It is intended that every entity named below appears somewhere in Chapter 6. If this is not true, it is an error in this specification. If such an error exists, the abovementioned chapter should be taken as definitive, rather than this appendix.

B.4 Notes on `lspace` and `rspace` attributes

The values for `lspace` and `rspace` given here range from 0 to `"verythickmathspace"`, which has a default value of 6/18 em. For the invisible operators whose content is `⁢` or `⁡`, 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>⁡</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>⁢</mo>`, the total spacing should be greater than zero when both operands (or the nearest tokens on either side, if on the baseline) are identifiers displayed in a non-slanted font (i.e. under the suggested rules, when both operands are multi-character identifiers).

Some renderers may wish to use no spacing for most operators appearing in scripts (i.e. when `scriptlevel` is greater than 0; see Section 3.3.4), as is the case in TeX.

B.5 Operator dictionary entries

```
"("                           form="prefix"  fence="true" stretchy="true"  lspace="0em" rspace="0em"
")"                           form="postfix" fence="true" stretchy="true"  lspace="0em" rspace="0em"
"["                           form="prefix"  fence="true" stretchy="true"  lspace="0em" rspace="0em"
"]"                           form="postfix" fence="true" stretchy="true"  lspace="0em" rspace="0em"
"{"                           form="prefix"  fence="true" stretchy="true"  lspace="0em" rspace="0em"
"}"                           form="postfix" fence="true" stretchy="true"  lspace="0em" rspace="0em"

"&CloseCurlyDoubleQuote;"     form="postfix" fence="true"  lspace="0em" rspace="0em"
"&CloseCurlyQuote;"           form="postfix" fence="true"  lspace="0em" rspace="0em"
"&LeftAngleBracket;"          form="prefix"  fence="true" stretchy="true"  lspace="0em" rspace="0em""
"&LeftCeiling;"               form="prefix"  fence="true" stretchy="true"  lspace="0em" rspace="0em"
"&LeftDoubleBracket;"         form="prefix"  fence="true" stretchy="true"  lspace="0em" rspace="0em"
"&LeftFloor;"                 form="prefix"  fence="true" stretchy="true"  lspace="0em" rspace="0em"
"&OpenCurlyDoubleQuote;"      form="prefix"  fence="true"  lspace="0em" rspace="0em"
"&OpenCurlyQuote;"            form="prefix"  fence="true"  lspace="0em" rspace="0em"
"&RightAngleBracket;"         form="postfix" fence="true" stretchy="true"  lspace="0em" rspace="0em"
"&RightCeiling;"              form="postfix" fence="true" stretchy="true"  lspace="0em" rspace="0em"
"&RightDoubleBracket;"        form="postfix" fence="true" stretchy="true"  lspace="0em" rspace="0em"
"&RightFloor;"                form="postfix" fence="true" stretchy="true"  lspace="0em" rspace="0em"

"&InvisibleComma;"            form="infix"   separator="true"  lspace="0em" rspace="0em"

","                           form="infix"   separator="true"  lspace="0em" rspace="verythickmathspace"

"&HorizontalLine;"            form="infix"   stretchy="true" minsize="0"  lspace="0em" rspace="0em"
"&VerticalLine;"              form="infix"   stretchy="true" minsize="0"  lspace="0em" rspace="0em"

";"                           form="infix"   separator="true"  lspace="0em" rspace="thickmathspace"
";"                           form="postfix" separator="true"  lspace="0em" rspace="0em"

":="                          form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&Assign;"                    form="infix"    lspace="thickmathspace" rspace="thickmathspace"

"&Because;"                   form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&Therefore;"                 form="infix"    lspace="thickmathspace" rspace="thickmathspace"

"&VerticalSeparator;"         form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"

"//"                          form="infix"    lspace="thickmathspace" rspace="thickmathspace"
```

```
"&Colon;"             form="infix"    lspace="thickmathspace" rspace="thickmathspace"

"&amp;"               form="prefix"   lspace="0em" rspace="thickmathspace"
"&amp;"               form="postfix"  lspace="thickmathspace" rspace="0em"

"*="                  form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"-="                  form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"+="                  form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"/="                  form="infix"    lspace="thickmathspace" rspace="thickmathspace"

"->"                  form="infix"    lspace="thickmathspace" rspace="thickmathspace"

":"                   form="infix"    lspace="thickmathspace" rspace="thickmathspace"

".."                  form="postfix"  lspace="mediummathspace" rspace="0em"
"..."                 form="postfix"  lspace="mediummathspace" rspace="0em"

"&SuchThat;"          form="infix"    lspace="thickmathspace" rspace="thickmathspace"

"&DoubleLeftTee;"     form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&DoubleRightTee;"    form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&DownTee;"           form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&LeftTee;"           form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&RightTee;"          form="infix"    lspace="thickmathspace" rspace="thickmathspace"

"&Implies;"           form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&RoundImplies;"      form="infix"    lspace="thickmathspace" rspace="thickmathspace"

"|"                   form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"||"                  form="infix"    lspace="mediummathspace" rspace="mediummathspace"
"&Or;"                form="infix"   stretchy="true"  lspace="mediummathspace" rspace="mediummathspace"

"&amp;&amp;"          form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&And;"               form="infix"   stretchy="true"  lspace="mediummathspace" rspace="mediummathspace"

"&amp;"               form="infix"    lspace="thickmathspace" rspace="thickmathspace"

"!"                   form="prefix"   lspace="0em" rspace="thickmathspace"
"&Not;"               form="prefix"   lspace="0em" rspace="thickmathspace"

"&Exists;"            form="prefix"   lspace="0em" rspace="thickmathspace"
```

```
"&ForAll;"                   form="prefix"   lspace="0em" rspace="thickmathspace"
"&NotExists;"                form="prefix"   lspace="0em" rspace="thickmathspace"

"&Element;"                  form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotElement;"               form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotReverseElement;"        form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotSquareSubset;"          form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotSquareSubsetEqual;"     form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotSquareSuperset;"        form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotSquareSupersetEqual;"   form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotSubset;"                form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotSubsetEqual;"           form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotSuperset;"              form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&NotSupersetEqual;"         form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&ReverseElement;"           form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&SquareSubset;"             form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&SquareSubsetEqual;"        form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&SquareSuperset;"           form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&SquareSupersetEqual;"      form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&Subset;"                   form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&SubsetEqual;"              form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&Superset;"                 form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&SupersetEqual;"            form="infix"    lspace="thickmathspace" rspace="thickmathspace"

"&DoubleLeftArrow;"          form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&DoubleLeftRightArrow;"     form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&DoubleRightArrow;"         form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&DownLeftRightVector;"      form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&DownLeftTeeVector;"        form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&DownLeftVector;"           form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&DownLeftVectorBar;"        form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&DownRightTeeVector;"       form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&DownRightVector;"          form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&DownRightVectorBar;"       form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LeftArrow;"                form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LeftArrowBar;"             form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LeftArrowRightArrow;"      form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LeftRightArrow;"           form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LeftRightVector;"          form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LeftTeeArrow;"             form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LeftTeeVector;"            form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
```

```
"&LeftVector;"              form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LeftVectorBar;"           form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LowerLeftArrow;"          form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&LowerRightArrow;"         form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&RightArrow;"              form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&RightArrowBar;"           form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&RightArrowLeftArrow;"     form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&RightTeeArrow;"           form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&RightTeeVector;"          form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&RightVector;"             form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&RightVectorBar;"          form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&ShortLeftArrow;"          form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&ShortRightArrow;"         form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&UpperLeftArrow;"          form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&UpperRightArrow;"         form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"

"="                         form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&lt;"                      form="infix"    lspace="thickmathspace" rspace="thickmathspace"
">"                         form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"!="                        form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"=="                        form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&lt;="                     form="infix"    lspace="thickmathspace" rspace="thickmathspace"
">="                        form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&Congruent;"               form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&CupCap;"                  form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&DotEqual;"                form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&DoubleVerticalBar;"       form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&Equal;"                   form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&EqualTilde;"              form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&Equilibrium;"             form="infix"   stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&GreaterEqual;"            form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&GreaterEqualLess;"        form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&GreaterFullEqual;"        form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&GreaterGreater;"          form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&GreaterLess;"             form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&GreaterSlantEqual;"       form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&GreaterTilde;"            form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&HumpDownHump;"            form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&HumpEqual;"               form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&LeftTriangle;"            form="infix"    lspace="thickmathspace" rspace="thickmathspace"
"&LeftTriangleBar;"         form="infix"    lspace="thickmathspace" rspace="thickmathspace"
```

```
"&LeftTriangleEqual;"          form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&le;"                         form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&LessEqualGreater;"           form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&LessFullEqual;"              form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&LessGreater;"                form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&LessLess;"                   form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&LessSlantEqual;"             form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&LessTilde;"                  form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NestedGreaterGreater;"       form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NestedLessLess;"             form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotCongruent;"               form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotCupCap;"                  form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotDoubleVerticalBar;"       form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotEqual;"                   form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotEqualTilde;"              form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotGreater;"                 form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotGreaterEqual;"            form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotGreaterFullEqual;"        form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotGreaterGreater;"          form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotGreaterLess;"             form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotGreaterSlantEqual;"       form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotGreaterTilde;"            form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotHumpDownHump;"            form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotHumpEqual;"               form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotLeftTriangle;"            form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotLeftTriangleBar;"         form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotLeftTriangleEqual;"       form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotLess;"                    form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotLessEqual;"               form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotLessGreater;"             form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotLessLess;"                form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotLessSlantEqual;"          form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotLessTilde;"               form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotNestedGreaterGreater;"    form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotNestedLessLess;"          form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotPrecedes;"                form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotPrecedesEqual;"           form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotPrecedesSlantEqual;"      form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotRightTriangle;"           form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotRightTriangleBar;"        form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotRightTriangleEqual;"      form="infix"   lspace="thickmathspace" rspace="thickmathspace"
```

```
"&NotSucceeds;"                form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotSucceedsEqual;"           form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotSucceedsSlantEqual;"      form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotSucceedsTilde;"           form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotTilde;"                   form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotTildeEqual;"              form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotTildeFullEqual;"          form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotTildeTilde;"              form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&NotVerticalBar;"             form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&Precedes;"                   form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&PrecedesEqual;"              form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&PrecedesSlantEqual;"         form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&PrecedesTilde;"              form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&Proportion;"                 form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&Proportional;"               form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&ReverseEquilibrium;"         form="infix"  stretchy="true"  lspace="thickmathspace" rspace="thickmathspace"
"&RightTriangle;"              form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&RightTriangleBar;"           form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&RightTriangleEqual;"         form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&Succeeds;"                   form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&SucceedsEqual;"              form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&SucceedsSlantEqual;"         form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&SucceedsTilde;"              form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&Tilde;"                      form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&TildeEqual;"                 form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&TildeFullEqual;"             form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&TildeTilde;"                 form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&UpTee;"                      form="infix"   lspace="thickmathspace" rspace="thickmathspace"
"&VerticalBar;"                form="infix"   lspace="thickmathspace" rspace="thickmathspace"

"&SquareUnion;"                form="infix"  stretchy="true"  lspace="mediummathspace" rspace="mediummathspace"
"&Union;"                      form="infix"  stretchy="true"  lspace="mediummathspace" rspace="mediummathspace"
"&UnionPlus;"                  form="infix"  stretchy="true"  lspace="mediummathspace" rspace="mediummathspace"

"-"                            form="infix"   lspace="mediummathspace" rspace="mediummathspace"
"+"                            form="infix"   lspace="mediummathspace" rspace="mediummathspace"
"&Intersection;"               form="infix"  stretchy="true"  lspace="mediummathspace" rspace="mediummathspace"
"&MinusPlus;"                  form="infix"   lspace="mediummathspace" rspace="mediummathspace"
"&PlusMinus;"                  form="infix"   lspace="mediummathspace" rspace="mediummathspace"
"&SquareIntersection;"         form="infix"  stretchy="true"  lspace="mediummathspace" rspace="mediummathspace"
```

```
"&Vee;"                              form="prefix"  largeop="true" movablelimits="true" stretchy="true"  lspace="0em" rspace="thinmathspace"
"&CircleMinus;"                      form="prefix"  largeop="true" movablelimits="true"  lspace="0em" rspace="thinmathspace"
"&CirclePlus;"                       form="prefix"  largeop="true" movablelimits="true"  lspace="0em" rspace="thinmathspace"
"&Sum;"                              form="prefix"  largeop="true" movablelimits="true" stretchy="true"  lspace="0em" rspace="thinmathspace"
"&Union;"                            form="prefix"  largeop="true" movablelimits="true" stretchy="true"  lspace="0em" rspace="thinmathspace"
"&UnionPlus;"                        form="prefix"  largeop="true" movablelimits="true" stretchy="true"  lspace="0em" rspace="thinmathspace"
"lim"                                form="prefix"  movablelimits="true"  lspace="0em" rspace="thinmathspace"
"max"                                form="prefix"  movablelimits="true"  lspace="0em" rspace="thinmathspace"
"min"                                form="prefix"  movablelimits="true"  lspace="0em" rspace="thinmathspace"

"&CircleMinus;"                      form="infix"    lspace="thinmathspace" rspace="thinmathspace"
"&CirclePlus;"                       form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&ClockwiseContourIntegral;"         form="prefix"  largeop="true" stretchy="true"  lspace="0em" rspace="0em"
"&ContourIntegral;"                  form="prefix"  largeop="true" stretchy="true"  lspace="0em" rspace="0em"
"&CounterClockwiseContourIntegral;"  form="prefix"  largeop="true" stretchy="true"  lspace="0em" rspace="0em"
"&DoubleContourIntegral;"            form="prefix"  largeop="true" stretchy="true"  lspace="0em" rspace="0em"
"&Integral;"                         form="prefix"  largeop="true" stretchy="true"  lspace="0em" rspace="0em"

"&Cup;"                              form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&Cap;"                              form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&VerticalTilde;"                    form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&Wedge;"                            form="prefix"  largeop="true" movablelimits="true" stretchy="true"  lspace="0em" rspace="thinmathspace"
"&CircleTimes;"                      form="prefix"  largeop="true" movablelimits="true"  lspace="0em" rspace="thinmathspace"
"&Coproduct;"                        form="prefix"  largeop="true" movablelimits="true" stretchy="true"  lspace="0em" rspace="thinmathspace"
"&Product;"                          form="prefix"  largeop="true" movablelimits="true" stretchy="true"  lspace="0em" rspace="thinmathspace"
"&Intersection;"                     form="prefix"  largeop="true" movablelimits="true" stretchy="true"  lspace="0em" rspace="thinmathspace"

"&Coproduct;"                        form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&Star;"                             form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&CircleDot;"                        form="prefix"  largeop="true" movablelimits="true"  lspace="0em" rspace="thinmathspace"

"*"                                  form="infix"    lspace="thinmathspace" rspace="thinmathspace"
"&InvisibleTimes;"                   form="infix"    lspace="0em" rspace="0em"

"&CenterDot;"                        form="infix"    lspace="thinmathspace" rspace="thinmathspace"
```

```
"&CircleTimes;"                  form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&Vee;"                          form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&Wedge;"                        form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&Diamond;"                      form="infix"    lspace="thinmathspace" rspace="thinmathspace"

"&Backslash;"                    form="infix"   stretchy="true"  lspace="thinmathspace" rspace="thinmathspace"

"/"                              form="infix"   stretchy="true"  lspace="thinmathspace" rspace="thinmathspace"

"-"                              form="prefix"   lspace="0em" rspace="veryverythinmathspace"
"+"                              form="prefix"   lspace="0em" rspace="veryverythinmathspace"
"&MinusPlus;"                    form="prefix"   lspace="0em" rspace="veryverythinmathspace"
"&PlusMinus;"                    form="prefix"   lspace="0em" rspace="veryverythinmathspace"

"."                              form="infix"    lspace="0em" rspace="0em"

"&Cross;"                        form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"**"                             form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"&CircleDot;"                    form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"&SmallCircle;"                  form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"&Square;"                       form="prefix"   lspace="0em" rspace="verythinmathspace"

"&Del;"                          form="prefix"   lspace="0em" rspace="verythinmathspace"
"&PartialD;"                     form="prefix"   lspace="0em" rspace="verythinmathspace"

"&CapitalDifferentialD;"         form="prefix"   lspace="0em" rspace="verythinmathspace"
"&DifferentialD;"                form="prefix"   lspace="0em" rspace="verythinmathspace"

"&Sqrt;"                         form="prefix"  stretchy="true"  lspace="0em" rspace="verythinmathspace"

"&DoubleDownArrow;"              form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&DoubleLongLeftArrow;"          form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&DoubleLongLeftRightArrow;"     form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
```

```
"&DoubleLongRightArrow;"     form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&DoubleUpArrow;"            form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&DoubleUpDownArrow;"        form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&DownArrow;"                form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&DownArrowBar;"             form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&DownArrowUpArrow;"         form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&DownTeeArrow;"             form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LeftDownTeeVector;"        form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LeftDownVector;"           form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LeftDownVectorBar;"        form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LeftUpDownVector;"         form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LeftUpTeeVector;"          form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LeftUpVector;"             form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LeftUpVectorBar;"          form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LongLeftArrow;"            form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LongLeftRightArrow;"       form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&LongRightArrow;"           form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&ReverseUpEquilibrium;"     form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&RightDownTeeVector;"       form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&RightDownVector;"          form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&RightDownVectorBar;"       form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&RightUpDownVector;"        form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&RightUpTeeVector;"         form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&RightUpVector;"            form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&RightUpVectorBar;"         form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&ShortDownArrow;"           form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"
"&ShortUpArrow;"             form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"
"&UpArrow;"                  form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&UpArrowBar;"               form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&UpArrowDownArrow;"         form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&UpDownArrow;"              form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&UpEquilibrium;"            form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"
"&UpTeeArrow;"               form="infix"   stretchy="true"  lspace="verythinmathspace" rspace="verythinmathspace"

"^"                          form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"&lt;>"                      form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"’"                          form="postfix"  lspace="verythinmathspace" rspace="0em"

"!"                          form="postfix"  lspace="verythinmathspace" rspace="0em"
```

```
"!!"                          form="postfix"  lspace="verythinmathspace" rspace="0em"

"~"                           form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"@"                           form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"-"                          form="postfix"  lspace="verythinmathspace" rspace="0em"
"-"                          form="prefix"   lspace="0em" rspace="verythinmathspace"
"++"                          form="postfix"  lspace="verythinmathspace" rspace="0em"
"++"                          form="prefix"   lspace="0em" rspace="verythinmathspace"

"&ApplyFunction;"             form="infix"    lspace="0em" rspace="0em"

"?"                           form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"_"                           form="infix"    lspace="verythinmathspace" rspace="verythinmathspace"

"&Breve;"                     form="postfix" accent="true"  lspace="0em" rspace="0em"
"&Cedilla;"                   form="postfix" accent="true"  lspace="0em" rspace="0em"
"&DiacriticalGrave;"          form="postfix" accent="true"  lspace="0em" rspace="0em"
"&DiacriticalDot;"            form="postfix" accent="true"  lspace="0em" rspace="0em"
"&DiacriticalDoubleAcute;"    form="postfix" accent="true"  lspace="0em" rspace="0em"
"&LeftArrow;"                 form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&LeftRightArrow;"            form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&LeftRightVector;"           form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&LeftVector;"                form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&DiacriticalAcute;"          form="postfix" accent="true"  lspace="0em" rspace="0em"
"&RightArrow;"                form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&RightVector;"               form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&DiacriticalTilde;"          form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&DoubleDot;"                 form="postfix" accent="true"  lspace="0em" rspace="0em"
"&DownBreve;"                 form="postfix" accent="true"  lspace="0em" rspace="0em"
"&Hacek;"                     form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&Hat;"                       form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&OverBar;"                   form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&OverBrace;"                 form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&OverBracket;"               form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&OverParenthesis;"           form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&TripleDot;"                 form="postfix" accent="true"  lspace="0em" rspace="0em"
"&UnderBar;"                  form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&UnderBrace;"                form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
```

```
"&UnderBracket;"                    form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
"&UnderParenthesis;"                form="postfix" accent="true" stretchy="true"  lspace="0em" rspace="0em"
```

Appendix C

Sample CSS Style Sheet for MathML (Non-Normative)

The Cascading Style Sheet sample given here is *not normative*. It is provided as a guide to illustrate the sort of CSS style sheet rules which a MathML renderer should include in its default style sheet in order to comply with both the CSS and MathML specifications. In particular, there is a need to provide rules to prevent the descent of CSS font rules into MathML expressions embedded in ambient text, and to provide support for the `mathfamily`, `mathslant`, `mathweight`, `mathsize`, `mathcolor` and `mathbackground` attributes.

We expect that implementation experience will allow us to provide a more authoritative default MathML style sheet in the future. In the interim, it is hoped that this illustrative sample will be helpful.

```
math, math[mode="inline"] {
  display: inline;
  font-family: CMSY10, CMEX10, Symbol, Times;
  font-style: normal;
}

math[mode="display"] {
  display: block;
  text-align: center;
  font-family: CMSY10, CMEX10, Symbol, Times;
  font-style: normal;
}

@media screen {  /* hide from old browsers */


/* Rules dealing with the various values of the "mathvariant" attribute: */

math *.[mathvariant="normal"] {
  font-family: "Times New Roman", Courier, Garamond, serif;
  font-weight: normal;
  font-style: normal;
}

math *.[mathvariant="bold"] {
  font-family: "Times New Roman", Courier, Garamond, serif;
  font-weight: bold;
```

```
  font-style: normal;
}

math *.[mathvariant="italic"] {
  font-family: "Times New Roman", Courier, Garamond, serif;
  font-weight: normal;
  font-style: italic;
}

math *.[mathvariant="bold-italic"] {
  font-family: "Times New Roman", Courier, Garamond, serif;
  font-weight: bold;
  font-style: italic;
}

math *.[mathvariant="double-struck"] {
  font-family: msbm;
  font-weight: normal;
  font-style: normal;
}

math *.[mathvariant="script"] {
  font-family: eusb;
  font-weight: normal;
  font-style: normal;
}

math *.[mathvariant="bold-script"] {
  font-family: eusb;
  font-weight: bold;
  font-style: normal;
}

math *.[mathvariant="fraktur"] {
  font-family: eufm;
  font-weight: normal;
  font-style: normal;
}

math *.[mathvariant="bold-fraktur"] {
  font-family: eufm;
  font-weight: bold;
  font-style: italic;
}

math *.[mathvariant="sans-serif"] {
  font-family: Arial, "Lucida Sans Unicode", Verdana, sans-serif;
  font-weight: normal;
  font-style: normal;
}
```

```
math *.[mathvariant="bold-sans-serif"] {
  font-family: Arial, "Lucida Sans Unicode", Verdana, sans-serif;
  font-weight: bold;
  font-style: normal;
}

math *.[mathvariant="sans-serif-italic"] {
  font-family: Arial, "Lucida Sans Unicode", Verdana, sans-serif;
  font-weight: normal;
  font-style: italic;
}

math *.[mathvariant="sans-serif-bold-italic"] {
  font-family: Arial, "Lucida Sans Unicode", Verdana, sans-serif;
  font-weight: bold;
  font-style: italic;
}

math *.[mathvariant="monospace"] {
  font-family: monospace
}


/* Rules dealing with "mathsize" attribute */

math *.[mathsize="small"] {
  font-size: 80%
}

math *.[mathsize="normal"] {
/*  font-size: 100%  - which is unnecessary */
}

math *.[mathsize="big"] {
  font-size:  125%
}

/*Set size values for the "base" children of script and limit schema to
  distinguish them from the script or limit children:
*/

msub>*:first-child[mathsize="big"],
msup>*:first-child[mathsize="big"],
msubsup>*:first-child[mathsize="big"],
munder>*:first-child[mathsize="big"],
mover>*:first-child[mathsize="big"],
munderover>*:first-child[mathsize="big"],
mmultiscripts>*:first-child[mathsize="big"],
mroot>*:first-child[mathsize="big"] {
```

```
  font-size: 125%
}

msub>*:first-child[mathsize="small"],
msup>*:first-child[mathsize="small"],
msubsup>*:first-child[mathsize="small"],
munder>*:first-child[mathsize="small"],
mover>*:first-child[mathsize="small"],
munderover>*:first-child[mathsize="small"],
mmultiscripts>*:first-child[mathsize="small"],
mroot>*:first-child[mathsize="small"] {
  font-size: 80%
}

msub>*:first-child,
msup>*:first-child,
msubsup>*:first-child,
munder>*:first-child,
mover>*:first-child,
munderover>*:first-child,
mmultiscripts>*:first-child,
mroot>*:first-child {
  font-size: 100%
}

/*Set size values for the other children of script and limit schema (the
  script and limit children) - include scriptlevel increment attribute?
*/

msub>*[mathsize="big"],
msup>*[mathsize="big"],
msubsup>*[mathsize="big"],
munder>*[mathsize="big"],
mover>*[mathsize="big"],
munderover>*[mathsize="big"],
mmultiscripts>*[mathsize="big"],
math[display="inline"] mfrac>*[mathsize="big"],
math *[scriptlevel="+1"][mathsize="big"] {
  font-size: 89%  /* (.71 times 1.25) */
}

msub>* [mathsize="small"],
msup>*[mathsize="small"],
msubsup>*[mathsize="small"],
munder>*[mathsize="small"],
mover>*[mathsize="small"],
munderover>*[mathsize="small"],
mmultiscripts>*[mathsize="small"],
math[display="inline"] mfrac>*[mathsize="small"],
math *[scriptlevel="+1"][mathsize="small"] {
```

```
  font-size: 57% /* (.71 times .80) */
}

msub>*,
msup>*,
msubsup>*,
munder>*,
mover>*,
munderover>*,
mmultiscripts>*,
math[display="inline"] mfrac>*,
math *[scriptlevel="+1"] {
  font-size: 71%
}

mroot>*[mathsize="big"] {
  font-size: 62%  /* (.50 times 1.25) */
}

mroot>*[mathsize="small"] {
  font-size: 40% /* (.50 times .80) */
}

mroot>* {
  font-size: 50%
}

/* Set size values for other scriptlevel increment attributes  */

math *[scriptlevel="+2"][mathsize="big"] {
  font-size: 63%  /* (.71 times .71 times 1.25) */
}

math *[scriptlevel="+2"][mathsize="small"] {
  font-size: 36% /* (.71 times .71 times .71) */
}

math *[scriptlevel="+2"] {
  font-size: 50%   /* .71 times .71 */
}

math *.[mathcolor="green"] {
  color: green
}

math *.[mathcolor="black"] {
  color: black
}

math *.[mathcolor="red"] {
```

```
  color: red
}

math *.[mathcolor="blue"] {
  color: blue
}

math *.[mathcolor="olive"] {
   color: olive
}

math *.[mathcolor="purple"] {
   color: purple
}

math *.[mathcolor="teal"] {
   color: teal
}

math *.[mathcolor="aqua"] {
   color: aqua
}

math *.[mathcolor="gray"] {
   color: gray
}

math *.[mathbackground="blue"] {
   background-color: blue
}

math *.[mathbackground="green"] {
   background-color: green
}

math *.[mathbackground="white"] {
   background-color: white
}

math *.[mathbackground="yellow"] {
   background-color: yellow
}

math *.[mathbackground="aqua"] {
   background-color: aqua
}

} /* Close "@media screen" scope */

@media aural {
```

```
}
```

Appendix D

Glossary (Non-Normative)

Several of the following definitions of terms have been borrowed or modified from similar definitions in documents originating from W3C or standards organizations. See the individual definitions for more information.

Argument A child of a presentation layout schema. That is, 'A is an argument of B' means 'A is a child of B and B is a presentation layout schema'. Thus, token elements have no arguments, even if they have children (which can only be `malignmark`).

Attribute A parameter used to specify some property of an SGML or XML element type. It is defined in terms of an attribute name, attribute type, and a default value. A value may be specified for it on a start-tag for that element type.

Axis The axis is an imaginary alignment line upon which a fraction line is centered. Often, operators as well as characters that can stretch, such as parentheses, brackets, braces, summation signs etc., are centered on the axis, and are symmetric with respect to it.

Baseline The baseline is an imaginary alignment line upon which a glyph without a descender rests. The baseline is an intrinsic property of the glyph (namely a horizontal line). Often baselines are aligned (joined) during typesetting.

Black box The bounding box of the actual size taken up by the viewable portion (ink) of a glyph or expression.

Bounding box The rectangular box of smallest size, taking into account the constraints on boxes allowed in a particular context, which contains some specific part of a rendered display.

Box A rectangular plane area considered to contain a character or further sub-boxes, used in discussions of rendering for display. It is usually considered to have a baseline, height, depth and width.

Cascading Style Sheets (CSS) A language that allows authors and readers to attach style (e.g. fonts, colors and spacing) to HTML and XML documents.

Character A member of a set of identifiers used for the organization, control or representation of text. ISO/IEC Standard 10646-1:1993 uses the word 'data' here instead of 'text'.

Character data (`CDATA`) A data type in SGML and XML for raw data that does not include markup or entity references. Attributes of type `CDATA` may contain entity references. These are expanded by an XML processor before the attribute value is processed as `CDATA`.

Character or expression depth Distance between the baseline and bottom edge of the character glyph or expression. Also known as the descent.

Character or expression height Distance between the baseline and top edge of the character glyph or expression. Also known as the ascent.

Character or expression width Horizontal distance taken by the character glyph as indicated in the font metrics, or the total width of an expression.

Condition A MathML content element used to place a mathematical condition on one or more variables.

Contained (element A is contained in element B) A is part of B's content.

Container (Constructor) A non-empty MathML Content element that is used to construct a mathematical object such as a number, set, or list.

Content elements MathML elements that explicitly specify the mathematical meaning of a portion of a MathML expression (defined in Chapter 4).

Content token element Content element having only `PCDATA`, `sep` and presentation expressions as content. Represents either an identifier (`ci`) or a number (`cn`).

Context (of a given MathML expression) Information provided during the rendering of some MathML data to the rendering process for the given MathML expression; especially information about the MathML markup surrounding the expression.

Declaration An instance of the declare element.

Depth (of a box) The distance from the baseline of the box to the bottom edge of the box.

Direct sub-expression (of a MathML expression 'E') A sub-expression directly contained in E.

Directly contained (element A in element B) A is a child of B (as defined in XML), in other words A is contained in B, but not in any element that is itself contained in B.

Document Object Model A model in which the document or Web page is treated as an object repository. This model is developed by the DOM Working Group (DOM) of the W3C.

Document Style Semantics and Specification Language (DSSSL) A method of specifying the formatting and transformation of SGML documents. ISO International Standard 10179:1996.

Document Type Definition (DTD) In SGML or XML, a DTD is a formal definition of the elements and the relationship among the data elements (the structure) for a particular type of document.

Em A font-relative measure encoded by the font. Before electronic typesetting, an `"em"` was the width of an 'M' in the font. In modern usage, an `"em"` is either specified by the designer of the font or is taken to be the height (point size) of the font. Em's are typically used for font-relative horizontal sizes.

Ex A font-relative measure that is the height of an 'x' in the font. `"ex"`s are typically used for font-relative vertical sizes.

Height (of a box) The distance from the baseline of the box to the top edge of the box.

Inferred `mrow` An `mrow` element that is 'inferred' around the contents of certain layout schemata when they have other than exactly one argument. Defined precisely in Section 3.1.6

Embedded object Embedded objects such as Java applets, Microsoft Component Object Model (COM) objects (e.g. ActiveX Controls and ActiveX Document embeddings), and plug-ins that reside in an HTML document.

Embellished operator An operator, including any 'embellishment' it may have, such as superscripts or style information. The 'embellishment' is represented by a layout schema that contains the operator itself. Defined precisely in Section 3.2.5.

Entity reference A sequence of ASCII characters of the form `&name;` representing some other data, typically a non-ASCII character, a sequence of characters, or an external source of data, e.g. a file containing a set of standard entity definitions such as ISO Latin 1.

Extensible Markup Language (XML) A simple dialect of SGML intended to enable generic SGML to be served, received, and processed on the Web.

Fences In typesetting, bracketing tokens like parentheses, braces, and brackets, which usually appear in matched pairs.

Font A particular collection of glyphs of a typeface of a given size, weight and style, for example 'Times Roman Bold 12 point'.

Glyph The actual shape (bit pattern, outline) of a character. ISO/IEC Standard 9541-1:1991 defines a glyph as a recognizable abstract graphic symbol that is independent of any specific design.

Indirectly contained A is contained in B, but not directly contained in B.

Instance of MathML A single instance of the top level element of MathML, and/or a single instance of embedded MathML in some other data format.

Inverse function A mathematical function that, when composed with the original function acts like an identity function.

Lambda expression A mathematical expression used to define a function in terms of variables and an expression in those variables.

Layout schema (plural: schemata) A presentation element defined in chapter 3, other than the token elements and empty elements defined there (i.e. not the elements defined in Section 3.2 and Section 3.5.5, or the empty elements `none` and `mprescripts` defined in Section 3.4.7). The layout schemata are never empty elements (though their content may contain nothing in some cases), are always expressions, and all allow any MathML expressions as arguments (except for requirements on argument count, and the requirement for a certain empty element in `mmultiscripts`).

Mathematical Markup Language (MathML) The markup language specified in this document for describing the structure of mathematical expressions, together with a mathematical context.

MathML element An XML element that forms part of the logical structure of a MathML document.

MathML expression (within some valid MathML data) A single instance of a presentation element, except for the empty elements `none` or `mprescripts`, or an instance of `malignmark` within a token element (defined below); or a single instance of certain of the content elements (see Chapter 4 for a precise definition of which ones).

Multi-purpose Internet Mail Extensions (MIME) A set of specifications that offers a way to interchange text in languages with different character sets, and multimedia content among many different computer systems that use Internet mail standards.

Operator, content element A mathematical object that is applied to arguments using the `apply` element.

Operator, an `mo` element Used to represent ordinary operators, fences, separators in MathML presentation. (The token element `mo` is defined in Section 3.2.5).

OpenMath A general representation language for communicating mathematical objects between application programs.

Parsed character data (`PCDATA`) An SGML/XML data type for raw data occurring in a context where text is parsed and markup (for instance entity references and element start/end tags) is recognized.

Point Point is often abbreviated 'pt'. The value of 1 pt is approximately 1/72 inch. Points are typically used to specify absolute sizes for font-related objects.

Pre-defined function One of the empty elements defined in [mathml3cds] and used with the `apply` construct to build function applications.

Presentation elements MathML tags and entities intended to express the syntactic structure of mathematical notation (defined in Chapter 3).

Presentation layout schema A presentation element that can have other MathML elements as content.

Presentation token element A presentation element that can contain only parsed character data or the `malignmark` element.

Qualifier A MathML content element that is used to specify the value of a specific named parameter in the application of selected pre-defined functions.

Relation A MathML content element used to construct expressions such as $a < b$.

Render Faithfully translate into application-specific form allowing native application operations to be performed.

Schema Schema (plural: schemata or schemas). See 'presentation layout schema'.

Scope of a declaration The portion of a MathML document in which a particular definition is active.

Selected sub-expression (of an `maction` element) The argument of an `maction` element (a layout schema defined in Section 3.6) that is (at any given time) 'selected' within the viewing state of a MathML renderer, or by the `selection` attribute when the element exists only in MathML data. Defined precisely in the abovementioned section.

Space-like (MathML expression) A MathML expression that is ignored by the suggested rendering rules for MathML presentation elements when they determine operator forms and effective operator rendering attributes based on operator positions in `mrow` elements. Defined precisely in Section 3.2.7.

Standard Generalized Markup Language (SGML) An ISO standard (ISO 8879:1986) that provides a formal mechanism for the definition of document structure via DTDs (Document Type Definitions), and a notation for the markup of document instances conforming to a DTD.

Sub-expression (of a MathML expression 'E') A MathML expression contained (directly or indirectly) in the content of E.

Suggested rendering rules for MathML presentation elements Defined throughout Chapter 3; the ones that use other terms defined here occur mainly in Section 3.2.5 and in Section 3.6.

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 2.5.2).

Typeface A typeface is a specific design of a set of letters, numbers and symbols, such as 'Times Roman' or 'Chicago'.

Valid MathML data MathML data that (1) conforms to the MathML DTD, (2) obeys the additional rules defined in the MathML standard for the legal contents and attribute values of each MathML element, and (3) satisfies the EBNF grammar for content elements.

Width (of a box) The distance from the left edge of the box to the right edge of the box.

Extensible Style Language (XSL) A style language for XML developed by W3C. See XSL FO and XSLT.

XSL Formatting Objects (XSL FO) An XML vocabulary to express formatting, which is a part of XSL.

XSL Transformation (XSLT) A language to express the transformation of XML documents into other XML documents.

Appendix E

Working Group Membership and Acknowledgments (Non-Normative)

E.1 The Math Working Group Membership

The present W3C Math Working Group (2006-2008) is co-chaired by Patrick Ion of the AMS and Robert Miner of Design Science. Contact the co-chairs about membership in the Working Group. For the present membership see the W3C Math home page.

Participants in the Working Group responsible for MathML 3.0 are:

- 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
- St\'ephane Dalmas, INRIA, Sophia Antipolis, FRA
- Stan Devitt, Agfa-Gevaert N. V., Trier, GER
- Margaret Hinchcliffe, Waterloo Maple, Inc., Waterloo ON, CAN
- Patrick Ion, W3C Invited Experts:Mathematical Reviews (American Mathematical Society), Ann Arbor MI, USA
- Michael Kohlhase, German Research Center for Artificial Intelligence (DFKI) Gmbh, GER
- Azzeddine Lazrek, W3C Invited Experts: University of Marrakesh, Morocco
- Dennis Leas, DAISY Consortium
- Paul Libbrecht, German Research Center for Artificial Intelligence (DFKI) Gmbh, GER
- Manolis Mavrikis, University of Edinburgh, Edinburg, UK
- Bruce Miller, National Institute of Standards and Technology (NIST), Gaithersburg MD, USA
- Robert Miner, Design Science Inc., Long Beach CA, USA
- Murray Sargent III, Microsoft, Redmond WA, USA
- Kyle Siegrist, Mathematical Association of America, Washington DC, USA
- Neil Soiffer, Design Science Inc., Long Beach CA, USA
- Stephen Watt, University of Western Ontario, London ON, CAN
- Mohamed Zergaoui, Innovimax, Paris, FRA

For 2003 to 2006 W3C Math Activity comprised a Math Interest Group, chaired by David Carlsisle 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
- St\'ephane 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
- St\'ephane 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
- St\'ephane 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 6, as well as Peter Flynn, Russell S.S. O'Connor, Andreas Strotmann, and other contributors to the www-math mailing list for their careful proofreading and constructive criticisms.

As the Math Working Group went on to MathML 2.0, it again was helped by many from the W3C family of Working Groups with whom we necessarily had a great deal of interaction. Outside the W3C, a particularly active relevant front was the interface with the Unicode Technical Committee (UTC) and the NTSC WG2 dealing with ISO 10646. There the STIX project put together a proposal for the addition of characters for mathematical notation to Unicode, and this work was again spearheaded by Barbara Beeton of the AMS. The whole problem ended split into three proposals, two of which were advanced by Murray Sargent of Microsoft, a Math WG member and

member of the UTC. But the mathematical community should be grateful for essential help and guidance over a couple of years of refinement of the proposals to help mathematics provided by Kenneth Whistler of Sybase, and a UTC and WG2 member, and by Asmus Freytag, also involved in the UTC and WG2 deliberations, and always a stalwart and knowledgeable supporter of the needs of scientific notation.

Appendix F

Changes (Non-Normative)

F.1 Changes between MathML 2.0 Second Edition and MathML 3.0

Issue (): The current appendix is just a stub that will be completed in later drafts.

- Changes to Chapter 4.The concept of a Content Dictionary was introduced in MathML3, the whole chapter and the content dictionaries were compiled anew.

Appendix G

References (Non-Normative)

[AAP-math] ANSI/NISO Z39.59-1998; *AAP Math DTD*, Standard for Electronic Manuscript Preparation and MarkUp. (Association of American Publishers, Inc., Washington, DC) Bethesda, MD, 1988.

[Abramowitz1997] Abramowitz, Milton, Irene A. Stegun (editors); *Mathematical Fuctions: With Formulas, Graphs, and Mathematical Tables.* Dover Publications Inc., December 1977, ISBN: 0-4866-1272-4.

[Behaviors] Vidur Apparao, Daniel Glazman, and Chris Wilson (editors) *Behavioral Extensions to CSS* World Wide Web Consortium Working Draft, 4 August 1999. (`http://www.w3.org/TR/1999/WD-becss-19990804`)

[Bidi] Mark Davis; *The Bidirectional Algorithm*, Unicode Standard Annex #9, August 2000. (`http://www.unicode.org/unicode/reports/tr9/`)

[Buswell1996] Buswell, S., Healey, E.R. Pike, and M. Pike; *SGML and the Semantic Representation of Mathematics.* UIUC Digital Library Initiative SGML Mathematics Workshop, May 1996 and SGML Europe 96 Conference, Munich 1996.

[CSS1] Lie, Håkon Wium and Bert Bos; *Cascading Style Sheets, level 1*, W3C Recommendation, 17 December 1996. (`http://www.w3.org/TR/1999/REC-CSS1-19990111`)

[CSS2] Bert Bos, Håkon Wium Lie, Chris Lilley, and Ian Jacobs (editors); *Cascading Style Sheets, level 2 CSS2 Specification*, W3C Recommendation, 12 May 1998. (`http://www.w3.org/TR/1998/REC-CSS2-19980512/`)

[CSS21] Bert Bos, Tantek Çelik, Ian Hickson, Håkon Wium Lie (editors); *Cascading Style Sheets, Level 2 Revision 1 (CSS 2.1) Specification*, W3C Candidate Recommendation 19 July 2007. (`http://www.w3.org/TR/CSS21/`)

[Cajori1928] Cajori, Florian; *A History of Mathematical Notations*, vol. I & II. Open Court Publishing Co., La Salle Illinois, 1928 & 1929 republished Dover Publications Inc., New York, 1993, xxviii+820 pp. ISBN 0-486-67766-4 (paperback).

[Carroll1871] Carroll, Lewis [Rev. C.L. Dodgson]; *Through the Looking Glass and What Alice Found There.* Macmillian & Co., 1871.

[Chaundy1954] Chaundy, T.W., P.R. Barrett, and C. Batey; *The Printing of Mathematics. Aids for authors and editors and rules for compositors and readers at the University Press, Oxford.* Oxford University Press, London, 1954, ix+105 pp.

[DOM] Arnaud Le Hors, Philippe Le H\'egaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne (editors); *Document Object Model (DOM) Level 2 Core Specification* World Wide Web Consortium Recommendation, 13 November, 2000. (`http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/`)

[Entities] David Carlisle, *XML Entity definitions for Characters (Editor's draft)* Draft 17 November 2007 (`http://www.w3.org/2003/entities/2007doc/`)

[HTML4] Raggett, Dave, Arnaud Le Hors and Ian Jacobs (editors); *HTML 4.01 Specification*, 24 December 1999. (`http://www.w3.org/TR/1999/REC-html401-19991224/`); section on data types.

[Higham1993] Higham, Nicholas J.; *Handbook of writing for the mathematical sciences*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1993. xii+241 pp. ISBN: 0-89871-314-5.

[ISO-12083] ISO 12083:1993; *ISO 12083 DTD* Information and Documentation - Electronic Manuscript Preparation and Markup. International Standards Organization, Geneva, Switzerland, 1993.

[ISOIEC10646-1] ISO/IEC JTC1/SC2/WG2; *ISO/IEC 10646-1: 2000*, Information technology – Universal-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane. International Standards Organization, Geneva, Switzerland, 2000.

[Knuth1986] Knuth, Donald E.; *The TEXbook*. American Mathematical Society, Providence, RI and Addison-Wesley Publ. Co., Reading, MA, 1986, ix+483 pp. ISBN: 0-201-13448-9.

[MathML1] Patrick Ion, Robert Miner, *Mathematical Markup Language (MathML) 1.01 Specification* W3C Recommendation, revision of 7 July 1999 (`http://www.w3.org/TR/REC-MathML/`)

[MathML2] David Carlisle, Patrick Ion, Robert Miner, Nico Poppelier, *Mathematical Markup Language (MathML) Version 2.0* W3C Recommendation 21 February 2001 (`http://www.w3.org/TR/2001/REC-MathML2-20010221/`)

[MathMLforCSS] Bert Bos, David Carlisle, George Chavchanidze, Patrick D. F. Ion, Bruce R. Miller*A MathML for CSS profile* W3C Working Draft 24 September 2007 (`http://www.w3.org/TR/2007/WD-mathml-for-css-20070924/`)

[Modularization] Robert Adams, Murray Altheim, Frank Boumphrey, Sam Dooley, Shane McCarron, Sebastian Schnitzenbaumer, Ted Wugofski (editors); *Modularization of XHTML[tm]*, World Wide Web Consortium Recommendation, 10 April 2001. (`http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410/`)

[Namespaces] Tim Bray, Dave Hollander, Andrew Layman (editors); *Namespaces in XML*, World Wide Web Consortium Recommendation, 14 January 1999. (`http://www.w3.org/TR/1999/REC-xml-names-19990114/`)

[OpenMath2000] O. Caprotti, D. P. Carlisle, A. M. Cohen (editors); *The OpenMath Standard*, February 2000. (http://www.openmath.org/standard)

[OpenMath2004] S. Buswell, O. Caprotti, D. P. Carlisle, M. C. Dewar, M. Gaëtano and M. Kohlhase (editors); *The OpenMath Standard Version 2.0*, June 2004. (`http://www.openmath.org/standard/om20-2004-06-30/`)

[Pierce1961] Pierce, John R.; *An Introduction to Information Theory. Symbols, Signals and Noise.*. Revised edition of *Symbols, Signals and Noise: the Nature and Process of Communication* (1961). Dover Publications Inc., New York, 1980, xii+305 pp. ISBN 0-486-24061-4.

[Poppelier1992] Poppelier, N.A.F.M., E. van Herwijnen, and C.A. Rowley; *Standard DTD's and Scientific Publishing*, EPSIG News 5 (1992) #3, September 1992, 10-19.

[RELAX-NG] Clark, James and Makoto Murata; *RELAX NG Specification*. The Organization for the Advancement of Structured Information Standards [OASIS] 2001.

[RFC2045] N. Freed and N. Borenstein; *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC 2045, November 1996. (`http://www.ietf.org/rfc/rfc2045.txt`)

[RFC2046] N. Freed and N. Borenstein; *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, RFC 2045, November 1996. (`http://www.ietf.org/rfc/rfc2046.txt`)

[RFC3023] M. Murata, S. St.Laurent and D. Kohn; *XML Media Types*, RFC 3023, January 2001. (`http://www.ietf.org/rfc/rfc3023.txt`)

[RelaxNG] *A Schema Language for XML* (`http://www.relaxng.org`)

[RelaxNGBook] Eric van der Vlist; *RELAXNG: A simple schema language for XML* O'Reilly 2004

[RodWatt2000] Igor Rodionov, Stephen Watt; *Content-Faithful Stylesheets for MathML*. Technical Report TR-00-23, Ontario Research Center for Computer Algebra, December 2000. (http://www.orcca.on.ca/TechReports/2000/TR-00-24.html)

[SVG1.1] Dean Jackson, Jon Ferraiolo, Jun Fujisawa, eds. *Scalable Vector Graphics (SVG) 1.1 Specification* W3C Recommendation, 14 January 2003 (http://www.w3.org/TR/2003/REC-SVG11-20030114/)

[Spivak1986] Spivak, M. D.; *The Joy of TEX A gourmet guide to typesetting with the AMS-TEX macro package*. American Mathematical Society, Providence, RI, MA 1986, xviii+290 pp. ISBN: 0-8218-2999-8.

[Swanson1979] Swanson, Ellen; *Mathematics into type: Copy editing and proofreading of mathematics for editorial assistants and authors*. Revised edition. American Mathematical Society, Providence, R.I., 1979. x+90 pp. ISBN: 0-8218-0053-1.

[Swanson1999] Swanson, Ellen; *Mathematics into type: Updated Edition*. American Mathematical Society, Providence, R.I., 1999. 102 pp. ISBN: 0-8218-1961-5.

[Thieme1983] Thieme, Romeo; *Satz und bedeutung mathematischer Formeln [Typesetting and meaning of mathematical formulas]*. Reprint of the 1934 original. Edited by Karl Billmann, Helmut Bodden and Horst Nacke. Werner-Verlag Gmbh, Dusseldorf, 1983, viii + 92 pp. ISBN 3-8041-3549-8.

[UAX15] Unicode Standard Annex 15, Version 4.0.0; *Unicode Normalization Forms*, The Unicode Consortium, 2003-04-17. (http://www.unicode.org/reports/tr15/tr15-23.html)

[Unicode] The Unicode Consortium; *The Unicode Standard, Version 5.0*, Addison-Wesley Professional. ISBN 0321480910. (http://www.unicode.org/unicode/standard/standard.html)

[XHTML] Steve Pemberton, Murray Altheim, et al.; *XHTML[tm] 1.0: The Extensible HyperText Markup Language* World Wide Web Consortium Recommendation, 26 January 2000. (http://www.w3.org/TR/2000/REC-xhtml1-20000126/)

[XHTML-MathML-SVG] Masayasu Ishikawa, ed., *An HTML + MathML + SVG Profile* W3C Working Draft, 9 August 2002. (http://www.w3.org/TR/2002/WD-XHTMLplusMathMLplusSVG-20020809/)

[XLink] Steve DeRose, Eve Maler, David Orchard (editors); *XML Linking Language (XLink) Version 1.0*, World Wide Web Consortium Recommendation, 27 June 2001. (http://www.w3.org/TR/2001/REC-xlink-20010627/)

[XML] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen and Eve Maler (editors); *Extensible Markup Language (XML) 1.0 (Second Edition)*, 6 October 2000. (http://www.w3.org/TR/2000/REC-xml-20001006)

[XMLSchemas] David C. Fallside, editor; *XML Schema Part 0: Primer*, World Wide Web Consortium Recommendation, 2 May 2001. (http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/)

[XPath] James Clark, Steve DeRose(editors); *XML Path Language Version 1.0*, World Wide Web Consortium Recommendation, 16. November 1999. (http://www.w3.org/TR/1999/REC-xpath-19991116)

[XPointer] Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh (editors); *XML Pointer Framework*, World Wide Web Consortium Recommendation, 25 March 2003. (http://www.w3.org/TR/2003/REC-xptr-framework-20030325/)

[XSLT] James Clark (editor); *XSL Transformations (XSLT) Version 1.0*, World Wide Web Consortium Recommendation, 16 November 1999. (http://www.w3.org/TR/1999/REC-xslt-19991116)

[Zwillinger1988] Daniel Zwillinger (editor); *Standard Mathematical Tables and Formulae (30th Edition)*. CRC Press LLC, January 1996, ISBN: 0-8493-2479-3.

[mathml3cds] Carlisle, Davenport, Kohlhase, eds; *The MathML3 Content Dictionaries*. Joint Document by OpenMath Society and W3C Math WG, under development.

[owl] Deborah L. McGuinness and Frank van Harmelen (editors); *OWL Web Ontology Language Overview* February 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/

[rdf] Graham Klyne, Jeremy J. Carroll, Brian McBride (editors); *Resource Description Framework (RDF): Concepts and Abstract Syntax*,February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[roadmap] Patrick Ion, Robert Minor, Math Working Group Roadmap 2007/8 (http://www.w3.org/Math/Roadmap/)

[sgml-xml] J. Clark; *Comparison of SGML and XML*, W3C Note, December 1997. (http://www.w3.org/TR/NOTE-sgml-xml-971215.html)

[xml11] World Wide Web Consortium *Extensible Markup Language (XML) 1.1.* W3C Recommendation, February 2004 (http://www.w3.org/TR/2004/REC-xml11-20040204/)

Appendix H

Index (Non-Normative)

H.1 MathML Elements

References to sections in which an element is defined are marked in bold.

H.2 MathML Attributes

In addition to the standard MathML attributes, some attributes from other namespaces such as Xlink or XML Schema are also listed here.