Mathematical Markup Language (MathML") 1.01 Specification

W3C Recommendation, revision of 7 July 1999

REC-MathML-19980407; revised 19990707

This version:

http://www.w3.org/1999/07/REC-MathML-19990707

Latest version:

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

Previous version:

http://www.w3.org/TR/1998/REC-MathML-19980407

Editors:

Patrick Ion <ion@ams.org>
(Mathematical Reviews / American Mathematical Society)

Robert Miner <rminer@geomtech.com>
(Geometry Technologies, Inc.)

Principal Writers:

Stephen Buswell, Stan Devitt, Angel Diaz, Patrick Ion, Robert Miner,
Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, Stephen Watt

---

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 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.

This document begins with background information on mathematical notation, the problems it poses, and the philosophy underlying the solutions MathML proposes. MathML can be used to encode both mathematical notation and mathematical content. Twenty-eight of the MathML tags describe abstract notational structures, while another seventy-five 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 MathML entities (extended characters) and their relation to fonts.

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

Status of this document

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

The fundamental eXtensible Markup Language (XML) 1.0 specification upon which MathML is based has been adopted as a W3C Recommendation. Should future changes in the XML specification necessitate changes in the MathML specification, it is the intention of the W3C Math Working Group to issue a revision of the MathML specification. However, any changes are very unlikely to be substantial.

Most of this document represents technology tested by multiple implementations. A summary of MathML rendering and authoring software is described on the W3C Math Working Group home page.

The www-math mailing list is a public forum for questions and comments about MathML and issues related to putting math on the Web.

The W3C Math Working Group intends further development of recommendations for mathematics on the Web, as set out below.

A list of current W3C Recommendations and other technical reports can be found at http://www.w3.org/TR.

This document is a revised version of the document first released on 7 April 1998. Changes from the original version are only editorial in nature. The present W3C Math Working Group is working on further improvements of MathML.

Available formats

The MathML 1.01 W3C Recommendation is made available in different formats from the W3C Math WG's site. In case of a discrepancy between any of the derived forms and that found in the W3C's archive of Recommendations the definitive version is naturally the Recommendation. At first it is expected that zipped and gzipped bundles will be made available, but such easily printable formats as PostScript or PDF may be supplied.

Available languages

The English version of this specification is the only normative version. However, for translations of this document, see http://www.w3.org/MarkUp/mathml101-updates/translations.html.

Errata

The list of known errors in this specification is available at:

http://www.w3.org/MarkUp/mathml101-updates/errata.html.

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

Table of contents

Extended Table of Contents

- Chapter 1. Introduction
- Chapter 2. MathML Fundamentals
- Chapter 3. Presentation Markup
- Chapter 4. Content Markup
- Chapter 5. Mixing Presentation and Content
- Chapter 6. Entities, Characters and Fonts
- Chapter 7. Implementing MathML
- Appendix A. DTD for MathML
- Appendix B. Glossary
- Appendix C. Operator Dictionary

- Appendix D. Working Group Membership
- Appendix E. Informal EBNF Grammar for Content Elements
- Appendix F. Default Semantic Bindings for Content Elements
- Appendix G. MathML 1.0 Changes
- References

REC-MathML-19980407; revised 19990707

Mathematical Markup Language 1.01 Specification

Table of Contents

- Title page and Abstract
- 1. Introduction
  - 1.1 Mathematics and its Notation
  - 1.2 Origins and Goals
    - 1.2.1 The History of MathML
    - 1.2.2 Limitations of HTML
    - 1.2.3 Requirements for Mathematical Markup
    - 1.2.4 Goals of MathML
  - 1.3 The Role of MathML on the Web
    - 1.3.1 Layered Design of Mathematical Web Services
    - 1.3.2 Relation to Other Web Technology
- 2. MathML Fundamentals
  - 2.1 MathML Overview
    - 2.1.1 Taxonomy of MathML Elements
    - 2.1.2 Expression Trees and Token Elements
    - 2.1.3 Presentation Markup
    - 2.1.4 Content Markup
    - 2.1.5 Mixing Presentation and Content
  - 2.2 Some MathML Examples
    - 2.2.1 Presentation Examples
    - 2.2.2 Content Examples
    - 2.2.3 Mixed Markup Examples
  - 2.3 MathML Syntax and Grammar
    - 2.3.1 An XML Syntax Primer

        - 2.3.2 Children vs. Arguments
        - 2.3.3 MathML Attributes Values
        - 2.3.4 Attributes Shared by all MathML Elements
        - 2.3.5 Collapsing Whitespace in Input
- 3. Presentation Markup
    - 3.1 Introduction
        - 3.1.1 What Presentation Elements Represent
        - 3.1.2 Terminology Used In This Chapter
        - 3.1.3 Required Arguments
        - 3.1.4 Elements with Special Behaviors
        - 3.1.5 Summary of Presentation Elements
    - 3.2 Token elements
        - 3.2.1 Attributes common to token elements
        - 3.2.2 **<mi>** -- identifier
        - 3.2.3 **<mn>** -- number
        - 3.2.4 **<mo>** -- operator, fence, or separator
        - 3.2.5 **<mtext>** -- text
        - 3.2.6 **<mspace/>** -- space
        - 3.2.7 **<ms>** -- string literal
    - 3.3 General Layout Schemata
        - 3.3.1 **<mrow>** -- horizontally group any number of subexpressions
        - 3.3.2 **<mfrac>** -- form a fraction from two subexpressions
        - 3.3.3 **<msqrt>** and **<mroot>** -- form a radical
        - 3.3.4 **<mstyle>** -- style change
        - 3.3.5 **<merror>** -- enclose a syntax error message from a preprocessor
        - 3.3.6 **<mpadded>** -- adjust space around content
        - 3.3.7 **<mphantom>** -- make content invisible but preserve its size
        - 3.3.8 **<mfenced>** -- surround content with a pair of fences
    - 3.4 Script and Limit Schemata
        - 3.4.1 **<msub>** -- attach a subscript to a base
        - 3.4.2 **<msup>** -- attach a superscript to a base
        - 3.4.3 **<msubsup>** -- attach a subscript-superscript pair to a base

        - 3.4.4 **<munder>** -- attach an underscript to a base
        - 3.4.5 **<mover>** -- attach an overscript to a base
        - 3.4.6 **<munderover>** -- attach an underscript-overscript pair to a base
        - 3.4.7 **<mmultiscripts>** -- attach prescripts and tensor indices to a base
    - 3.5 Tables and Matrices
        - 3.5.1 **<mtable>** -- table or matrix
        - 3.5.2 **<mtr>** -- row in a table or matrix
        - 3.5.3 **<mtd>** -- one entry in a table or matrix
        - 3.5.4 **<maligngroup/>** and **<malignmark/>** -- alignment markers
    - 3.6 Enlivening Expressions
        - 3.6.1 **<maction>** -- bind actions to a subexpression
- 4. Content Markup -- Index of All Content Elements
    - 4.1 Introduction
        - 4.1.1 The Intent of Content Markup
        - 4.1.2 The Scope of Content Markup
        - 4.1.3 Basic Concepts of Content Markup
    - 4.2 Content Element Usage Guide
        - 4.2.1 Overview of Syntax and Usage
        - 4.2.2 Containers
        - 4.2.3 Functions, Operators and Qualifiers
        - 4.2.4 Relations
        - 4.2.5 Conditions
        - 4.2.6 Syntax and Semantics
        - 4.2.7 Semantic Mappings
        - 4.2.8 MathML element types
    - 4.3 Content Element Attributes
        - 4.3.1 Content Element Attribute Values
        - 4.3.2 Attributes Modifying Content Markup Semantics
        - 4.3.3 Attributes Modifying Content Markup Rendering
    - 4.4 The Content Markup Elements
        - 4.4.1 Token Elements
        - 4.4.2 Basic Content Elements

- 4.4.3 Arithmetic, Algebra and Logic
- 4.4.4 Relations
- 4.4.5 Calculus
- 4.4.6 Theory of Sets
- 4.4.7 Sequences and Series
- 4.4.8 Trigonometry
- 4.4.9 Statistics
- 4.4.10 Linear Algebra
- 4.4.11 Semantic Mapping Elements

- 5. Mixing Presentation and Content
  - 5.1 When to Use Mixed Markup
    - 5.1.1 Why Two Different Kinds of Markup?
    - 5.1.2 Reasons to Mix Markup
  - 5.2 How to use Mixed Markup
    - 5.2.1 Presentation Markup Contained in Content Markup
    - 5.2.2 Content Markup Contained in Presentation Markup
  - 5.3 Anticipating Macros for Combined Markup
- 6. Entities, Characters and Fonts
  - 6.1 Introduction
    - 6.1.1 The Intent of Entity Names
    - 6.1.2 The STIX Project
  - 6.2 Entity Listings
    - 6.2.1 Non-Marking Entities
    - 6.2.2 Printing Entity List
    - 6.2.3 Special Constants
    - 6.2.4 Full Alphabetical Lists
    - 6.2.5 ISO Entity Set Groupings
      - 6.2.5.1 ISO Symbol Entity Sets
      - 6.2.5.2 ISO Math Font Entity Sets
      - 6.2.5.3 Other ISO Font Entity Sets
    - 6.2.6 Additional Entity Set Grouping
- 7. Implementing MathML

- 7.1 Embedding MathML in HTML
  - 7.1.1 The Top-Level **math** Element
  - 7.1.2 Requirements for a MathML Browser Interface
  - 7.1.3 Invoking Embedded Objects as Renderers
  - 7.1.4 Invoking Other Applications
  - 7.1.5 Mixing and Linking MathML and HTML
- 7.2 Generating, Processing and Rendering MathML
  - 7.2.1 MathML Compliance
  - 7.2.2 Handling Of Errors
  - 7.2.3 An Attribute for Unspecified Data
- 7.3 Future Extensions
  - 7.3.1 Macros and Style Sheets
  - 7.3.2 XML Extensions to MathML

- Appendix A. DTD for MathML
- Appendix B. Glossary
- Appendix C. Operator Dictionary
- Appendix D. Working Group Membership
- Appendix E. Informal EBNF Grammar for Content Elements
- Appendix F. Default Semantic Bindings for Content Elements
- Appendix G. MathML 1.0 Changes
- References

1. Introduction

- 1.1 Mathematics and its Notation
- 1.2 Origins and Goals
  - 1.2.1 The History of MathML
  - 1.2.2 Limitations of HTML
  - 1.2.3 Requirements for Math Markup
  - 1.2.4 Design Goals of MathML
- 1.3 The Role of MathML on the Web
  - 1.3.1 Layered Design of Mathematical Web Services
  - 1.3.2 Relation to Other Web Technology

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 [Pierce 1961]. Mathematical ideas 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 in putting math on the Web is to capture both notation and content in such a way that documents can utilize the highly-evolved notational practices of print, and the potential for interconnectivity in electronic media.

Mathematical notations are constantly evolving as people continue to discover innovative 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 [Cajori 1928/1929]. Modern mathematical notation is the product of centuries of refinement, and the notational conventions for high-quality typesetting are quite complicated. For example, variables, or letters which stand for numbers, are usually typeset today in a special italic font subtly distinct from the text italic. Spacing around symbols for operations such as +, -, x and / is slightly different from that of text, to reflect conventions about operator precedence. Entire books have been devoted to the conventions of mathematical typesetting, from the alignment of superscripts and subscripts, to rules for choosing parenthesis

sizes, to specialized notational practices for subfields of mathematics [for instance, Chaudry, Barret and Batey, 1954, Swanson 1979, Higham 1993, or in the TeX literature Knuth 1986, and Spivak 1986].

Notational conventions in mathematics, and 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 rely on hundreds of 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 (an invention due to G. Peano, who is probably better known for his axioms for the natural numbers). Such notational conventions are even more important for electronic media, where one must contend with the difficulties of on-screen reading.

However, there is more to putting math 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 *interconnectivity* plays a central role. It is becoming increasingly 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 is an important step in developing Mathematics on the Web.

1.2 Origins and Goals

1.2.1 The History of MathML

The problem of encoding mathematics for computer processing or electronic communication is much older than the Web. The common practice among scientists before the Web was to write papers in some encoded form based on the ASCII character set, and e-mail them to each other. Several markup methods for mathematics, in particular TeX [Knuth 1986], were already in wide use in 1992, just before the Web rose to prominence, [Poppelier, van Herwijnen and Rowley 1992].

Since its inception, the Web has demonstrated itself to be a very effective method of making information available to widely separated groups of individuals. However, even though the World Wide Web was initially conceived and implemented by scientists for scientists, the capability to include mathematical expressions in HTML is very limited. At present, most mathematics on the Web consists of text with GIF images of scientific notation, which are difficult to read and to author.

The World Wide Web Consortium (W3C) recognized that lack of support for scientific communication was a serious problem. Dave Raggett included a proposal for HTML Math in the HTML 3.0 working draft in 1994. A panel discussion on math markup was held at the WWW Conference in Darmstadt in April 1995. In November 1995, representatives from

Wolfram Research presented a proposal for doing math in HTML to the W3C team. In May 1996, the Digital Library Initiative meeting in Champaign-Urbana played an important role in bringing together many interested parties. Following the meeting, an HTML Math Editorial Review Board was formed. In the intervening years, this group has grown, and was formally reconstituted as the W3C Math working group in March 1997.

The MathML proposal reflects the interests and expertise of a very diverse group. Many contributions to the development of MathML deserve special mention, some of which we touch on here. One such contribution concerns the question of accessibility, especially for the visually handicapped. T. V. Raman is particularly notable in this regard. Neil Soiffer and Bruce Smith from Wolfram Research shared their experience with the problems of representing mathematics in connection with the design of Mathematica 3.0, which was an important influence in the design of the presentation elements. Paul Topping from Design Science also contributed his expertise in mathematical formatting and editing. MathML has benefited from the participation of a number of working group members involved in other math encoding efforts in the SGML and Nico Poppelier from Elsevier Science, Stéphane Dalmas from INRIA, Sophia Antipolis, Stan Devitt from Waterloo Maple, Angel Diaz and Robert S. Sutor from IBM, and Stephen M. Watt from the University of Western Ontario. In particular, MathML has been influenced by the OpenMath project, the work of the ISO 12083 working group, and Stilo Technologies' work on a 'semantic' math DTD fragment. The American Mathematical Society has played a key role in the development of MathML. Among other things, it has provided two working group chairs: Ron Whitney led the group from May 1996 to March 1997, and Patrick Ion, who has co-chaired the group with Robert Miner from The Geometry Center, from March 1997 to the present.

The working group has benefited from the help of many people. 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, Russel S. S. O'Connor, Andreas Strotmann, and other contributors to the www-math mailing list for their careful proofreading and constructive criticisms.

1.2.2 Limitations of HTML

The demand for effective means of electronic scientific communication is high. Increasingly, researchers, scientists, engineers, educators, students and technicians find themselves working remotely and relying on electronic communication. At the same time, the image-based methods that are currently the predominant means of transmitting scientific notation over the Web are primitive and inadequate. Document quality is poor, authoring is difficult, and mathematical information contained in images is not available for searching, indexing, or reuse in other applications.

The most obvious problems with HTML for mathematical communication are of two types:

Display Problems. Consider the equation $2^{2^x} = 10$. This equation is sized to match the surrounding line in 14pt type on the system where it was authored. Of course, on other systems,

or for other font sizes, the equation is too small or too large. A second point to observe is that the equation image was generated against a white background. Thus, if a reader or browser resets the page background to another color, the anti-aliasing in the image results in white "halos." Next, consider the equation $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. This equation has a descender which places the baseline for the equation at a point about a third of the way from the bottom of the image. One can pad the image like this: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, so that the centerline of the image and the baseline of the equation coincide, but this causes problems with the inter-line spacing, which also makes the equation difficult to read. Moreover, center alignment of images is handled in slightly different ways by different browsers, making it impossible to guarantee proper alignment for different clients.

Image-based equations are generally harder to see, read and comprehend than the surrounding text in the browser window. Moreover, these problems become worse when the document is printed. The resolution of the equations will be around 70 dots per inch, while the surrounding text will typically be 300 or more dots per inch. The disparity in quality is judged to be unacceptable by most people.

Encoding Problems. Consider trying to search this page for part of an equation, for example, the "=10" from the first equation above. In a similar vein, consider trying to cut and paste an equation into another application; even more demanding is to cut and paste a subexpression. Using image based methods, neither of these common needs can be adequately addressed. Although the use of ALT text in the document source can help, it is clear that highly interactive Web documents must provide a more sophisticated interface between browsers and mathematical notation. Another problem with encoding mathematics as images is that it requires more bandwidth. By using markup-based encoding, more of the rendering process is moved to the client machine. Markup describing an equation is typically smaller and more compressible than an image of the equation.

1.2.3 Requirements for Math Markup

Some display problems associated with including math notation in HTML documents as images could be addressed by improving browser image handling. However, even if image handling were improved, the problem of making the information contained in mathematical expressions available to other applications would remain. Therefore, in planning for the future, it is not sufficient to merely upgrade image-based methods. To fully integrate mathematical material into Web documents, a markup-based encoding of mathematical notation and content is required.

In designing any markup language, it is essential to carefully consider the needs of its potential users. In the case of MathML, the needs of potential users cover a broad spectrum, from education to research, and on to commerce:

The education community is a large and important group that must be able to put scientific curriculum materials on the Web. At the same time, educators often have limited resources of time and equipment, and are severely hampered by the difficulty of authoring technical Web documents. Students and teachers need to be able to create mathematical content quickly and easily, using intuitive, easy-to-learn, low-cost tools.

Electronic textbooks are another way of using the Web which will potentially be very important in education. Management consultant Peter Drucker has recently been prophesying the end of big-campus residential higher education and its distribution over the Web [Drucker 1997]. Electronic textbooks will need to be active, allowing intercommunication between the text and scientific software and graphics.

The academic research community generates large volumes of dense scientific material. Increasingly, research publications are being stored in databases, such as the highly successful physics preprint server at Los Alamos National Laboratory. This is especially true in some areas of physics and mathematics where academic journal prices have been increasing at an unsustainable rate. In mathematics there are large collections at Duke, MSRI and SISSA, and on the AMS e-MATH server. In addition, databases of information on mathematical research, such as Mathematical Reviews and Zentralblatt für Mathematik, offer on the Web millions of records containing math.

To accommodate the research community, a design for math markup must facilitate the maintenance and operation of large document collections, where automatic searching and indexing are important. Because of the large collection of legacy data, especially TeX documents, the ability to convert between existing formats and new formats is also very important to the research community. Finally, the ability to maintain information for archival purposes is vital to academic research.

Corporate and academic scientists and engineers also use technical documents in their work to collaborate, to record results of experiments and computer simulations, and to verify calculations. For such uses, math on the Web must provide a standard way of sharing information that can be easily read, processed and generated using commonly available tools.

Another design requirement is the ability to render mathematical material in other media such as speech or braille, which is extremely important for the visually impaired.

Commercial publishers are also involved with math on the Web at all levels from electronic versions of print books to interactive textbooks to academic journals. Publishers require a method of putting math on the Web that is capable of high-quality output, robust enough for large-scale commercial use, and preferably compatible with their current, usually SGML-based, production systems.

1.2.4 Design Goals of MathML

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

MathML should:

- 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 math formats, both presentational and semantic. Output formats should include:
    - graphical displays
    - speech synthesizers
    - computer algebra systems' input
    - other math layout languages, such as TeX
    - plain text displays, e.g. VT100 emulators
    - print media, including braille

  It is recognized that conversion to and from other notational systems or media may entail loss of information in the process.
- allow the passing of information intended for specific renderers and applications.
- support efficient browsing for lengthy expressions.
- provide for extensibility.
- be well suited to template and other math editing techniques.
- be human legible, and simple for software to generate and process.

No matter how successfully MathML might achieve its goals as a markup language, it is clear that MathML will only be useful if it is implemented well. To this end, the W3C Math working group has identified a short list of additional implementation goals. These goals attempt to describe concisely the minimal functionality MathML rendering and processing software should try to provide.

- MathML equations in HTML 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 documents containing MathML equations should print properly and at high-quality printer resolutions.
- MathML equations in Web pages should be able to react to mouse gestures, and coordinate communication with other applications through the browser.
- Equation editors and converters should be developed to facilitate the creation of Web pages containing MathML equations.

These goals can probably be adequately addressed in the near term by using embedded elements

such as Java applets, plug-ins and ActiveX controls to render MathML. However, 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 will continue to work with the Document Object Model working group and the proposed Extensible Style Language working group to ensure that the needs of the scientific community will be met in the future.

1.3 The Role of MathML on the Web

1.3.1 Layered Design of Mathematical Web Services

The design goals of MathML require a system for encoding mathematical material for the Web which is flexible and extensible, suitable for interaction with external software, and capable of producing high-quality rendering in several media. Any markup language that encodes enough information to do all these tasks well will of necessity involve some complexity.

At the same time, it is important for many groups, such as students, to have simple ways to include math in Web pages by hand. Similarly, other groups, such as the TeX community, would be best served by a system which allowed the direct entry of markup languages like TeX in Web pages. In general, specific user groups are better served by more specialized kinds of input and output tailored to their needs. Therefore, the ideal system for communicating mathematics on the Web should provide both specialized services for input and output, and general services for interchange of information and rendering to multiple media.

In practical terms, the observation that math on the Web should provide for both specialized and general need naturally leads to the idea of a layered architecture. One layer consists of powerful, general software tools exchanging, processing and rendering suitably encoded mathematical data. A second layer consists of specialized software tools aimed at specific user groups, and which are capable of easily generating encoded mathematical data which can then be shared with a general audience.

MathML is designed to provide the encoding of mathematical data for the bottom, more general layer in a two-layer architecture. It is intended to encode complex notational and semantic structure in an explicit, regular, and easy to process way for renderers, searching and indexing software, and other mathematical applications.

As a consequence, MathML is *not* primarily intended for direct use by authors. While MathML is human-readable, in all but the simplest cases it is too verbose and error-prone for hand generation. Instead, it is anticipated that authors will use equation editors, conversion programs, and other specialized software tools to generate MathML. Alternatively, some renderers may convert other kinds of input directly included in Web pages into MathML on the fly, in response to a cut-and-paste operation, for example.

In some ways, MathML is analogous to other low-level, communication formats such as Adobe's PostScript language. You can create a PostScript file in a variety of ways, depending on your needs; experts write and modify them by hand, authors create them with word

processors, graphic artists with illustration programs, and so on. Once you have a PostScript file, however, you can share it with a very large audience, since devices which render PostScript, such as printers and screen previewers, are widely available.

Part of the reason for designing MathML as a markup language for a low-level, general, communication layer is to stimulate mathematical Web software development in the layers above. MathML provides a way of coordinating the development of modular authoring tools and rendering software. By making it easier to develop a functional piece of a larger system, MathML can stimulate a "critical mass" of software development, greatly to the benefit of potential users of math on the Web.

One can envision a similar situation for mathematical data. Authors are free to create MathML documents using the tools best suited to their needs. For example, a student might prefer to use a menu-driven equation editor that can write out MathML to an HTML file. A researcher might use a computer algebra package that automatically encodes the mathematical content of an expression, so that it can be cut from a Web page and evaluated by a colleague. An academic journal publisher might use a program that converts TeX markup to HTML and MathML. Regardless of the method used to create a MathML web page, once it exists, all the advantages of a powerful and general communication layer become available. A variety of MathML software could all be used with the same document to render it in speech or print, to send it to a computer algebra system, or to manage it as part of a large Web document collection. One may expect that eventually MathML can be integrated into other arenas where mathematical formulas occur, such as spreadsheets, statistical packages and engineering tools.

The W3C Math working group is working with vendors to ensure that a wide variety of MathML software will soon be available, including both rendering and authoring tools. A current list of MathML software is maintained at the World Wide Web Consortium.

1.3.2 Relation to Other Web Technology

The original conception of HTML Math was a simple, straightforward extension to HTML that would be natively implemented in browsers. However, very early on, the explosive growth of the Web made it clear that a general extension mechanism was required, and that math was only one of many kinds of structured data which would have to be integrated into the Web using such a mechanism.

Given that MathML must integrate into the Web as an extension, it is extremely important that MathML and MathML software can interact well with the existing Web environment. In particular, MathML has been designed with three kinds of interaction in mind. First, in order to create mathematical Web content, it is important that existing mathematical markup languages can be converted to MathML, and that existing authoring tools can be modified to generate MathML. Second, it must be possible to embed MathML markup seamlessly in HTML markup in such a way that it will be accessible to future browsers, search engines, and all kinds of Web applications which now manipulate HTML. Finally, it must be possible to render MathML embedded in HTML in today's web browsers in some fashion, even if it is less than ideal.

Existing Mathematical Markup Languages

Perhaps the most important influence on mathematical markup languages of the last two decades is the TeX typesetting system developed by Donald Knuth [Knuth 1986]. TeX is a de facto standard in the mathematical research community, and it is pervasive in the scientific community at large. TeX sets a standard for quality of visual rendering, and a great deal of effort has gone into ensuring MathML can provide the same visual rendering quality. Moreover, because of the many legacy documents in TeX, and because of the large authoring community versed in TeX, a priority in the design of MathML was the ability to convert TeX math input into MathML format. The feasibility of such conversion has been demonstrated by prototype software.

Extensive work on encoding mathematics has also been done in the SGML community, and SGML-based encoding schemes are widely used by commercial publishers. ISO 12083 is an important markup language which contains a math DTD fragment primarily intended for describing the visual presentation of mathematical notation. Because ISO 12083 math and its derivatives share many presentational aspects with TeX, and because SGML enforces structure and regularity more than TeX, much of the work in ensuring MathML is compatible with TeX also applies well to ISO12083.

MathML also pays particular attention to compatibility with other mathematical software, and in particular, with computer algebra systems. Many of the presentation elements of MathML are derived in part from the mechanism of typesetting boxes. The MathML content elements are heavily indebted to the OpenMath project and the Semantic Maths DTD. The OpenMath project has close ties to both the SGML and computer algebra communities, and has laid a foundation for an SGML-based means of communication between mathematical software packages, among other things. The feasibility of both generating and interpreting MathML in computer algebra systems has been demonstrated by prototype software.

HTML Extension Mechanisms

As noted above, the success of HTML has led to enormous pressure to incorporate a wide variety of data types and software applications into the Web. Each new format or application potentially places new demands on HTML and on browser vendors. For some time, it has been clear that a general extension mechanism is necessary to accommodate new extensions to HTML. We began our work thinking of a plain extension to HTML in the spirit of the first math support suggested for HTML 3.2. But for various reasons, once we got into the details this proved to be not so good an idea. Since work first began on MathML, XML has emerged as the leading candidate for such a general extension mechanism.

XML stands for Extensible Markup Language. It is designed as a simplified version of SGML, the meta-language used to define the grammar and syntax of HTML. One of the goals of XML is to be suitable for use on the Web, and in the context of this discussion it can be viewed as a general mechanism for extending HTML. As its name implies, extensibility is a key feature of

XML; authors are free to declare and use new tags and attributes. At the same time, XML grammar and syntax rules carefully enforces document structure to facilitate automatic processing and maintenance of large document collections.

Though details about how XML markup will ultimately be embedded in HTML remain to be resolved, XML has garnered broad industry support including major browser vendors. Devising a standard way of embedding XML in HTML is also important with the W3C. Furthermore, other applications of XML for all kinds of document publishing and processing promise to become increasingly important. Consequently, both on theoretical and pragmatic grounds, it makes a great deal of sense to specify MathML as an XML application, and we have done so.

Browser Extension Mechanisms

While details of a general model for rendering and processing XML extensions to HTML is still being being resolved, broad features of the model are already fairly clear. Formatting Properties developed by the Cascading Style Sheets and Formatting Properties Working Group for CSS and made available through the Document Object Model (DOM) will be applied to MathML elements to obtain some stylistic control over the presentation of MathML. Further development of these Formatting Properties falls within the charter of both the CSS&FP and the XSL working groups. Thus, it may soon be possible to write a style sheet which will largely describe the correct display of MathML.

MathML was designed with the goal of style sheet-based rendering in mind. It is the intention of the W3C Math Working Group to work closely with W3C style sheet activities to ensure both that adequate support for MathML is incorporated into future style sheet mechanisms, and that MathML style sheets are developed. In particular, providing for adequate follow-on activities beyond the scope of the W3C Math working group charter is a high priority.

Until style sheet mechanisms are capable of delivering native browser rendering of MathML, however, it is necessary to extend browser capabilities by using embedded elements to render MathML. It may soon be possible to instruct a browser to use a particular embedded renderer to process embedded XML markup such as MathML, and coordinate the resulting output with the surrounding Web page. Indeed, for specialized processing, such as connecting to a computer algebra system, this capability is likely to remain highly desirable. However, for this kind of interaction to be really satisfactory, it will be necessary to define a document object model rich enough to facilitate complicated interactions between browsers and embedded elements. For this reason, the W3C Math working group is coordinating its efforts closely with the Document Object Model working group.

For processing by embedded elements, and for inter-communication between scientific software generally, a style sheet-based layout model is less than ideal in some ways. It can impose an additional implementation burden in a setting where it may offer few advantages, and it imposes implementation requirements for coordination between browsers and embedded renderers that will likely be unavailable in the immediate future.

For these reasons, the MathML specification defines an attribute-based layout model, which has proven very effective for high-quality rendering of complicated mathematical expressions in several independent implementations. MathML presentation attributes utilize W3C Formatting Properties where possible. Also, MathML elements accept class, style and id attributes to facilitate their use with CSS style sheets. However, at present, there are few settings where CSS machinery is currently available to MathML renderers.

When style sheet mechanisms become available to MathML, it is anticipated their use will become the dominant method of stylistic control of MathML presentation meant for use in rendering environments which support those mechanisms.

Next: MathML Fundamentals
Up: Table of Contents

2. MathML Fundamentals

- 2.1 MathML Overview
  - 2.1.1 Taxonomy of MathML Elements
  - 2.1.2 Expression Trees and Token Elements
  - 2.1.3 Presentation Markup
  - 2.1.4 Content Markup
  - 2.1.5 Mixing Presentation and Content
- 2.2 Some MathML Examples
  - 2.2.1 Presentation Examples
  - 2.2.2 Content Examples
  - 2.2.3 Mixed Markup Examples
- 2.3 MathML Syntax and Grammar
  - 2.3.1 An XML Syntax Primer
  - 2.3.2 Children vs. Arguments
  - 2.3.3 MathML Attribute Values
    - Syntax notations used in the MathML specification
    - Attributes with units
    - CSS-compatible attributes
    - Default values of attributes
    - Attribute values in the MathML DTD
  - 2.3.4 Attributes Shared by all MathML Elements
  - 2.3.5 Collapsing Whitespace in Input

MathML Overview

This chapter introduces the basic ideas of MathML. The first section describes the overall design of MathML. The second section present a number of motivating examples, to give the reader something concrete to refer to while reading subsequent chapters of the MathML Specification. The final section describes basic features of the MathML syntax and grammar,

which apply to all MathML markup. In particular, Section 2.3 should be read *before* Chapters 3, 4 and 5.

A fundamental challenge in defining a mathematics markup language for the Web is reconciling the need to encode both the presentation of a mathematical notation and the content of the mathematical idea or object which it represents.

The relationship between a mathematical notation and a mathematical idea is subtle and deep. On a formal level, the results of mathematical logic raise unsettling questions about the correspondence between symbolic logic systems and the phenomena they model. At a more intuitive level, anyone who uses mathematical notation knows the difference that a good choice of notation can make; the symbolic structure of the notation suggests the logical structure. For example, the Leibniz notation for derivatives "suggests" the chain rule of calculus through the symbolic cancellation of fractions:

$$\frac{\partial f}{\partial x}\frac{\partial x}{\partial t} = \frac{\partial f}{\partial t}$$

Mathematicians and teachers understand this very well; part of their expertise lies in choosing notation that emphasizes key aspects of a problem while hiding or diminishing extraneous aspects. It is commonplace in math and science to write one thing when technically something else is meant, because long experience shows this actually communicates the idea better at some higher level.

In many other settings, though, mathematical notation is used to encode the full, precise meaning of a mathematical object. Mathematical notation is capable of prodigious rigor, and when used carefully, it is virtually free of ambiguity. Moreover, it is precisely this lack of ambiguity which makes it possible to describe mathematical objects so that they can be used by software applications such as computer algebra systems and voice renderers. In situations where such inter-application communication is of paramount importance, the nuances of visual presentation generally play a minimal role.

MathML allows authors to encode both the notation which represents a mathematical object and the mathematical structure of the object itself. Moreover, authors can mix both kinds of encoding in order to specify both the presentation and content of a mathematical idea. The remainder of this section gives a basic overview of how MathML can be used in each of these ways.

2.1.1 Taxonomy of MathML Elements

All MathML elements fall into one of three categories: presentation elements, content elements and interface elements. Each of these categories is described in detail in chapters 3, 4 and 7 respectively.

Presentation elements describe mathematical notation structure. Typical examples are the **mrow** element, which is used to indicate a horizontal row of characters, and the **msup** element, which is used to indicate a base and superscript. As a general rule, each presentation element corresponds to a single kind of notational "schema" such as a row, a superscript, an underscript and so on. Since many notational schemata have a number of frequently occurring variants, most presentation elements accept a number of attributes which can be used to select between variants. For example, the superscript element accepts a "superscript shift" attribute which specifies the minimum amount the superscript should shift upward.

Content elements describe mathematical objects directly, as opposed to describing the notation which represents them. Typical examples include the **plus** element, which denotes the usual addition operator for real numbers, and the **vector** element, which denotes a vector from linear algebra. Each content element corresponds to a carefully defined mathematical concept. Some elements represent mathematical objects like vectors, while others represent functions or operations like addition.

Every MathML element but one is either a presentation element or a content element. The **math** element is neither, since its role is to serve as a top-level, interface element. One function of the **math** element is to pass on parameters to a MathML processor that affect an entire expression, such as style preferences. A second function is to communicate parameters to a Web browser about what software to use to render a MathML expression, and how the expression should be integrated into the surrounding HTML page. (As XML support is added to browsers, it may ultimately be necessary to introduce one or two more interface elements, to handle these functions separately. See chapter 7 for details.)

2.1.2 Expression Trees and Token Elements

Presentation and content expressions both share a number of formal properties. In both cases, most expressions naturally decompose into pieces, or subexpressions. For example, the expression

$$(a + b)^2$$

naturally breaks into a "base," the $(a + b)$, and a "script," which is the single character '2' in this case. Furthermore, as this example shows, the subexpressions may themselves decompose into further subexpressions, and so on. Of course, the decomposition process eventually terminates with indivisible expressions such as digits, letters, or other symbol characters.

Although this particular example involves mathematical notation, and hence presentation markup, the same observation applies equally well to abstract mathematical objects, and hence to content markup. For example, our superscript example would typically denote an exponentiation operation that would require two operands: a "base" and an "exponent." This is no coincidence, since as a general rule, mathematical notation closely mirrors the logical structure of the underlying mathematical objects.

The recursive nature of mathematical objects and notation is strongly reflected in MathML

markup. Most presentation or content elements contain some number of other MathML elements corresponding to the constituent pieces out of which the original object is recursively built. The original schema is commonly called the *parent* schema, and the constituent pieces are called *child* schemata. More generally, MathML expressions can be regarded as trees, where each node corresponds to a MathML element, the branches under a "parent" node correspond to its "children", and the leaves in the tree correspond to indivisible notation or content units such as numbers, characters, etc.

Most leaf nodes in a MathML expression tree are either canonically *empty elements*, or *token elements*. Canonically empty elements directly represent symbols in MathML, such as the content element **plus**. MathML token elements are the only MathML elements permitted to directly contain character data. The character data may consist of ASCII characters and MathML *entities*, which are escape sequences of the form &*entity_name*;. MathML entities typically denote non-ASCII Unicode characters such as α, → and ∑. A third kind of leaf node permitted in MathML is the **annotation** element, which is used to hold data in a non-MathML format.

The most important presentation token elements are **mi**, **mn** and **mo** for representing identifiers, numbers and operators respectively. Typically a renderer will employ slightly different typesetting styles for each of these kinds of character data: numbers are usually in upright font, identifiers in italics, and operators have extra space around them. In content markup, there are only two tokens, **ci** and **cn** for identifiers and numbers respectively. In content markup, separate elements are provided for commonly used functions and operators. The **fn** element is provided for user-defined extensions to the base set.

In terms of markup, most MathML elements have a *start* tag and an *end* tag, which enclose the markup for their contents. In the case of tokens, the content is character data, and in most other cases, the content is the markup for child elements. A third category of elements, called canonically empty elements, don't require any contents, and are marked up using a single tag of the form <element_name/>. An example of this kind of markup is the content element **<plus/>**.

Returning to the example of $(a + b)^2$, we can now see how the principles discussed above play out in practice. One form of presentation markup for this example is:

```
<msup>
  <mfenced>
    <mrow>
      <mi>a</mi>
      <mo>+</mo>
      <mi>b</mi>
    </mrow>
  </mfenced>
  <mn>2</mn>
</msup>
```

The content markup for the same example is:

```
<apply>
  <power/>
  <apply>
    <plus/>
    <ci>a</ci>
    <ci>b</ci>
  </apply>
  <cn>2</cn>
</apply>
```

While a full discussion of presentation and content markup must wait until Chapters 3 and 4, the main features of these sample encodings should now be relatively clear.

2.1.3 Presentation Markup

MathML presentation markup consists of 28 elements which accept over 50 attributes. Most of the elements correspond to *layout schemata*, which contain other presentation elements. Each layout schema corresponds to a 2-dimensional notational device, such as a super- or sub-script, fraction or table. In addition, there are the presentation token elements **mi**, **mn** and **mo** introduced above, as well as several other less commonly used token elements. The remaining few presentation elements are empty elements, and are used mostly in connection with alignment.

The layout schemata fall into several classes. One group of elements is concerned with scripts, and contains elements such as **msub**, **munder**, and **mmultiscripts**. Another group focuses on more general layout and includes **mrow**, **mstyle**, and **mfrac**. A third group deals with tables. The **maction** element is a category by itself, and represents various kinds of actions on notation, such as in an expression which toggles between two pieces of notation.

An important feature of many layout schemata is that the order of child schemata is significant. For example, the first child of an **mfrac** element is the numerator and the second child is the denominator. Since the order of child schemata is not enforced at the XML level by the MathML DTD, the information added by ordering is only available to a MathML processor, as opposed to a generic XML processor. When we want to emphasize that a MathML element such as **mfrac** requires children in a specific order, we will refer to them as *arguments*, and think of the **mfrac** element as a notational "constructor".

2.1.4 Content Markup

Content markup consists of about 75 elements accepting roughly a dozen attributes. The majority of these elements are empty elements corresponding to a wide variety of operators, relations and named functions. Examples of this sort include **partialdiff**, **leq** and **tan**. Others such as **matrix** and **set** are used to encode various mathematical data types, and a third,

important category of content elements such as **apply** are used to make new mathematical objects from others.

The **apply** element is perhaps the single most important content element. It is used to apply a function to a collection of arguments. The positions of the child schemata is again significant, with the first child denoting the function to be applied, and the remaining children denoting the arguments of the function, with order preserved. Note that the apply construct always uses prefix notation, like the programming language LISP. In particular, even binary operations like subtraction are marked up by applying a prefix subtraction operator to two arguments. For example, *a* - *b* would be marked up as

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

A number of functions and operations require one or more quantifiers to be well-defined. For example, in addition to an integrand, a definite integral must specify the limits of integration and the bound variable. For this reason, there are several *qualifier* schemata such as **bvar** and **lowlimit**. They are used with operators such as **diff** and **int**.

The **declare** construct is especially important for content markup that might be evaluated by a computer algebra system. The **declare** element provides a basic assignment mechanism, where a variable can be declared to be of a certain type, with a certain value. Typically, declarations are ignored for visual rendering, and are used when an expression is evaluated.

2.1.5 Mixing Presentation and Content

Different kinds of markup will be most appropriate for different kinds of tasks. Legacy data is probably best translated into pure presentation markup, since semantic information about what the author meant can only be guessed at heuristically. By contrast, some mathematical applications and pedagogically-oriented authoring tools will likely choose to be entirely content-based. However, the majority of applications fall somewhere in between these extremes. For these applications, the most appropriate markup is a mixture of both presentation and content markup.

The rules for mixing presentation and content markup derive from the general principle that mixed content should only be allowed in places where it makes sense. For content markup embedded in presentation markup this basically means that any content fragments should be semantically meaningful, and should not require additional arguments or quantifiers to be fully specified. For presentation markup embedded in content markup, this usually means that presentation markup must be contained in a content token element, so that it will be treated as an indivisible notational unit used as a variable or function name.

Another option is to use a **semantics** element. The **semantics** element is used to bind MathML expressions to various kinds of annotations. One common use for the **semantics** element is to bind a content expression to a presentation expression as a semantic annotation. In this way, an author can specify a non-standard notation be used when displaying a particular content expression. Another use of the **semantics** element is to bind some other kind of semantic specification, such as an OpenMath expression, to a MathML expression. In this way, the **semantics** element can be used to extend the scope of MathML content markup.

2.2 Some MathML Examples

2.2.1 Presentation Examples

Notation: $x^2 + 4x + 4 = 0$

Markup:

```
<mrow>
  <mrow>
    <msup>
      <mi>x</mi>
      <mn>2</mn>
    </msup>
    <mo>+</mo>
    <mrow>
      <mn>4</mn>
      <mo>&InvisibleTimes;</mo>
      <mi>x</mi>
    </mrow>
    <mo>+</mo>
    <mn>4</mn>
  </mrow>
  <mo>=</mo>
  <mn>0</mn>
</mrow>
```

Note the use of nested **mrow** elements to denote terms, in this case the left-hand side of the equation functioning as an operand of "=". Marking terms greatly facilitates things like spacing for visual rendering, voice rendering, and line breaking.

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

Markup:

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

Notice that the plus/minus sign is given by a special named entity ±. MathML

provides a very comprehensive list of entity names for mathematical symbols. In addition to the mathematical symbols needed for screen and print rendering, MathML provides symbols to facilitate audio rendering. For audio rendering, it is important to be able to automatically determine whether

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

should be read as "z times the quantity x plus y" or "z of x plus y". The entities ⁢ and ⁡ provide a way for authors to directly encode the distinction for audio renderers. For instance, in the first case ⁢ should be inserted after the line containing the z. MathML also introduces entities like ⅆ which represents a "differential d" which renders with slightly different spacing in print, and is usually rendered as "with respect to" in speech. Unless content tags, or some other mechanism, are used to eliminate the ambiguity, authors should always use these entities, in order to make their documents more accessible.

Notation: $A = \begin{bmatrix} x & y \\ z & w \end{bmatrix}$

Markup:

```
<mrow>
  <mi>A</mi>
  <mo>=</mo>
  <mfenced open="[" close="]">
    <mtable>
      <mtr>
         <mtd><mi>x</mi></mtd>
         <mtd><mi>y</mi></mtd>
      </mtr>
      <mtr>
         <mtd><mi>z</mi></mtd>
         <mtd><mi>w</mi></mtd>
```

```
      </mtr>
    </mtable>
  </mfenced>
</mrow>
```

Most elements have a number of attributes that control the details of their screen and print rendering. For example, there are several attributes for the **mfenced** element that control what delimiters should be used at the beginning and the end of the expression. The attributes for operator elements given using **<mo>** are set to default values determined by a dictionary. (For the suggested MathML operator dictionary, see appendix C.)

2.2.2 Content Examples

Notation: $x^2 + 4x + 4 = 0$

Markup:

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

Note that the **reln** element is used much like the **apply** element, except that it is used with relations instead of operators and functions.

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

Markup:

```
<reln>
  <eq/>
  <ci>x</ci>
  <apply>
    <divide/>
    <apply>
      <fn><mo>&PlusMinus;</mo></fn>
      <apply>
        <minus/>
        <ci>b</ci>
      </apply>
      <apply>
        <root/>
        <apply>
          <minus/>
          <apply>
            <power/>
            <ci>b</ci>
            <cn>2</cn>
          </apply>
          <apply>
            <times/>
            <cn>4</cn>
            <ci>a</ci>
            <ci>c</ci>
          </apply>
        </apply>
        <cn>2</cn>
      </apply>
    </apply>
    <apply>
      <times/>
      <cn>2</cn>
      <ci>a</ci>
    </apply>
  </apply>
```

```
</reln>
```

MathML content markup does not directly contain an element for the "plus or minus" operation. Therefore, we use the **fn** element to declare that we want the presentation markup for this operator to act as a content operator. This is a simple example of how presentation and content markup can be mixed to extend content markup.

Notation: $A = \begin{pmatrix} x & y \\ z & w \end{pmatrix}$

Markup:

```
<reln>
  <eq/>
  <ci>A</ci>
  <matrix>
    <matrixrow>
      <ci>x</ci>
      <ci>y</ci>
    </matrixrow>
    <matrixrow>
      <ci>z</ci>
      <ci>w</ci>
    </matrixrow>
  </matrix>
</reln>
```

Note that by default, the rendering of the content element **matrix** includes enclosing parentheses, so we need not directly encode them. This is quite different from the presentation element **mtable** which may or may not refer to a matrix, and hence requires explicit encoding of the parentheses if they are desired.

2.2.3 Mixed Markup Examples

Notation: $\int_0^t \frac{dx}{x}$

Markup:

```
<semantics>
```

```
  <mrow>
    <msubsup>
      <mo>&int;</mo>
      <mn>0</mn>
      <mi>t</mi>
    </msubsup>
    <mfrac>
      <mrow>
        <mo>&dd;</mo>
        <mi>x</mi>
      </mrow>
      <mi>x</mi>
    </mfrac>
  </mrow>

  <annotation-xml encoding="MathML-Content">
    <apply>
      <int/>
      <bvar><ci>x</ci></bvar>
      <lowlimit><cn>0</cn></lowlimit>
      <uplimit><ci>t</ci></uplimit>
      <apply>
        <divide/>
        <cn>1</cn>
        <ci>x</ci>
      </apply>
    </apply>
  </annotation-xml>

</semantics>
```

In this example, we use the **semantics** element to provide a MathML content expression to serve as a "semantic annotation" for a presentation expression. The **semantics** element has as its first child the expression being annotated, and the subsequent children are the annotations. There is no restriction on the kind of annotation that can be attached using the **semantics** element. For example, one might give a TeX encoding, or computer algebra input in an annotation. The type of annotation is specified by the **encoding** attribute and the **annotation** and **annotation-xml** elements.

Another common use of the **semantics** element arises when one wants to use a content coding, and provide a suggestion for its presentation. In this case, we would have the markup:

```
<semantics>
```

```
  <apply>
    <int/>
    <bvar><ci>x</ci></bvar>
    <lowlimit><cn>0</cn></lowlimit>
    <uplimit><ci>t</ci></uplimit>
    <apply>
      <divide/>
      <cn>1</cn>
      <ci>x</ci>
    </apply>
  </apply>

  <annotation-xml encoding="MathML-Presentation">
    <mrow>
      <msubsup>
        <mo>&int;</mo>
        <mn>0</mn>
        <mi>t</mi>
      </msubsup>
      <mfrac>
        <mrow>
          <mo>&dd;</mo>
          <mi>x</mi>
        </mrow>
        <mi>x</mi>
      </mfrac>
    </mrow>
  </annotation-xml>

</semantics>
```

This kind of annotation is useful when something other than the default rendering of the content encoding is desired. For example, by default, some renderers might layout the integrand something like "1/*x dx*". Specifying that the integrand should by preference render as "*dx* / *x*" instead can be accomplished with the use of a MathML Presentation annotation as shown. Be aware, however, that renderers are not required to take into account information contained in annotations, and what use is made of them, if any, will depend on the renderer.

2.3 MathML Syntax and Grammar

MathML is an application of XML, or Extensible Markup Language, and as such, its syntax is governed by the rules of XML syntax, and its grammar is in part specified by a DTD, or

Document Type Definition. In other words, the details of using tags, attributes, entity references and so on are defined in the XML language specification, and the details about MathML element and attribute names, which elements can be nested inside each other, and so on are specified in the MathML DTD.

However, 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 complex DTD. A grammar for content markup expressions is given in Appendix E. Of course, one drawback to using MathML specific rules is that they are invisible to generic XML processors and validators.

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. 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 DTD and usage required by MathML specific rules. However, we will frequently allude to "MathML errors" without identifying which part of the specification is being violated.

2.3.1 An XML Syntax Primer

Since MathML is an application of XML, the MathML Specification uses the terminology of XML to describe it. Briefly, XML data is composed of Unicode characters (which include ordinary ASCII characters), "entity references" (informally called "entities") such as " " which usually represent "extended characters", and "elements" such as `<mi fontstyle="normal"> x </mi>`. Elements enclose other XML data called their "content" between a "start tag" (sometimes called a "begin tag") and an "end tag", much like in HTML. There are also "empty elements" such as `<plus/>`, whose start tag ends with `/>` to indicate that the element has no content or end tag. The start tag can contain named parameters called "attributes", such as `fontstyle="normal"` in the example above. For further details on XML, consult the XML specification.

As XML is case-sensitive, MathML element and attribute names are case-sensitive. For reasons of legibility, the MathML defines them almost all in lowercase.

In formal discussions of XML markup a distinction is maintained between an element, such as an **mrow** element, and the tags **<mrow>** and **</mrow>** marking it. What is between the **<mrow>** start tag and the **</mrow>** end tag is the **mrow** element's content. An "empty element" such as **none** is defined to have no content and so has a single tag of the form

<none/>. Usually, the distinction between elements and tags will not be so finely drawn in this specification. For instance, we will often refer to the **<mrow>** and **<none/>** elements, really meaning the elements whose tags these are, in order that references to elements are visually distinguishable from references to attributes. However, the words "element" and "tag" themselves will be used strictly in accordance with XML terminology.

2.3.2 Children vs. Arguments

Many MathML elements require a specific number of child elements and/or attach additional meanings to children in certain positions. As noted above, these kinds of requirements are MathML specific, and cannot be specified entirely in terms of 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 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, and for content elements in the EBNF grammar for content markup in appendix E.

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

2.3.3 MathML Attribute Values

According to the XML language specification, attributes given to elements must have one of the forms

```
attribute-name = "value"
```

or

```
attribute-name = 'value'
```

where whitespace around the '=' is optional.

Attribute names are generally shown in **bold** within descriptive text in this specification, but not within examples.

The attribute 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 quotes not used to surround the value may be included within it.

MathML uses a more complicated syntax for attribute values than the generic XML syntax required by the MathML DTD. These additional rules are intended for use by MathML applications, and it is a MathML error to violate them, though they are not 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 it can be used with, using a notation described below. In MathML applications these attribute values should be further processed as follows, unless otherwise specified: whitespace is ignored except to separate letter and/or digit sequences into individual words or numbers; and the same entity references (listed in Chapter 6) which can be used within token elements to represent characters can be used to represent those characters in attribute values (whenever those characters would be permitted by that attribute value's syntax).

In particular, the characters `"`, `'`, and `&` can be included in MathML attribute values (when permitted by the attribute value syntax) using the entity references `&quot;`, `&apos;`, and `&`, respectively. (`<` can also be used for `<`, but this is not required in attribute values, only in token element content.)

The MathML DTD provided in Appendix A declares most attribute value types as CDATA strings. This permits increased interoperability with existing SGML software and allows extension to the lists of predefined values.

Syntax notations used in the MathML specification

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

Notation	**what it matches**
number	decimal integer or real number (digits with one decimal point), optionally starting with '-'
unsigned-number	decimal integer or real number, no sign
integer	decimal integer, optionally starting with '-'
positive-integer	decimal integer, unsigned, not 0
string	arbitrary string (always the entire attribute value)
character	single non-whitespace character, or MathML entity reference; whitespace separation is optional
#rgb	RGB color value
#rrggbb	RGB color value
h-unit	unit of horizontal length (allowable units are listed below)
v-unit	unit of vertical length (allowable units are listed below)
css-fontfamily	explained in CSS subsection, below
html-color-name	explained in 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*]	optional instance of *form*
(*form*)	same as *form*
word in plain text	that word, literally present in attribute value (unless it is obviously part of an explanatory phrase)
quoted symbol	that symbol, literally present in 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; it must use entity references for certain characters, as described earlier. It can contain XML-format entity or character references for any of the characters listed in Chapter 6. No syntax rule in MathML includes *string* as only part of an attribute value, only as the entire value.

A *character* consists of a single non-whitespace character or entity reference.

As a simple example, the permissible values of boolean attributes are specified as `true | false`, meaning that the entire attribute value should be either "true" or "false".

Adjacent keywords and/or numbers must be separated by whitespace in the actual attribute values, except for unit identifiers (symbolized 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 CSS1) immediately before unit identifiers, between the '-' signs and digits of negative numbers, or between # and *rgb* or *rrggbb*.

Numeric 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 "ems", and vertical dimensions in "exs", 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 "ems", 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 proportionately to the current font size.

For most numeric 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 numeric 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 to provide one, even if 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 numeric 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 *html-color-name* are explained in the following subsections.

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 they are each used in.

The unit identifiers and meanings are taken from CSS1. (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 can't 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	ems (font-relative unit traditionally used for horizontal lengths)
ex	exs (font-relative unit traditionally used for vertical lengths)
px	pixels, or pixel size of a "typical computer 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 default value

The typesetting units *em* and *ex* are defined in the Glossary, and discussed further under "Additional notes", below.

% is a "relative unit"; when an attribute value is given as "*nnn*%" (for any numeric value *nnn*), the value being specified is the default value for the property being controlled multiplied by *nnn* 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 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 it applies to, e.g. *number* `[` *h-unit* `]`. The meaning of specifying no unit is given in the documentation 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 numeric 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 the same 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 typical unit is usually `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 <**mtable**>) can contain more than one numeric 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 fontsize 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 fontsize can change the length of these units.

The definition of the length of each unit (but not the MathML syntax for length values) is as specified in CSS1, except that if a font provides specific values for em and/or ex which differ from the values defined by CSS1 (the font size and 'x'-height respectively), those values should be used.

CSS-compatible attributes

Several MathML attributes, listed below, correspond closely with text rendering properties defined by Cascading Style Sheets, Level 1 (CSS1).

The names and acceptable values of these attributes have been aligned with the CSS1 recommendation where possible. In general, the MathML syntax for each attribute is intended to be a subset of the CSS syntax for the corresponding property. Differences in detail, where they exist, are explained with the documentation about each attribute, in the sections of this specification listed in the table.

The syntax of certain attributes is partially specified, in the tables of attribute syntax in this specification, using one of the symbols *css-fontfamily* or *html-color-name*, as shown in the following table. These symbols refer to syntaxes from other W3C Recommendations, and are explained in the sections of this specification referred to in the table.

MathML attribute	CSS property	syntax symbol	MathML elements	refer to
fontsize	font-size	-	presentation tokens; <mstyle>	Section 3.2.1
fontweight	font-weight	-	presentation tokens; <mstyle>	Section 3.2.1
fontstyle	font-style	-	presentation tokens; <mstyle>	Section 3.2.1
fontfamily	font-family	*css-fontfamily*	presentation tokens; <mstyle>	Section 3.2.1
color	color	*html-color-name*	presentation tokens; <mstyle>	Section 3.3.4
background	background	*html-color-name*	<mstyle>	Section 3.3.4

See also Section 2.3.4 below for a discussion of the **class**, **style**, and **id** attributes for use with style sheets.

Order of processing attributes vs. style sheets

CSS or analogous style sheets specify changes to rendering properties of selected MathML elements (selecting the elements in various ways). Either the properties listed above, or any other MathML rendering attributes or properties supported by a style sheet mechanism, can be affected, in principle for any element. Since rendering properties can also be changed by attributes on an element, or automatically (which can happen to **fontsize**, as explained in the discussion on **scriptlevel** in Section 3.3.4), it is necessary to specify the relative order in which changes from various sources occur. 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 CSS1, 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
- style sheet from reader: styles which are *not* declared "important"

- explicit attribute settings on this MathML element
- style sheet from author: styles which are *not* declared "important"
- style sheet from reader: styles which *are* declared "important"
- style sheet from author: styles which *are* declared "important"

  (last changes; highest priority)

Note that the order of the various CSS-style sheet-derived changes is specified by CSS itself. The following rationale is related only to the issue of where in this preexisting 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 CSS1 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.

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.)

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 MathML. 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/or 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.

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 7.2.2 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 Chapters 3 and 4, then Section 2.3, and then other parts of the text.)

2.3.4 Attributes Shared by all MathML Elements

In order to facilitate compatibility with Cascading Style Sheets, Level 1 (CSS1), all MathML elements accept **class**, **style**, and **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 value syntaxes as general strings, even if style sheet mechanisms have more restrictive syntaxes for them. That is, any value for them is valid in MathML.)

Renderers supporting CSS (or analogous style sheet mechanisms) may use these attributes to help determine which MathML elements should be subject to which style sheet-induced changes to various rendering properties. The properties that can be affected, and how these changes affect them, are discussed in the subsection CSS-compatible attributes in Section 2.3.3 above.

Every MathML element also accepts the attribute **other** (Section 7.2.3) 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.

See also Section 3.2.1 for a list of MathML attributes which can be used on most presentation token elements.

2.3.5 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), and "collapsed" internally (i.e. each sequence of 1 or more whitespace characters is replaced with one blank character).

In MathML, as in XML, "whitespace" means blanks, 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 blank, without having them ignored, must use   or other "whitespace" non-marking entities as described in section 6.2.1. For example, compare

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

with

```
<mtext>
 Theorem&NewLine;  1:
</mtext>
```

When the first example is rendered, there is no whitespace before "Theorem", one blank between "Theorem" and "1:", and no whitespace after "1:". In the second example, a single blank is rendered before "Theorem", a new line is placed after "Theorem", two blanks 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.

Next: Presentation Markup -- Introduction
Up: Table of Contents

3. Presentation Markup

- 3.1 Introduction
  - 3.1.1 What Presentation Elements Represent
  - 3.1.2 Terminology Used In This Chapter
  - 3.1.3 Required Arguments
  - 3.1.4 Elements with Special Behaviors
  - 3.1.5 Summary of Presentation Elements
- 3.2 Token Elements
  - 3.2.1 Attributes common to token elements
  - 3.2.2 **<mi>** -- identifier
  - 3.2.3 **<mn>** -- number
  - 3.2.4 **<mo>** -- operator, fence, or separator
  - 3.2.5 **<mtext>** -- text
  - 3.2.6 **<mspace/>** -- space
  - 3.2.7 **<ms>** -- string literal
- 3.3 General Layout Schemata
  - 3.3.1 **<mrow>** -- horizontally group any number of subexpressions
  - 3.3.2 **<mfrac>** -- form a fraction from two subexpressions
  - 3.3.3 **<msqrt>** and **<mroot>** -- form a radical
  - 3.3.4 **<mstyle>** -- style change
  - 3.3.5 **<merror>** -- enclose a syntax error message from a preprocessor
  - 3.3.6 **<mpadded>** -- adjust space around content
  - 3.3.7 **<mphantom>** -- make content invisible but preserve its size
  - 3.3.8 **<mfenced>** -- surround content with a pair of fences
- 3.4 Script and Limit Schemata
  - 3.4.1 **<msub>** -- attach a subscript to a base
  - 3.4.2 **<msup>** -- attach a superscript to a base
  - 3.4.3 **<msubsup>** -- attach a subscript-superscript pair to a base

    - 3.4.4 <**munder**> -- attach an underscript to a base
    - 3.4.5 <**mover**> -- attach an overscript to a base
    - 3.4.6 <**munderover**> -- attach an underscript-overscript pair to a base
    - 3.4.7 <**mmultiscripts**> -- attach prescripts and tensor indices to a base
- 3.5 Tables and Matrices
    - 3.5.1 <**mtable**> -- table or matrix
    - 3.5.2 <**mtr**> -- row in a table or matrix
    - 3.5.3 <**mtd**> -- one entry in a table or matrix
    - 3.5.4 <**maligngroup/**> and <**malignmark/**> -- alignment markers
- 3.6 Enlivening Expressions
    - 3.6.1 <**maction**> -- bind actions to a subexpression

3.1 Introduction

This chapter specifies the "presentation" elements of MathML, which can be used to describe the layout structure of mathematical notation. It is strongly recommended that one read Section 2.3 on MathML syntax and grammar before reading Chapter 3. Section 2.3 contains important information on MathML notations and conventions which are necessary for understanding some of the material in this chapter.

3.1.1 What Presentation Elements Represent

Presentation elements correspond to the "constructors" of traditional math notation -- that is, to the basic kinds of symbols and expression-building structures out of which any particular piece of traditional math notation is built. They are designed to be medium-independent, in the sense that there are sensible ways to render them in audio, as well as in traditional visual media for math. Because of the importance of traditional visual notation, the descriptions of which notational constructs the elements represent, and how they are typically rendered, is often given here in visual terms. However, the elements have been designed to contain enough information for good spoken renderings as well, provided the conventions described herein for their proper use are followed. 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).

One major anticipated use of MathML is to describe mathematical expressions within HTML documents, using multiple MathML expressions embedded in some manner in an HTML document. Note that HTML in general describes logical structures such as headings, paragraphs, etc. but only suggests (i.e. does not require) specific ways of rendering various

logical parts of the document, in order to allow for medium-dependent rendering and for individual preferences of style; MathML presentation elements are fully compatible with this philosophy. 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 math 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 subexpressions 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 math, 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 extended characters, represented by entity references, are used to name operators or identifiers which in traditional notation render the same as other symbols, such as "ⅆ", "ⅇ", or "ⅈ", or operators which usually render invisibly, such as "⁢", "⁡", 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

The MathML specification uses a number of technical terms to describe MathML-specific rules and conventions. The most notable example is the attribute value notations and conventions described in Section 2.3.3. (See also the brief description of XML terminology in Section 2.3.1.)

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

Types of presentation elements

The presentation elements are divided into two classes. *Token elements* represent individual symbols, names, numbers, labels, etc. and can have only characters and entity references (or the vertical alignment element <**malignmark**/>) as content. *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 which 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 subexpressions are built into larger expressions. The terminology derives from the fact that each layout schema corresponds to a different way of "laying out" its subexpressions to form a larger expression in traditional mathematical typesetting.

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 the chapter. These elements are listed in Section 3.1.5. The *content elements* are the MathML elements defined in chapter 4. The content elements are listed in Section 4.4.

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 (listed in Section 5.2.2). A *subexpression* of an expression *E* is any MathML expression which is part of the content of *E*, whether *directly* or *indirectly*, i.e. whether it is a "child" of *E* or not.

A child of a layout schema is also called an *argument* of that element. Token elements have no arguments, by definition, even though they can contain the **<malignmark/>** element; this means that a **<malignmark/>** element in a token is not an argument, whereas in a layout schema it is one.

As a consequence of the above definitions, the content of a layout schema consists exactly of a sequence of zero or more nonoverlapping elements which are its arguments (possibly with intervening whitespace, which is ignored in MathML). Note that an argument is almost always a subexpression; the only exceptions are the empty elements **<none/>** and **<mprescripts/>** which are allowed only as special arguments of the **<mmultiscripts>** element, but are not subexpressions because they are not MathML expressions as defined above.

Descriptions of presentation elements

Each MathML presentation element is described below in detail. The description starts with the information needed by authors of MathML (or of programs which generate MathML). The intended use of each element is described, along with the argument syntax it accepts. (There is also a table of argument count requirements and argument roles in Section 3.1.3.) The valid attributes, along with their permissible and default values, are listed, and the effect of each attribute is discussed.

For certain elements, further information of interest mainly to those implementing MathML renderers is given in a subsection. This includes many details of one suggested set of rendering rules which can be used to render MathML expressions in a manner reminiscent of traditional visual notation.

3.1.3 Required Arguments

Many of the elements described herein require a specific number of arguments (always 1, 2, or 3). Recall that MathML uses the term *argument* to describe a child element with additional MathML-specific requirements, usually related to which position it occupies in its parent.

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. The descriptions, interpreted according to the convention just stated, fully specify the allowed numbers of arguments for every element defined in this Chapter. 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 0 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 they appear in. See Section 3.2.6 for a discussion of the proper use of such spacelike elements.

Inferred <mrow>s

The elements listed in the following table as requiring exactly 1 argument (<**msqrt**>, <**mstyle**>, <**merror**>, <**mpadded**>, <**mphantom**>, and <**mtd**>) actually accept any number of arguments, but 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.

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 which 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 exactly 1, since they are most naturally understood as acting on

a single expression.

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. Recall that a required argument count of 1 may indicate an inferred **<mrow>**.

Element	**Required argument count** (and 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
<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 (**<mtr>**s, inferred if necessary)
<mtr>	0 or more table elements (**<mtd>**s, inferred if necessary)
<mtd>	1
<maction>	1 or more (argument roles depend on **actiontype** attribute)

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 spacelike; these are defined in Section 3.2.6. This definition affects some of the suggested rendering rules for **<mo>** elements (Section 3.2.4).

Certain elements (e.g., **<msup>**) are able to embellish operators which are their first argument. These elements are listed in Section 3.2.4, 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.

The **<mtable>** element can infer **<mtr>**s around its arguments, and the **<mtr>** element can infer **<mtd>**s, as explained in the sections about those elements.

3.1.5 Summary of Presentation Elements

Token Elements:

<mi>	identifier
<mn>	number
<mo>	operator, fence, or separator
<mtext>	text
<mspace/>	space
<ms>	string literal

General Layout Schemata:

<mrow>	group any number of subexpressions horizontally
<mfrac>	form a fraction from two subexpressions
<msqrt>	form a square root sign (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

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

Tables and Matrices:

<mtable>	table or matrix
<mtr>	row in a table or matrix
<mtd>	one entry in a table or matrix
<maligngroup/> and **<malignmark/>**	alignment markers

Enlivening Expressions:

<maction>	bind actions to a subexpression

Next: Presentation Markup -- Token Elements
Up: Table of Contents

4. Content Markup

- 4.1 Introduction
  - 4.1.1 The Intent of Content Markup
  - 4.1.2 The Scope of Content Markup
  - 4.1.3 Basic Concepts of Content Markup
- 4.2 Content Element Usage Guide
  - 4.2.1 Overview of Syntax and Usage
  - 4.2.2 Containers
  - 4.2.3 Functions, Operators and Qualifiers
  - 4.2.4 Relations
  - 4.2.5 Conditions
  - 4.2.6 Syntax and Semantics
  - 4.2.7 Semantic Mappings
  - 4.2.8 MathML element types
- 4.3 Content Element Attributes
  - 4.3.1 Content Element Attribute Values
  - 4.3.2 Attributes Modifying Content Markup Semantics
  - 4.3.3 Attributes Modifying Content Markup Rendering
- 4.4 The Content Markup Elements
  - 4.4.1 Token Elements
  - 4.4.2 Basic Content Elements
  - 4.4.3 Arithmetic, Algebra and Logic
  - 4.4.4 Relations
  - 4.4.5 Calculus
  - 4.4.6 Theory of Sets
  - 4.4.7 Sequences and Series
  - 4.4.8 Trigonometry
  - 4.4.9 Statistics
  - 4.4.10 Linear Algebra
  - 4.4.11 Semantic Mapping Elements

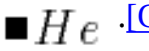
4.1 Introduction

4.1.1 The Intent of Content Markup

As has been noted in the introductory section of this report, mathematics can be distinguished by its use of a (relatively) formal language, mathematical notation. However, mathematics and its presentation should not be viewed as one and the same thing. Mathematical sums or products exist and are meaningful to many applications completely without regard to how they are rendered aurally or visually. The intent of the content markup in Mathematical Markup Language is to provide an explicit encoding of the *underlying mathematical structure* of an expression, rather than any particular rendering for the expression.

There are many reasons for providing a specific encoding for content. Even a disciplined and systematic use of presentation tags cannot properly capture this semantic information. This is because without additional information it is impossible to decide if a particular presentation was chosen deliberately to encode the mathematical structure or simply to achieve a particular visual or aural effect. Furthermore, an author using the same encoding to deal with both the presentation and mathematical structure might find a particular presentation encoding unavailable simply because convention had reserved it for a different semantic meaning.

The difficulties stem from the fact that there are many to one mappings from presentation to semantics and vice versa. For example the mathematical construct "H multiplied by e" is often encoded using an explicit operator as in H ***** e. In different presentational contexts, the multiplication operator might be invisible "H e" , or rendered as the spoken word "times". Generally, many different presentations are possible depending on the context and style preferences of the author or reader. Thus, given "H e" out of context it may be impossible to decide if this is the name of a chemical or a mathematical product of two variables H and e.

Mathematical presentation also changes with culture and time: some expressions in combinatorial mathematics today have one meaning to an English mathematician, and quite another to a French mathematician. Notations may lose currency, for example the use of musical sharp and flat symbols to denote maxima and minima. [Chaudry 1954] A notation in use in 1644 for the multiplication mentioned above was  .[Cajori, 1928/1929]

When we encode the underlying mathematical structure explicitly, without regard to how it is presented aurally or visually, we are able to interchange information more precisely with those systems which are able to manipulate the mathematics. In the trivial example above, such a system could substitute values for the variables H and e and evaluate the result. Further interesting application areas include interactive textbooks and other teaching aids.

4.1.2 The Scope of Content Markup

The semantics of general mathematical notation is not a matter of consensus. It would be an enormous job to systematically codify most of mathematics - a task which can never be complete. Instead, MathML makes explicit a relatively small number of commonplace mathematical constructs, chosen carefully to be sufficient in a large number of applications. In addition, it provides a mechanism for associating semantics with new notational constructs. In this way, mathematical concepts that are not in the base collection of tags can still be encoded (see section 4.2.6).

The base set of content elements are chosen to be adequate for simple coding of most of the formulas used

from kindergarten to the end of high school in the United States, and probably beyond through the first two years of college, that is up to A-Level or Baccalaureate level in Europe. Subject areas covered to some extent in MathML are:

- Arithmetic, Algebra, Logic and Relations
- Calculus
- Set Theory
- Sequences and Series
- Trigonometry
- Statistics
- Linear Algebra

It is not claimed, or even suggested, that the proposed element set is complete for these areas, but the provision for author extensibility greatly alleviates any problem which omissions from this finite list might cause.

4.1.3 Basic Concepts of Content Markup

The design of the MathML content elements are driven by the following principles:

- The expression tree structure of a mathematical expression should be directly encoded by the MathML content elements.
- The encoding of an expression tree should be explicit, and not dependent on the special parsing of CDATA or on additional processing such as operator precedence parsing.
- The basic set of mathematical content constructs that are provided should have default mathematical semantics.
- There should be a mechanism for associating specific mathematical semantics with the constructs.

The primary goal of the content encoding is to establish explicit connections between mathematical structures and their mathematical meanings. The content elements correspond directly to parts of the underlying mathematical expression tree. Each structure has an associated default semantics and there is a mechanism for associating new mathematical definitions with new constructs.

Significant advantages to the introduction of content specific tags include:

- Presentation element usage is less constrained. When mathematical semantics are inferred from presentation markup, processing agents must either be quite sophisticated, or they run the risk of inferring incomplete or incorrect semantics when irregular constructions are used to achieve a particular aural or visual effect.
- It is immediately clear which kind of information is being encoded simply by the kind tags which are used.
- Combinations of semantic and presentation tags can be used to convey both the appearance and its mathematical meaning much more effectively than simply trying to infer one from the other.

Expressions described in terms of content elements must still be rendered. For common expressions, default visual presentations are usually clear. "Take care of the sense and the sounds will take care of themselves" wrote Lewis Carroll [Carroll 1871]. Default presentations are included in the detailed description of each element occurring in section 4.4.

To accomplish these goals, the MathML content encoding is based on the concept of an expression tree. A

content expression tree is constructed from a collection of more primitive objects, referred to herein as *containers* and *operators*. MathML possesses a rich set of predefined container and operator objects, as well as constructs for combining containers and operators in mathematically meaningful ways. The syntax and usage of these content elements and constructions is described in the next section.

4.2 Content Element Usage Guide

Since the intent of MathML content markup is to encode mathematical expressions in such a way that the mathematical structure of the expression is clear, the syntax and usage of content markup must be consistent enough to facilitate automated semantic interpretation. There must be no doubt when, for example, an actual sum, product or function application is intended and if specific numbers are present there must be enough information present to reconstruct the correct number for purposes of computation. Of course, it is still up to a MathML-compliant processor to decide what is to be done with such a content based expression, and computation is only one of many options. A renderer or a structured editor might simply use the data and its own built-in knowledge of mathematical structure to render the object. Alternatively, it might manipulate the object to build a new mathematical object. A more computationally oriented system might attempt carry out the indicated operation or function evaluation.

The purpose of this section is to describe the intended, consistent usage. The requirements involve more than just satisfying the syntactic structure specified by an XML DTD. Failure to conform to the usage as described below will result in a MathML error, even though the expression may be syntactically valid according to the DTD.

In addition to the usage information contained in this section, section 4.4 gives a complete listing of each content element, providing reference information about about their attributes, syntax, examples and suggested default semantics and renderings. An informal EBNF grammar describing the syntax for the content markup is given in appendix E.

4.2.1 Overview of Syntax and Usage

MathML content encoding is based on the concept of an expression tree. 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.

The MathML content elements can be grouped into the following categories based on their usage:

- Containers
- Operators
- Qualifiers
- Relations
- Conditions
- Semantic

These are the building blocks out of which MathML content expressions are constructed. Each category is

discussed in a separate section below. In the remainder of this section, we will briefly introduce some of the most common elements of each type, and consider the general constructions for combining them in mathematically meaningful ways.

4.2.1.1 Constructing Mathematical Objects

Content expression trees are built up from basic mathematical objects. At the lowest level, "leaf nodes," are encapsulated in non-empty elements that define their type. Numbers and symbols are marked by the *token* elements **cn** and **ci**. More elaborate constructs such as sets, vectors and matrices are also marked using elements to denote their types, but rather than containing data directly, these *container* elements are constructed out of other elements. Elements are used in order to clearly identify the underlying objects. In this way, standard XML parsing can be used and attributes can be used to specify global properties of the objects.

The containers such as **<cn>**12345**</cn>** and **<ci>**x**</ci>**, represent mathematical numbers and variables. Below, we will look at *operator* elements such as **<plus/>** or **<sin/>**, which provide access to the basic mathematical operations and functions applicable to those objects. Additional containers such as **<set>**...**</set>** for sets, and **<matrix>**...**</matrix>** for matrices are provided for representing a variety of common compound objects.

For example, the number 12345 is encoded as

<cn>12345**</cn>**

The attributes and CDATA content together provide the data necessary for an application to parse the number. For example, a default base of 10 is assumed, but to communicate that the underlying data was actually written in base 8, simply set the "base" attribute to 8 as in

<cn base="8">12345**</cn>**

while complex number 3 + 4 *i* can be indicated as

<cn type="complex">3**<sep/>**4**</cn>**

Such information makes it possible for another application to easily parse this into the correct number.

As another example, the scalar symbol **v** is encoded as

<ci>v**</ci>**

By default **ci** elements represent elements from a commutative field (see Appendix F.) If a vector is intended then this fact can be encoded as

<ci type="vector">v**</ci>**

This invokes default semantics associated with the **vector** element, namely an arbitrary element of a finite dimensional vector space.

By using the **ci** element we have made clear that we are referring to a mathematical symbol but this does not say much about how it is rendered. By default a symbol is rendered as if the **ci** element were actually the presentation element **mi** (see section 3.2.2). The actual rendering of a mathematical symbol can be made as elaborate as necessary simply by using the more elaborate presentational constructs (as described in chapter 3) in the body of the **ci** element.

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 CDATA by providing explicit **mn** tags. This is described in detail in section 4.4 .

The issues for compound objects such as sets, vectors and matrices are all similar to those outlined above for numbers and symbols. Each such object has global properties as a mathematical object that impact how they are to be parsed. This may affect everything from the interpretation of operations that are applied to them through to how to render the symbols representing them. These mathematical properties are captured by setting attribute values.

4.2.1.2 Constructing General Expressions

The notion of constructing a general expression tree is essentially that of applying an operator to sub-objects. For example, the sum *a* + *b* can be thought of as an application of the addition operator to two arguments *a* and *b*. In MathML, elements are used for operators for much the same reason that elements are used to contain objects. They are recognized at the XML parse level and their attributes can be used to record or modify the intended semantics. For example, with the MathML **plus** element, setting the type attribute to **vector** as in <**plus type="vector"**/> can communicate that the intended operation is vector based.

There is also another reason for using elements to denote operators. There is a crucial semantic distinction between the function itself and the expression resulting from applying that function to zero or more arguments which must be captured. 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.

MathML contains many pre-defined operator elements, covering a range of mathematical subjects. However, an important class of expressions involve unknown or user-defined functions. For these situations, MathML provide a general **fn** element, which is discussed below.

4.2.1.3 The apply construct

The most fundamental way of building up a mathematical expression in MathML content markup is the **apply** construct. An **apply** element typically applies an operator to its arguments. It corresponds to a complete mathematical expression. Roughly speaking, this means a piece of mathematics which could be surrounded by parentheses or "logical brackets" without changing its meaning.

For example, (*x* + *y*) might be encoded as

```
<apply><plus/> <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 the described as:

```
<apply> op a b </apply>
```

where the *operands* a and b are containers or other content-based elements themselves, and *op* is an operator or function. Note that since **apply** is a container, this allows **apply** constructs to 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><plus/>
  <ci> a </ci>
  <ci> b </ci>
```

```
  <ci> c </ci>
</apply>
```

Mathematical expressions involving a mixture of operations result in nested occurrences of **apply**. For example, *ax* + *b* would be encoded as

```
<apply><plus/>
  <apply><times/>
    <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><times/>
  <apply><plus/>
    <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><plus/> <ci> F </ci> <ci> G </ci> </apply>
```

and applying it to the argument<**ci**> x </**ci**> as in

```
<apply>
  <apply><plus/>
    <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.

Another construction closely related to the use of the **apply** with operators and arguments involves the **reln** element. The **reln** element is used to denote that a mathematical relation holds between its arguments, as opposed to applying an operator. Thus, the MathML markup for the expression *x* < *y* is given by:

```
<reln>
  <lt/>
  <ci> x </ci>
```

```
  <ci> y </ci>
</reln>
```

4.2.1.4 Explicitly defined functions and the fn construct

The most common operations and functions such as <**plus**/> and <**sin**/> have been predefined explicitly as empty elements (see section section 4.4). They have type and definition URL attributes, and by changing these attributes, the author can record that a different sort of algebraic operation is intended. This allows essentially the same notation to be re-used for a discussion taking place in a different algebraic domain.

Due to the nature of mathematics the notation must be extensible. The key to extensibility is the ability of the user to define new functions.

It is always possible to apply arbitrary expressions as if they were functions and to infer their functional properties directly from that usage as was done in the previous section. However such an approach would preclude being able to encode the fact that the construct was a function or to record its mathematical properties except by actually using it. The **fn** element is used as a container to construct an actual function object in much the same way that **ci** is used to construct a symbol.

To record the fact that *F*+*G* is being used semantically as if it were a function, encode it as:

```
<fn>
  <apply><plus/>
    <ci>F</ci>
    <ci>G</ci>
  </apply>
</fn>
```

Its intended semantic role (as a function) has now been indicated. Furthermore, the **definitionURL** attribute of the **fn** can now be used to point to a written definition of such a function as in

```
<fn definitionURL="http://www.defs.org/function_spaces.html#my_def">
  <apply><plus/>
    <ci>F</ci>
    <ci>G</ci>
  </apply>
</fn>
```

This would be important information to any application wanting to evaluate or simplify such an expression according to systematic rules provided by an algebra of functions.

To indicate that a matrix is being used as an operator encode it as

```
<fn>
  <matrix>
    <matrixrow>
      <ci> a </ci>
      <ci> b </ci>
    </matrixrow>
    <matrixrow>
      <ci> c </ci>
```

```
      <ci> d </ci>
    </matrixrow>
  </matrix>
</fn>
```

A common usage of **fn** is to describe a completely new function. The **definitionURL** attribute can then be used to refer explicitly to the mathematical definition. An example of such a construct is:

<**fn definitionURL="mydefs.html#NewG"**> <**ci**>NewG</**ci**> </**fn**>

The **definitionURL** attribute specifies a URL which provides a written definition for the NewG. Suggested default definitions for pre-defined MathML content elements appear in Appendix F in a format based on OpenMath, although there is no requirement that a particular format be used. The role of the **definitionURL** attribute is very similar to the role of definitions included at the beginning many mathematical papers, and which often just refer to a definition used by a particular book.

4.2.1.5 The inverse construct

Given functions, it is natural to have functional inverses. This is handled by the **inverse** element.

Functional inverses can be problematic from a mathematical point of view in that it implicitly involves the definition of an inverse for an arbitrary function *F*. Even at the K through 12 level the concept of an inverse F^{-1} of many common functions *F* is not used in a uniform way. For example, the definitions used for the inverse trigonometric functions may differ slightly depending on the choice of domain and/or branch cuts.

MathML adopts the view:

> If *F* is a function from a domain *D* to *D'*, then the inverse *G* of *F* is a function over *D'* such that *G(F(x)) = x* for *x* in *D*.

This definition does not assert that such an inverse exists for all or indeed any *x* in *D*, or that it is single-valued anywhere. Also, depending on the functions involved, additional properties such as *F(G(y)) = y* for *y* in *D'* may hold.

The **inverse** element is applied to a function whenever an inverse is required. For example, application of the inverse sine function to x (i.e., $sin^{(-1)}(x)$ is encoded as:

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

While **arcsin** is one of the predefined MathML functions, and explicit reference to $sin^{(-1)}(x)$ might occur in a document discussing possible definitions of **arcsin**.

4.2.1.6 The declare construct

Consider a document discussing the vectors **A** = (a,b,c) and **B** = (d,e,f) and later including the expression **V** = **A** + **B**. It is important to be able communicate the fact that wherever **A** and **B** are used they represent a particular vector. The properties of that vector may determine aspects of operators such as **plus**.

The simple fact that **A** is a vector can be communicated by using the tagging

<ci type="vector">A</ci>

but this still does not communicate, for example, which vector is involved or its dimensions.

The **declare** construct is used to associate specific properties or meanings with an object. The actual declaration itself is not rendered visually (or in any other form). However, it indirectly impacts the semantics of all affected uses of the declared object.

The scope of a declaration is, by default, local to the MathML element in which the declaration is made. If the **scope** attribute of the **declare** element is set to "global", the declaration applies to the entire MathML expression in which it appears.

The uses of the **declare** element range from resetting default attribute values to associating an expression with a particular instance of of a more elaborate structure. Subsequent uses of the original expression (within the scope of the declare) play the same semantic role as would the paired object.

For example, the declaration

```
<declare>
  <ci> A </ci>
  <vector>
    <ci> a </ci>
    <ci> b </ci>
    <ci> c </ci>
  </vector>
</declare>
```

specifies that **A** stands for the particular vector (a,b,c) so that subsequent uses of **A** as in **V** = **A** + **B** can take this into account. When **declare** is used in this way, the actual encoding

```
<apply><eq/>
  <ci> V </ci>
  <apply><plus/>
    <ci> A </ci>
    <ci> B </ci>
  </apply>
</apply>
```

remains unchanged but the expression can be interpreted properly as vector addition.

There is no requirement to declare an expression to stand for a specific object. For example, the declaration

```
<declare type="vector">
  <ci> A </ci>
</declare>
```

specifies that **A** is a vector without indicating the number of components or the values of specific components. The possible values for the **type** attribute include all the predefined container element names such as **vector**, **matrix** or **set**. (See 4.3.2.9 type.)

4.2.1.7 The lambda construct

The lambda calculus allows a user to construct a function from a variable and an expression. For example,

the lambda construct underlies the common mathematical idiom illustrated here:

> Let *f* be the function taking *x* to $x^2 + 2$

There are various notations for this concept in mathematical literature, such as *lambda*(*x*, *F*(*x*)) = *F* or *lambda*(*x*, [*F*]) = *F*, where x is a free variable in F.

This concept is implemented in MathML with the **lambda** element. A lambda construct with n internal variables is encoded by a **lambda** element with n + 1 children. All but the last child must be **bvar** elements containing the identifiers of the internal variables. The last is an expression defining the function. This is typically an **apply**, but can also be any container element.

The following constructs lambda(x, sin (x+1)):

```
<lambda>
  <bvar><ci> x </ci></bvar>
  <apply> <sin/>
    <apply><plus/>
      <ci> x </ci>
      <cn> 1 </cn>
    </apply>
  </apply>
</lambda>
```

To use **declare** and **lambda** to construct the function *f* for which $f(x) = x^2 + x + 3$ use:

```
<declare type="fn">
  <ci> f </ci>
  <lambda>
    <bvar><ci> x </ci></bvar>
    <apply><plus/>
      <apply><power/>
        <ci> x </ci>
        <cn> 2 </cn>
      </apply>
      <ci> x </ci>
      <cn> 3 </cn>
    </apply>
  </lambda>
</declare>
```

The following markup declares and constructs the function *J* such that *J(x,y)* = the integral from *x* to *y* of t^4 with respect to *t*.

```
<declare type="fn">
  <ci> J </ci>
  <lambda>
    <bvar><ci> x </ci></bvar>
    <bvar><ci> y </ci></bvar>
    <apply> <int/>
      <bvar>
```

```
          <ci> t </ci>
        </bvar>
        <lowlimit>
          <ci> x </ci>
        </lowlimit>
        <uplimit>
          <ci> y </ci>
        </uplimit>
        <apply> <power/>
          <ci>t</ci>
          <cn>4</cn>
        </apply>
      </apply>
    </lambda>
  </declare>
```

The function *J* can then in turn be applied to an argument pair.

4.2.1.8 The use of qualifier elements and the condition construct

The last example of the preceding section illustrates the use of *qualifier* elements **lowlimit**, **uplimit**, and **bvar** used in conjunction with the **int** element. A number of common mathematical constructions involve additional data which is 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.

Content markup uses qualifier elements in conjunction with a number of operators, including integrals, sums, series, and certain differential operators. Qualifier elements appear in the same **apply** element with one of these operators. In general, they must appear in a certain order, and their precise meaning depends on the operators being used. For details, see section 4.2.3.4.

The **bvar** qualifier element is also used in another important MathML construction. The **condition** element is used to place conditions on bound variables in other expressions. This allows MathML to define sets by rule, rather than enumeration, for example. The following markup, for instance, encodes the set $\{x \mid x < 1\}$:

```
<set>
  <bvar><ci> x </ci></bvar>
  <condition>
    <reln><lt/>
      <ci> x </ci>
      <cn> 1 </cn>
    </reln>
  </condition>
</set>
```

4.2.1.9 Rendering of Content elements

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. Each content element has a default rendering, given in section 4.4. and several mechanisms (including style attributes, declarations and semantics elements) are provided for associating a particular rendering with an

object.

4.2.2 Containers

Containers provide a means for the construction of mathematical objects of a given type.

Tokens	**ci, cn**
Constructors	**interval, list, matrix, matrixrow, set, vector, apply, reln, lambda, fn**
Specials	**declare**

4.2.2.1 Tokens

Token elements are typically the leaves of the MathML expression tree. Token elements are used to indicate numbers and symbols.

It is also possible for the canonically empty operator elements such as **<exp/>**, **<sin/>** and **<cos/>** to be leaves in an expression tree. The usage of of operator elements is described in Section 4.2.3.

cn

The **cn** element is the MathML token element used to represent numbers. The supported types of numbers include: real,integer,rational,complex-cartesian, and complex-polar, with real being the default type. A base attribute (defaulting to base 10) is used to help specify how the content is to be parsed. The content itself is essentially PCDATA, separated by **<sep/>** when two parts are needed in order to fully describe a number. For example, the real number 3 is constructed by **<cn type="real">** 3 **</cn>** while the rational number 3/4 is constructed as **<cn type="rational">** 3 **<sep/>** 4 **</cn>** The detailed structure and specifications are provided in section 4.4.1.1.

ci

The **ci** element, or "content identifier" is used to construct a variables, or symbols. A **type** attribute indicates the type of object the symbol represents. Typically, they represent real scalars, but no default is specified. Their content is either CDATA or a general presentation construct . For example,

```
<ci>
<msub>
  <mi>c</mi>
  <mn>1</mn>
</msub>
</ci>
```

encodes an atomic symbol which displays visually as c_1 which, for purposes of content, is treated as a single symbol representing a real number. The detailed structure and specifications is provided in section 4.4.1.2.

4.2.2.2 Constructors

MathML provides a number of elements for combining elements into familiar compound objects. The compound objects include things like lists, sets. Each constructor produces a new type of object.

interval

The **interval** element is described in detail in section 4.4.2.4. It denotes an interval on the real line with the values represented by its children as end points. The **closure** attribute is used to qualify the type of interval being represented. For example,

```
<interval closure="open-closed">
  <ci> a </ci>
  <ci> b </ci>
</interval>
```

represents the open-closed interval often written (a,b].

set and **list**

The **list** and **set** elements are described in detail in sections 4.4.6.1 and 4.4.6.2.

Typically, the child elements of a possibly empty **list** element are the actual components of an ordered **list**. For example an ordered list of the three symbols **a, b, and c** is encoded as

<list> **<ci>** a **</ci>** **<ci>** b **</ci>** **<ci>** c **</ci>** **</list>**

Alternatively, bvar and **condition** elements can be used to define lists where membership depends on satisfying certain conditions.

An **order** attribute which is used to specify what ordering is to be used. When the nature of the child elements permits, the ordering defaults to a numeric or lexicographic ordering.

Sets are structured much the same as lists except that there is no implied ordering and the **type** of set may be "normal" or "multiset" with "multiset" indicating that repetitions are allowed.

For both sets and lists, the child elements must be valid MathML content elements. The type of the child elements is not restricted. For example, one might construct a list of equations, or inequalities.

matrix and **matrixrow**

The **matrix** element is used to represent mathematical matrices. It is described in detail in section 4.4.10.2. It has zero or more child elements, all of which are **matrixrow** elements. These in turn expect zero or more child elements which evaluate to algebraic expressions or numbers. These sub-elements are often real numbers, or symbols as in

```
<matrix>
  <matrixrow> <cn> 1 </cn> <cn> 2 </cn> </matrixrow>
  <matrixrow> <cn> 3 </cn> <cn> 4 </cn> </matrixrow>
</matrix>
```

The **matrixrow** elements must always be contained inside of a matrix and all **matrixrow**s in a given matrix must have the same number of elements.

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

vector

The **vector** element is described in detail in section 4.4.10.1. It constructs vectors from a n-dimensional vector space so that its ***n*** child elements typically represent real or complex valued scalars as in the three-element vector

```
<vector>
  <apply><plus/>
    <ci> x </ci>
    <ci> y </ci>
  </apply>
  <cn> 3 </cn>
  <cn> 7 </cn>
</vector>
```

apply

The **apply** element is described in detail in section 4.4.2.1. Its purpose is apply a function or operator to its arguments to produce an an expression representing an element of the range of the function. It is involved in everything from forming sums such as $a + b$ as in

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

through to using the sine function to construct sin(a) as in

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

or constructing integrals. Its usage in any particular setting is determined largely by the properties of the function (the first child element) and as such its detailed usage is covered together with the functions and operators in section 4.2.3 Functions, Operators and Qualifiers.

reln

The **reln** element is described in detail in section 4.4.2.2. It is used to construct an expression such as $a = b$, as in

```
<reln><eq/>
  <ci> a </ci>
  <ci> b </ci>
</reln>
```

indicating an intended comparison between two mathematical values.

Such expressions could in principle be regarded as applications of a boolean function, and as such could be constructed using **apply**. They have treated as a special class of expressions in order to better reflect traditional usage.

The actual structure of expressions constructed using **reln** is similar to that for the **apply** element. The use of **reln** is described in 4.2.4 Relations.

fn

The **fn** element is used to identify an expression as a defined function or operator. It is discussed in detail in section 4.4.2.3. The use of **fn** is also described in 4.2.3.3. It differs from the **lambda** element

in that it does not make any attempt to describe how to map the arguments occurring in any application of the function into a new MathML expression. Instead, it depends on its **definitionURL** attribute to point to a particular meaning.

lambda

The **lambda** element is used to construct an user-defined function from a an expression and one or more free variables. The lambda construct with n internal variables takes n + 1 children. The first (second, up to n) is a **bvar** containing the identifiers of the internal variables. The last is an expression defining the function. This is typically an **apply**, but can also be any container element. The following constructs lambda(x, sin x)

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

The following constructs the constant function lambda(x, 3)

```
<lambda>
  <bvar><ci> x </ci></bvar>
  <cn> 3 </cn>
</lambda>
```

4.2.2.3 Special Constructs

The **declare** construct is described in detail in section 4.4.2.8. It is special in that its entire purpose is to modify the semantics of other objects. It is not rendered visually or aurally.

The need for declarations arises any time a symbol (including more general presentations) is being used to represent an instance of an object of a particular type. For example, you may wish to declare that the symbolic identifier **V** represents a vector.

The declaration <**declare type="vector"**><**ci**>**V**</**ci**></**declare**> resets the default type attribute of <**ci**>**V**</**ci**> to **vector** for all affected occurrences of <**ci**>**V**</**ci**>. This avoids having to write <**ci type="vector"**>**V**</**ci**> every time you use the symbol.

More generally, **declare** can be used to associate expressions with specific content. For example, the declaration

```
<declare>
  <ci>F</ci>
  <lambda>
    <bvar><ci> U </ci></bvar>
    <apply><int/>
      <bvar><ci> x </ci></bvar>
      <lowlimit><cn> 0 </cn></lowlimit>
      <uplimit><ci> a </ci></uplimit>
      <ci> U </ci>
```

```
      </apply>
    </lambda>
  </declare>
```

associates the symbol **F** with a new function defined by the **lambda** construct. Within the scope where the declaration is in effect, the expression

```
<apply><ci>F</ci>
  <ci> U </ci>
</apply>
```

stands for the integral of **U** from 0 to **a**.

The **declare** element can also be used to change the definition of a function or operator. For example, if the URL "HTTP://.../MATHML:NONCOMMUTPLUS" described a non-commutative plus operation then the declaration

```
<declare definitionURL="HTTP://.../MATHML:NONCOMMUTPLUS">
<plus/>
</declare>
```

would indicate that all affected uses of **plus** are to be interpreted as having that definition of **plus**.

4.2.3 Functions, Operators and Qualifiers

Table of Operators

unary arithmetic	**exp, factorial, abs, conjugate**
unary logical	**not**
unary functional	**inverse , ident**
unary trigonometric	**sin, cos, tan, sec, csc, cot, sinh, cosh, tanh, sech, csch, coth, arcsin, arccos, arctan**
unary linear algebra	**determinant, transpose**
unary calculus	**ln, log**
binary arithmetic	**quotient, divide, minus, power, rem**
binary logical	**implies**
binary set operators	**setdiff**
n-ary arithmetic	**plus, times, max, min, gcd**
n-ary statistical	**mean, sdev, variance, median, mode**
n-ary logical	**and, or, xor**
n-ary linear algebra	**selector**
n-ary set operator	**union, intersect**
n-ary functional	**fn, compose**
integral, sum, product operator	**int, sum, product**
differential operator	**diff, partialdiff**
quantifier	**forall, exists**

From the point of view of usage, MathML regards functions (eg. *sin*, *cos*) and operators (eg. *plus*, *times*) in the same way. MathML predefined functions and operators are all canonically empty elements

Note: The **fn** element can be used to construct a user-defined function or operator. **fn** is discussed in more detail below.

4.2.3.2 MathML predefined functions and operators

MathML functions can be used in two ways. They can be used as the operator within an **apply** element, in which case they refer to a function evaluated at a specific value. For example,

```
<apply><sin/><cn>5</cn></apply>
```

denotes a real number, namely sin(5).

MathML functions can also be used as arguments to other operators, for example

```
<apply><plus/><sin/><cos/></apply>
```

denotes a function, namely the result of adding the sine and cosine functions in some function space. (The default semantic definition of **plus** is such that it infers what kind of operation is intended from the type of its arguments.)

The number of child elements in the **apply** is defined by the element in the first (i.e. operator) position.

Unary operators are followed by exactly one other child element within the **apply**.

Binary operators are followed by exactly two child elements.

N-ary operators are followed by zero or more child elements.

The one exception to these rules is that **declare** elements may be inserted in any position except the first. **declare** elements are not counted when satisfying the child element count for an **apply** containing a unary or binary operator element.

Integral, sum, product and differential operators are discussed below in section 4.2.3.4 Operators taking Qualifiers.

4.2.3.3 The fn element

In MathML, only functions and operators can be applied to arguments. In order to provide a way of applying functions constructed out of other functions, or functions other than the functions provided by the content elements, MathML provides the **fn** element. The **fn** element accepts any valid MathML expression as content, and allows it to be used as a content function. It is an error for the **fn** element to have no content.

One typical way of using the **fn** element is with author-named functions, such as *f*(5), encoded as:

```
<apply>
  <fn><ci>f</ci></fn>
  <cn> 5 </cn>
</apply>
```

Another common use is to designate the result of combining several functions as a function again: (*sin* + *cos*)(*z*):

```
<apply>
  <fn>
```

```
      <apply>
        <plus/>
        <sin/>
        <cos/>
      </apply>
    </fn>
    <ci>z</ci>
  </apply>
```

4.2.3.4 Operators taking Qualifiers

Table of Qualifiers and Operators taking Qualifiers

qualifiers	**lowlimit, uplimit, bvar, degree, logbase, interval, condition**
operators	**int, sum, product, diff, partialdiff, limit, log, moment, min, max, forall, exists**

Operators taking qualifiers are canonically empty functions which differ from ordinary empty functions only in that they support the use of special "qualifier" elements to specify their meaning more fully. They are used in exactly the same way as ordinary operators, except that when they are used as operators, certain qualifier elements are also permitted to be in the enclosing **apply**. They always precede the argument if it is present. If more than one qualifier is present, they appear in the order **bvar lowlimit uplimit interval condition degree logbase**. A typical example is:

```
<apply>
  <int/>
  <bvar><ci>x</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <uplimit><cn>1</cn></uplimit>
  <apply>
    <power/>
    <ci>x</ci>
    <cn>2</cn>
  </apply>
</apply>
```

It is also valid to use qualifier schema with a function not applied to an argument. For example, a function acting on integrable functions on the interval [0,1] might be denoted:

```
<fn>
  <apply>
    <int/>
    <bvar><ci>x</ci></bvar>
    <lowlimit><cn>0</cn></lowlimit>
    <uplimit><cn>1</cn></uplimit>
  </apply>
</fn>
```

The meaning and usage of qualifier schema varies from function to function. The following list summarizes the usage of qualifier schema with the MathML functions taking qualifiers.

int

The **int** function accepts the **lowlimit**, **uplimit**, **bvar**, **interval** and **condition** schema. If both **lowlimit** and **uplimit** schema are present, they denote the limits of a definite integral.The domain of integartion may alternatively be specified using interval or condition The **bvar** schema signifies the variable of integration. When used with **int**, each qualifier schema is expected to contain a single child schema; otherwise an error is generated.

diff

The **diff** function accepts the **bvar** schema. The **bvar** schema specifies with respect to which variable the derivative is being taken. The **bvar** may itself contain a **degree** schema which is used to specify the order of the derivative, i.e. a first derivative, a second derivative, etc. For example, the second derivative of *f* with respect to *x* is:

```
<apply><diff/>
  <bvar>
    <ci> x </ci>
    <degree>
      <cn> 2 </cn>
    </degree>
  </bvar>
  <apply><fn><ci>f</ci></fn>
    <ci> x </ci>
  </apply>
</apply>
```

partialdiff

The **partialdiff** function accepts zero or more **bvar** schema. The **bvar** schema specify with respect to which variables the derivative is being taken. The **bvar** elements may themselves contain **degree** schema which are used to specify the order of the derivative. Variables specified by multiple **bvar** elements will be used in order as the variable of differentiation in mixed partials. When used with **partialdiff**, the **degree** schema is expected to contain a single child schema. For example,

```
<apply>
  <partialdiff/>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <fn><ci>f</ci></fn>
</apply>
```

denote the mixed partial $(d^2 / dx\, dy)\, f$.

sum, product

The **sum** and **product** functions accept the **bvar**, **lowlimit**, **uplimit**, **interval** and **condition** schema. If both **lowlimit** and **uplimit** schema are present, they denote the limits of the sum/product. The limits may alternatively be specified using the **interval** or **condition** schema. The **bvar** schema signifies the index variable in the sum or product. A typical example might be:

```
<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>
```

When used with **sum** or **product**, each qualifier schema is expected to contain a single child schema; otherwise an error is generated.

limit

The **limit** function accepts zero or more **bvar** schema and optional **condition** and **lowlimit** schema. A **condition** may be used to place constraints on the **bvar**. The **bvar** schema denotes the variable with respect to which the limit is being taken. The **lowlimit** schema denotes the limit point. When used with **limit**, the **bvar** and **lowlimit** schemata are expected to contain a single child schema; otherwise an error is generated.

log

The **log** function accepts only the **logbase** schema. If present, the **logbase** schema denotes the base with respect to which the logarithm is being taken. Otherwise, the log is assumed to be base 10. When used with **log**, the **logbase** schema is expected to contain a single child schema; otherwise an error is generated.

moment

The **moment** function accepts only **degree** schema. If present, the **degree** schema denotes the order of the moment. Otherwise, the moment is assumed to be the first order moment. When used with **moment**, the **degree** schema is expected to contain a single child schema; otherwise an error is generated.

min, **max**

The **min** and **max** functions accept a **bvar** schema in cases where the max or min is being taken over a set of values specified by a **condition** schema together with an expression to be evaluated on that set. The **min** and **max** functions are unique in that they provide the only context in which the **bvar** element is optional when using a **condition**; if a **condition** element containing a single variable is given by itself following a **min** or **max** operator, the variable is implicitly assumed to be bound, and the expression to be maximized or minimized is assumed to be the identity.

The **min** and **max** elements may also be applied to a list of values in which case no qualifier schemata are used. For examples of all three usages, see section 4.4.3.5

forall, **exists**

The universal and existential quantifier operators **forall** and **exists** are used conjuction with one or more **bvar** schemata to represent simple logical assertions. There are two ways of using the logical quantifier operators. The first usage is for representing a simple, quantified assertion. For example, the statement "there exists $x < 9$" would be represented as:

```
<apply>
  <exists/>
  <bvar><ci> x </ci></bvar>
  <reln><lt/>
```

```
      <ci> x </ci><cn> 9 </cn>
    </reln>
  </apply>
```

The second usage is for representing implications. Hypotheses are given by a **condition** element following the bound variables. For example the statement "for all $x < 9$, $x < 10$" would be represented as:

```
<apply>
  <forall/>
  <bvar><ci> x </ci></bvar>
  <condition>
    <apply><lt/>
      <ci> x </ci><cn> 9 </cn>
    </apply>
  </condition>
  <reln><lt/>
    <ci> x </ci><cn> 10 </cn>
  </reln>
</apply>
```

Note, in both usages one or more **bvar** qualifier is mandatory.

4.2.4 Relations

binary relation	**neq**
binary logical relation	**implies**
binary set relation	**in, notin, notsubset, notprsubset**
binary series relation	**tendsto**
n-ary relation	**eq, leq, lt, geq, gt**
n-ary set relation	**subset, prsubset**

The MathML content tags include a number of canonically empty elements which denote arithmetic and logical relations. Relations are characterized by the fact that, if an external application were to evaluate them (MathML does not specify how to evaluate expressions), they would typically return a truth value. By contrast, operators generally return a value of the same type as the operands. For example, the result of evaluating $a < b$ is either true or false (by contrast, 1 + 2 is again a number).

Relations are bracketed with their arguments using the **reln** element in much the same way that other functions are bracketed with **apply**. The relation element is the first child element of the **reln**. Thus, the example from the preceding paragraph is properly marked up as:

```
<reln>
  <lt/>
  <ci>a</ci>
  <ci>b</ci>
</reln>
```

It is an error to enclose a relation in an element other than **reln**.

The number of child elements in the **reln** is defined by the element in the first (i.e. relation) position.

Unary relations are followed by exactly one other child element within the **reln**.

Binary relations are followed by exactly two child elements.

N-ary relations are followed by zero or more child elements.

The one exception to these rules is that **declare** elements may be inserted in any position except the first. **declare** elements are not counted when satisfying the child element count for an **reln** containing a unary or binary relation element.

4.2.5 Conditions

condition	**condition**

The **condition** element is used to define the "such that" construct in mathematical expressions. Condition elements are used in a number of contexts in MathML. They are used to construct objects like sets and lists by rule instead of by enumeration. They can be used with the **forall** and **exists** operators to form logical expressions. And finally, they can be used in various ways in conjunction with certain operators. For example, they can be used with and **int** element to specify domains of integration, or to specify argument lists for operators like **min** and **max**.

The **condition** element is almost always used together with one or more **bvar** elements. The only exception is a special usage with the **min** and **max** operators, where the bound variable may be implied. See section 4.4.3.5 for an example of this usage.

The exact interpretation depends on the context, but generally speaking, the **condition** element is used to restrict the permissible values of a bound variable appearing in another expression to those which satisfy the relations contained in the **condition**. Similarly, when the **condition** element contains a **set**, the values of the bound variables are restricted to that set.

A condition element contains a single child which is typically a **reln** element, but may also be an **apply** or a **set** element. The **apply** element is allowed so that several relations can be combined by applying logical operators.

Examples:

The following encodes "there exists x such that $x^5 < 3$".

```
<apply><exists/>
  <bvar><ci> x </ci></bvar>
  <condition>
    <reln><lt/>
      <apply><power/>
        <ci>x</ci>
        <cn>5</cn>
      </apply>
      <cn>3</cn>
    </reln>
  </condition>
```

```
</apply>
```

The next example encodes "for all x,y such that $x^y < 1$ and $y^x < x + y$, $x < Q(y)$".

```
<apply><forall/>
  <bvar><ci>x</ci></bvar>
  <bvar><ci>y</ci></bvar>
  <condition>
    <apply><and/>
      <reln>
        <lt/>
        <apply><power/>
          <ci>x</ci>
          <ci>y</ci>
        </apply>
        <cn>1</cn>
      </reln>
      <reln>
        <lt/>
        <apply><power/>
          <ci>y</ci>
          <ci>x</ci>
        </apply>
        <apply><plus/>
          <ci>y</ci>
          <ci>x</ci>
        </apply>
      </reln>
    </apply>
  </condition>
  <reln><lt/>
     <ci> x </ci>
     <apply>
       <fn><ci> x </ci></fn>
       <ci> y </ci>
     </apply>
  </reln>
</apply>
```

A third example shows the use of quantifiers with **condition**. The following markup encodes "there exists $x < 3$ such that $x^2 = 4$".

```
<apply>
  <exists/>
  <bvar><ci> x </ci></bvar>
  <condition>
    <reln><lt/><ci>x</ci><cn>3</cn></reln>
  </condition>
  <reln>
```

```
      <eq/>
      <apply>
        <power/><ci>x</ci><cn>2</cn>
      </apply>
      <cn>4</cn>
    </reln>
  </apply>
```

4.2.6 Syntax and Semantics

mappings	**semantics, annotation, annotation-xml**

The use of content rather than presentation tagging for mathematics is sometimes referred to as "semantic tagging" [Buswell 1996]. The parse-tree of a fully bracketed MathML content tagged element structure corresponds directly to the expression-tree of the underlying mathematical expression. We therefore regard the content tagging itself as encoding the *syntax* of the mathematical expression. This is, in general, sufficient to obtain some rendering and even some symbolic manipulation (e.g., polynomial factorization).

However, even in such apparently simple expressions as $X + Y$, some additional information may be required for applications such as computer algebra. Are X and Y integers,or functions, etc.? 'Plus' represents addition over which field? This additional information is referred to as *Semantic Mapping*. In MathML, this mapping is provided by the **semantics**, **annotation** and **annotation-xml** elements.

The **semantics** element is the container element for the MathML expression together with its semantic mappings. **semantics** expects a variable number of child elements. The first is the element (which may itself be a complex element structure) for which this additional semantic information is being defined. The second and subsequent children, if any, are instances of the elements **annotation** and/or **annotation-xml**.

The **semantics** tags also accepts a **definitionURL** attribute for use by external processing applications. One use might be a URL for a semantic content dictionary, for example. Since the semantic mapping information might in some cases be provided entirely by the **definitionURL** attribute, the **annotation** or **annotation-xml** elements are optional.

The **annotation** element is a container for arbitrary data. This data may be in the form of text, computer algebra encodings, C programs, or whatever a processing application expects. **annotation** has an attribute **encoding** defining the form in use. Note that the content model of **annotation** is PCDATA, so care must be taken that the particular encoding does not conflict with XML parsing rules.

The **annotation-xml** element is a container for semantic information in well-formed XML. For example, an XML form of the OpenMath semantics could be given. Another possible use here is to embed, for example, the presentation tag form of a construct given in content tag form in the first child element of **semantics** (or vice versa). **annotation-xml** has an attribute **encoding** defining the form in use.

For Example:

```
  <semantics>
    <apply> <divide/>
      <cn>123</cn>
      <cn>456</cn>
    </apply>
```

```
    <annotation encoding="Mathematica">
      N[123/456, 39]
    </annotation>

    <annotation encoding="TeX">
      $0.269736842105263157894736842105263157894\ldots$
    </annotation>

    <annotation encoding="Maple">
      evalf(123/456, 39);
    </annotation>

    <annotation-xml encoding="MathML-Presentation">
      <mrow>
        <mn> 0.269736842105263157894 </mn>
        <mover accent='true'>
          <mn> 736842105263157894 </mn>
          <mo> &obar; </mo>
        </mover>
      </mrow>
    </annotation-xml>

    <annotation-xml encoding="OpenMath">
      <OMA>..</OMA>
    </annotation-xml>
  </semantics>
```

where <OMA>..</OMA> are the elements defining the additional semantic information.

Of course, providing an explicit semantic mapping at all is optional, and in general would only be provided where there is some requirement to process or manipulate the underlying mathematics.

4.2.7 Semantic Mappings

Although semantic mappings can easily be provided by various proprietary, or highly specialized encodings, there are no widely available, non-proprietary standard semantic mapping schemes. In part to address this need, the goal of the OpenMath effort is to provide a platform-independent, vendor-neutral standard for the exchange of mathematical objects between applications. Such mathematical objects include semantic mapping information. The OpenMath group has defined an SGML syntax for the encoding of this information [OpenMath, 1996]. This element set could provide the basis of one **annotation-xml** element set.

An attraction of this mechanism is that the OpenMath syntax is specified in SGML, so that a MathML expression together with its semantic annotations can be validated DTD-based parsers.

4.2.8 MathML element types

MathML functions, operators, and relations can all be thought of as mathematical functions if viewed in a sufficiently abstract way. For example, the standard addition operator can be regarded as a function mapping pairs of real numbers to real numbers. Similarly, a relation can be thought of as a function from

some space of ordered pairs into the set of values {true, false}. To be mathematically meaningful, the domain and range of a function must be precisely specified. In practical terms, this means that functions only make sense when applied to certain kinds of operands. For example, thinking of the standard addition operator, it makes no sense to speak of "adding" a set to a function. Since MathML content markup seeks to encode mathematical expressions in a way that can be unambiguously evaluated, it is no surprise that the types of operands is an issue.

MathML specifies the types of arguments in two ways. The first way is by providing precise instructions for processing applications about the kinds of arguments expected by the MathML content elements denoting functions, operators and relations. These operand types are defined in a dictionary of Default Semantic Bindings for Content Elements given in Appendix F. For example, the MathML Content dictionary specifies that for real scalar arguments the plus operator is the standard commutative addition operator over a field. Elements such as **cn** and **ci** have **type** attributes with default values of "real". Thus some processors will be able to use this information to verify the validity of the indicated operations.

Although MathML specifies the types of arguments for functions, operators and relations, and provides a mechanism for typing arguments, a MathML compliant processor is not required to do any type checking. In other words, a MathML processor will not generate errors if argument types are incorrect. If the processor is a computer algebra system, it may be unable to evaluate an expression, but no MathML error is generated.

4.3 Content Element Attributes

4.3.1 Content Element Attribute Values

Content element attributes are all of the type CDATA, that is, any character string will be accepted as valid. In addition, each attribute has a list of predefined values, which a content processor is expected to recognize and process. The reason that the attribute values are not formally restricted to the list of predefined values is to allow for extension. A processor encountering a value (not in the predefined list) which it does not recognize may validly process it as the default value for that attribute.

4.3.2 Attributes Modifying Content Markup Semantics

Each attribute is followed by the elements to which it can be applied.

4.3.2.1 base

cn

indicates numerical base of the number. Predefined values: any numeric string

Default = **"10"**

4.3.2.2 closure

interval

indicates closure of the interval. Predefined values: **open, closed, open-closed, closed-open**.

Default = **"closed"**

4.3.2.3 definitionURL

fn, declare, semantics,

any operator element

points to an external definition of the semantics of the function or construct being declared. The value is a URL which should point to some kind of definition. This definition overrides the MathML default semantics.

At present, MathML does not specify the format in which external semantic definitions should be given. In particular, *there is no requirement that the target of the URL be loadable and parsable.* An external definition could, for example, define the semantics in human-readable form.

Ideally, in most situations the definition pointed to by the **definitionURL** attribute would be some standard, machine-readable format. However, there are several reasons why MathML does not require such a format.

First, no such format currently exists. There are several projects underway to develop and implement standard semantic encoding formats, most notably the OpenMath effort. But by nature, the development of a comprehensive system of semantic encoding is a very large enterprise, and while much work has been done, much additional work remains. Therefore, even though the **defintionURL** is designed and intended for use with a formal semantic encoding language such as OpenMath, it is premature to require any one particular format.

Another reason for leaving the format of the **definitionURL** attribute unspecified is that there will always be situations where some non-standard format is preferable. This is particularly true in situations where authors are describing new ideas.

It is anticipated that in the near term, there will be a variety of renderer-dependent implementations of the **definitionURL** attribute. For example, a translation tool might simply prompt the user with the specified definition in situations where the proper semantics have been overridden, and in this case, human-readable definitions will be most useful. Other software may utilize OpenMath encodings. Still other software may use proprietary encodings, or look for definitions in any of several formats.

As a consequence, authors need to be aware that there is no guarantee a generic renderer will be able to take advantage of information pointed to by the **definitionURL** attribute. Of course, when widely-accepted standardized semantic encodings are available, the definitions pointed to can be replaced without modifying the original document. However, this is likely to be labor intensive.

There is no default value for the **definitionURL** attribute, i.e. the semantics are defined within the MathML fragment, and/or by the MathML default semantics.

4.3.2.4 encoding

annotation, annotation-xml

indicates the encoding of the annotation. Predefined values **MathML-Presentation, MathML-Content**. Other typical values: **TeX, OpenMath**

Default = "", i.e. unspecified.

4.3.2.5 nargs

declare

indicates number of arguments for function declarations. Pre-defined values: **"nary"**, any numeric string.

Default = **"1"**

4.3.2.6 occurrence

declare

indicates occurrence for operator declarations. Pre-defined values: **prefix, infix, function-model**

Default = **"function-model"**

4.3.2.7 order

list

indicates ordering on the list. Predefined values: **lexicographic, numeric**

Default = **"numeric"**

4.3.2.8 scope

declare

indicates scope of applicability of the declaration. Pre-defined values: **local, global**.

- **local** means the containing MathML element.
- **global** means the containing **math** element.

Default = **"local"**

At present, declarations cannot affect anything outside of the containing **math** element. Ideally, one would like to make document-wide declarations by setting the value of the **scope** attribute to be "global-document". However, the proper mechanism for document-wide declarations very much depends on details of the way in which XML will be embedded in HTML, future XML style sheet mechanisms, and the underlying document object model.

Since these supporting technologies are still in flux at present, the MathML specification does not include "global-document" as a pre-defined value of the **scope** attribute. It is anticipated, however, that this issue will be revisited in future revisions of MathML as supporting technologies stabilize. In the near term, MathML implementors that wish to simulate the effect of a document-wide declaration are encouraged to pre-process documents in order to distribute document-wide declarations to each individual **math** element in the document.

4.3.2.9 type

cn

indicates type of the number. Predefined values: **integer, rational, real, float, complex, complex-polar, complex-cartesian, constant**.

Default = **"real"**

Notes: Each data type implies that the data adheres to certain formating conventions, detailed below. If the data fails to conform to the expected format, an error is generated. Details of the individual

formats are:

real: A real number is presented in decimal notation. Decimal 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 .3, 1, and -31.56. If a different BASE is specified, then the digits are interpreted as being digits computed to that base.

A real number may also be presented in scientific notation. Such numbers have two parts (a mantissa and an exponent) separated by e. The first part is a real number while the second part is an integer exponent indicating a power of the base. For example, 12.3e5 represents 12.3 times 10 ^5.

integer: An integer is represented by an optional sign followed by a string of 1 or more "digits". What a "digit" is depends on the **base** attribute. If **base** is present, it specifies the base for the digit encoding, and it specifies it base ten. Thus **base**='16' specifies a hex encoding. When **base** > 10, letters are added in alphabetical order as digits. The legitimate values for **base** are therefore between 2 and 36.

rational: A rational number is two integers separated by the <**sep**/> element. If **base** is present, it specifies the base used for the digit encoding of both integers.

complex-cartesian: A complex number is of the form two real point numbers separated by <**sep**/>.

complex-polar: A complex number is specified in the form of a magnitude and an angle (in radians). The raw data is in the form of two real numbers separated by <**sep**/>.

constant: The "constant" type is used to denote named constants. For example, an instance of `<cn type="constant">&pi;</cn>` should be interpreted as having the semantics of the mathematical constant Pi. The data for a constant **cn** tag may be one of the following common constants:

Symbol	*Value*
π	The usual π of trigonometry: approximately 3.141592653...
ⅇ (or ⅇ)	The base for natural logarithms: approximately 2.718281828 ...
ⅈ (or ⅈ)	Square root of -1.
γ	Euler's constant: approximately .5772156649...
∞ (or &infty;)	Infinity. Proper interpretation varies with context
&true;	the logical constant 'TRUE'
&false;	the logical constant 'FALSE'
&NotANumber; (or &NaN;)	represents the result of an ill-defined floating point division

ci

indicates type of the identifier. Predefined values: **integer, rational, real, float, complex, complex-polar, complex-cartesian, constant**, any content element name. The meaning of the various attribute values is the same as that listed above for the **cn** element.

Default = `""` , i.e. unspecified.

declare

indicates type of the identifier being declared. Predefined values: any content element name.

Default = **"ci"** , i.e. a generic identifier

set

indicates type of the set. Predefined values: **normal**, **multiset**. "multiset" indicates that repetitions are allowed.

Default = **"normal"**

tendsto

indicates the direction from which the limiting value is approached. Predefined values: **above, below, two-sided**.

Default = **"above"**

4.3.3 Attributes Modifying Content Markup Rendering

4.3.3.1 type

The **type** attribute, in addition to conveying semantic information, can be interpreted to provide rendering information. For example in

```
<ci type="vector">V</ci>
```

a renderer could display a bold **V** for the vector.

4.3.3.2 General Attributes

All content elements support the following general attributes which can be used to modify the rendering of the markup.

- **class**
- **style**
- **id**
- **other**

The **class**, **style** and **id** attributes are intended for compatibility with Cascading Style Sheets, as described in 2.3.4.

Content or semantic tagging goes along with the (frequently implicit) premise that, if you know the semantics, you can always work out a presentation form. When an author's main goal is to mark up re-usable, evaluatable mathematical expressions, the exact rendering of the expression is probably not critical, provided that it is easily understandable. However, when an author's goal is more along the lines of providing enough additional semantic information to make a document more accessible by facilitating better visual rendering, voice rendering, or specialized processing, controlling the exact notation used becomes more of an issue.

MathML elements accept an attribute **other** (see 7.2.3) which can be used to specify things not specifically documented in MathML. On content tags, this attribute can be used by an author to express a *preference* between equivalent forms for a particular content element construct, where the selection of the presentation has nothing to do with the semantics. Examples might be

- inline or displayed equations
- script style fractions
- use of x with a dot for a derivative over dx/dt

Thus, if a particular renderer recognized a display attribute to select between script style and display style fractions, an author might write

```
<apply other='display="scriptstyle"'>
  <divide/>
  <mn> 1 </mn>
  <mi> x </mi>
</apply>
```

to indicate that the rendering 1/*x* is preferred.

The information provided in the **other** attribute is intended for use by specific renderers or processors, and therefore, the permitted values are determined by the renderer being used. It is legal for a renderer to ignore this information. This might be intentional, in the case of a publisher imposing a house style, or simply because the renderer does not understand them, or is unable to carry them out.

Next: Content Markup -- The Content Markup Elements
Up: Table of Contents

5. Mixing Presentation and Content Markup

- 5.1 When to Use Mixed Markup
  - 5.1.1 Why Two Different Kinds of Markup?
  - 5.1.2 Reasons to Mix Markup
- 5.2 How to use Mixed Markup
  - 5.2.1 Presentation Markup Contained in Content Markup
  - 5.2.2 Content Markup Contained in Presentation Markup
- 5.3 Anticipating Macros for Combined Markup

5.1 When to Use Mixed Markup

MathML offers authors elements for both content and presentation markup. Whether to use one or the other, or a mixture of both, depends on what aspects of rendering and interpretation an author wishes to control, and what kinds of re-use he or she wishes to facilitate.

5.1.1 Why Two Different Kinds of Markup?

Chapters 3 and 4 describe two kinds of markup for encoding mathematical material in documents:

Presentation markup captures *notational structure*. It encodes notational structure in a sufficiently abstract way to facilitate rendering to various media. Thus, the same presentation markup can be rendered with relative ease on screen in either wide and narrow windows, in ASCII or graphics, in print, or it can be enunciated in a sensible way when spoken. It does this by providing information such as grouping of expression parts, classification of symbols, etc.

Presentation markup does *not* directly concern itself with the mathematical structure or meaning of an expression. In many situations, notational structure and mathematical structure are closely related, so a sophisticated processing application may be able to heuristically infer mathematical meaning from notational structure. However, in practice, the inference of mathematical meaning from mathematical notation must often be left to the reader.

Employing presentation tags alone may limit the ability to re-use a MathML object in another

context, especially evaluation by external applications.

Content markup captures *mathematical structure*. It encodes mathematical structure in a sufficiently regular way in order to facilitate the assignment of mathematical meaning to an expression by applications. Though the details of mapping from mathematical structure to mathematical meaning are exceedingly 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, independently 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 (such as future versions of Axiom, Maple or Mathematica) with confidence that sensible values will be computed.

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. While a sufficiently sophisticated renderer and style-sheet mechanism could in principle allow a user to read mathematical documents using personalized notational preferences, in practice, rendering content expressions with notational nuances still requires human intervention of some sort.

Employing content tags alone may limit the ability of the author to precisely control how an expression is rendered.

It is important to emphasize that both content and presentation tags are necessary in order to provide the full expressive capability one would like in a mathematical markup language. Often the same mathematical notation is used to represent several completely different concepts. For example, the notation x^i may be intended (in polynomial algebra) as the *i-th* power of the variable *x*, or (in vector analysis) as the *i-th* component of the 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. In order to provide authors with the ability to precisely control notational nuances while at the same time encoding meanings in a machine readable way, both content and presentation markup are needed.

In general, if it is important to control exactly how an expression is rendered, presentation markup will generally be more satisfactory. If it is important that the meaning of an expression can be dependably and automatically interpreted, then content markup will generally be more satisfactory.

5.1.2 Reasons to Mix Markup

In many common situations, an author or authoring tool may choose to generate either presentation or content markup exclusively. For example, a programs for translating legacy documents would most likely generate pure presentation markup. Similarly, an educational software package might very well 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 presentation-oriented, interspersing some content markup will often produce more accessible, more re-usable results. 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 and the superscript denotes a vector component, and **x** is a real variable. On account of the linear algebra context, a visually impaired reader may have directed his or her voice synthesis software to render superscripts as vector components. By explicitly encoding the power, the content markup yields a much better voice rendering than would likely happen by default.

If an author is primarily content-oriented, there are two reasons to intersperse presentation markup. First, using presentation markup provides a way of modifying or refining how a content expression is rendered. For example, one might write:

```
<reln>
  <in/>
  <ci><mi fontweight="bold">v</mi></ci>
  <ci>S</ci>
</reln>
```

In this case, the use of embedded presentation markup allows the author to specify that **v** should be rendered in boldface.

A second reason to intersperse presentation in content markup is that there is a continually growing list of areas of discourse which do not have pre-defined content elements for encoding their objects and operators. As a consequence, any system of content markup inevitably requires an extension mechanism which combines notation with semantics in some way. MathML content markup specifies several ways of attaching an external semantic definitions to content objects. However, it is necessary 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 use the **mo** presentation element inside a **ci** element to achieve the proper presentation, along with a **semantics** element which binds a semantic definition to the symbol. The **mo** element indicates to a renderer that it should use standard operator spacing around the content identifier "rank", just as it would for "sin" or "log":

```
<reln>
  <eq/>
  <apply>
    <fn>
      <semantics>
        <ci><mo>rank</mo></ci>
        <annotation-xml encoding="OpenMath">
          <OMS CD="BasicLinAlg">matrix-rank</OMS>
        </annotation-xml>
      </semantics>
    </fn>
    <apply>
      <times/>
      <apply>
        <transpose/>
        <ci>u</ci>
      </apply>
      <ci>v</ci>
    </apply>
  </apply>
  <cn>1</cn>
</reln>
```

Here, the semantics of the presentation subexpressions have been given using symbols from OpenMath content dictionaries (CD).

5.2 How to Mix Markup

The main consideration when presentation markup and content markup are mixed together in a single expression is that the result should still make sense. When both kinds of markup are contained in a presentation expression, this means it should be possible to render the resulting mixed expressions simply and sensibly. Conversely, when mixed markup appears in a content expression, it should be possible to simply and sensibly assign a semantic interpretation to the expression as whole. These requirements place a few natural constraints on how presentation and content markup can be mixed in a single expression, in order to avoid ambiguous or otherwise problematic expressions.

Two motivating examples illustrate the kinds of problems that must be avoided in mixed markup. Consider:

```
<mrow> <plus/> <mi> x </mi> <mi> y </mi> </mrow>
```

In this example, the content element **plus** has been indiscriminately embedded in a presentation expression. Should the plus sign appear in its usual infix position, as it would in content markup, or should it render as the first thing in the row? Neither choice is very satisfactory, and consequently, this kind of mixing is not allowed. Similarly, consider:

```
<apply> <ci> x </ci> <mo> + </mo> <ci> y </ci> </apply>
```

As before, the **mo** element is problematic. Should a renderer infer that the usual arithmetic operator is intended, and act as if the prefix content element **plus** had been used? Again, there is no compelling answer, and thus this kind of mixing of content and presentation markup is also prohibited.

5.2.1 Presentation Markup Contained in Content Markup

The use of presentation markup within content markup is limited to situations which do not effect the ability of content markup to unambiguously encode mathematical meaning. More specifically, presentation markup may only appear in content markup in three ways:

1. within **ci** and **cn** token elements
2. within the **fn** 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 #PCDATA, presentation elements, and **sep** empty elements. Contiguous blocks of #PCDATA in **ci** and **fn** elements are rendered as if they were wrapped in mi elements. A contiguous blocks of #PCDATA within **cn** should be rendered as if wrapped in **mn**. If a token element contains both #PCDATA and presentation elements, contiguous blocks of #PCDATA (if any) are treated as if wrapped in **mi** or **mn** elements as appropriate, and the resulting collection of presentation elements are rendered as if wrapped in an **mrow** element.

The **sep** element is only meaningful in identifiers and numbers defined to be of complex type, where it separates #PCDATA into real and imaginary parts. When a token elements contains both **sep** elements and presentation elements, the **sep** elements are ignored.

Presentation markup within the fn element.

The **fn** element may contain either #PCDATA interspersed with presentation markup, or content container elements. It is a MathML error for an **fn** element to contain both presentation and content elements. When the **fn** element contains both raw data and

presentation markup, the same rendering rules that apply to content token elements 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 wrapping it in a **semantics** element. Of course, the intention is that the meaning of the wrapped expression should be indicated by one or more annotation elements also contained in the **semantics** element. Suggested rendering for a **semantics** element is discussed in 4.2.6.

5.2.2 Content Markup Contained 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. In practice, this basically translates into the condition that content container elements are permitted, while other content elements such as operators, relations, and qualifier elements are not.

As a rule, content elements other than containers derive part of their semantic meaning from the surrounding context, such as whether an operator is being applied to arguments or not, or whether a **bvar** element is qualifying an integral, and so on. Thus, in a presentation context, these elements do not have a clearly defined meaning, and hence there is no obvious choice for a rendering. Consequently, they are not allowed.

The complete list of content elements which may appear as a child in a presentation element is:

<ci>	**<cn>**	**<apply>**	**<fn>**	**<lambda>**
<reln>	**<interval>**	**<list>**	**<matrix>**	**<matrixrow>**
<set>	**<vector>**	**<declare>**		

Note that within presentation markup, content expressions may only appear in locations where it is valid for any MathML expression to appear. In particular, content expressions may not appear within presentation token elements. In this regard mixing presentation and content are asymmetrical.

For rendering purposes, when a permitted content element appears within a presentation context, a processing application should treat it as if it were replaced with an **mrow** containing a presentation encoding of the rendering the application would ordinarily generate for that content markup. For example, consider:

```
<mfrac>
  <mi>x</mi>
  <interval closure="open-closed">
```

```
      <cn>1</cn>
      <cn>3</cn>
    </interval>
  </mfrac>
```

In this case, a visual renderer would typically render the **interval** construct as (1,3]. Using presentation markup, this might be encoded as:

```
<mfenced close="]">
  <mn>1</mn>
  <mn>3</mn>
</mfenced>
```

Consequently, the original mixed markup should be visually rendered as

```
<mfrac>
  <mi>x</mi>
  <mfenced close="]">
    <mn>1</mn>
    <mn>3</mn>
  </mfenced>
</mfrac>
```

5.3 Anticipating Macros for Combined Markup

The examples above show that introducing new mathematical content as *combined* presentation-content pairs is verbose.

Certainly one can imagine generating this kind of markup with software tools, but it is at the borderline of what might be deemed tolerable to do by hand.

This is one of the areas of the MathML specification which anticipates most strongly the use of macros. With some kind of HTML/XML macro mechanism, it would be possible, for example, to define macros

```
<rank/>
```

and

```
<tr>X</tr>
```

which respectively expand to

```
<fn>
  <semantics>
    <ci><mo>rank</mo></ci>
    <annotation-xml encoding="OpenMath">
      <OMS CD="BasicLinAlg">matrix-rank</OMS>
    </annotation-xml>
  </semantics>
```

```
</fn>
```

and

```
<apply>
  <transpose/>
  <ci>X</ci>
</apply>.
```

The sample encoding of

rank($u^T v$) = 1,

from section 5.1.3 could then be condensed to

```
<reln>
  <eq>
  </apply>
    <rank/>
    <apply>
      <times/>
      <tr>u</tr>
      <ci>v</ci>
    </apply>
  </apply>
  <cn>1</cn>
</reln>
```

From this example we see how the combination of presentation and content markup could become much simpler and effective to generate as standard macro libraries become available.

Next: Entities, Characters and Fonts
Up: Table of Contents

6. Entities, Characters and Fonts

- 6.1 Introduction
  - 6.1.1 The Intent of Entity Names
  - 6.1.2 The STIX Project
- 6.2 Entity Listings
  - 6.2.1 Non-Marking Entities
  - 6.2.2 Printing Entity List
  - 6.2.3 Special Constants
  - 6.2.4 Alphabetical Lists
  - 6.2.5 ISO Entity Set Groupings
    - 6.2.5.1 ISO Symbol Entity Sets
    - 6.2.5.2 ISO Math Font Entity Sets
    - 6.2.5.3 Other ISO Font Entity Sets
  - 6.2.6 Additional Entity Set Grouping

6.1 Introduction

6.1.1 The Intent of Entity Names

Notation has proved very important for mathematics. Mathematics has grown in part because of the succinctness and suggestiveness of its evolving notation. There have been many new signs evolved for use in mathematical notation, and mathematicians have not held back from making use of many symbols originally developed 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 in coding. It is difficult to read mathematics if glyphs are not available for presentation on specific display devices.

This situation poses a problem for the W3C Math Working Group. It does not fall naturally within the purview of a math for HTML specification and DTD production to worry about more than the entities allowed in the DTD. Moreover, as experience has shown, a long list of entities with no means to display them is of little use, and a cause of frequent frustrations in trying use a standard. On the other hand, a large collection of glyphs or characters without a standard way to

refer to them is not of much use either.

The W3C Math Working Group has therefore taken on directly specification of part of the full mechanism of proceeding from notation to final presentation, and is collaborating with organizations undertaking specification of the rest.

For instance, we try to use entity names that are contained in ISO TR 9573, which supersedes the ISO TR 8879 annex as far as math is concerned. There are considerations of mathematical usage that do on occasion militate against this, and the TR 9573 lists need supplementing. We hope to be able to agree with the TR 9573 WG on suitable extensions, in the course of the revision of their document that they are presently undertaking.

The STIX project of the STIPUB group of scientific and technical publishers has also been working toward a common collection of mathematical symbols and names. The W3C Math Working Group expects to issue further updates on the matter of character entities as a consequence of this project's useful work. For the latest character tables and fonts information, see the W3C Math Working Group home page.

6.1.2 The STIX Project

The STIX project team leader, Nico Poppelier, is a member of the W3C Math Working Group. The STIX project, set up by the STIPUB group of publishers, aims to formulate a collection of characters needed in the course of scientific and technical publishing. A database of characters in common use is being produced by collaborating publishing organizations. The team will propose to the Unicode consortium the additions to the next revision of the Unicode character set that this process shows are needed, together with the appropriate character codes. Finally the STIX project will commission the production of a complete set of fonts covering those Unicode characters for science and technology, to be made available to the public under license, but free of charge. The STIPUB group recognizes that easy availability of the characters and fonts greatly facilitates communication and publication.

6.2 Entity Listings

This chapter of the MathML proposal contains a listing of entities for use in MathML.

To provide more background on the characters used by mathematics we have used a larger comparative database showing codes and meanings in other common math environments. The W3C Math Working Group is very grateful to Elsevier Science and to Wolfram Research (makers of Mathematica ®) for making available to us so much useful data.

6.2.1 Non-Marking Entities

Some character entities, although important for the quality of print rendering do not directly have glyph marks that correspond. They are called here non-marking entities. Below we have a table of those adopted for the purposes of MathML. Their roles are discussed in Chapters 3 and

4, respectively on Presentation and Content Markup. The values of the spaces given are recommendations. Some of these characters do not already have Unicode values. Arbitrary values up in the Private Zone E8 range have been assigned. The correspondence between the spacing values mentioned below and those in the Unicode descriptions are not exact, but are good matches.

Entity name	Unicode	Description
	0009	tabulator stop; horizontal tabulation
000A	force a line break; line feed
&IndentingNewLine;	E891	force a line break and indent appropriately on next line
	E892	never break line here
&GoodBreak;	E893	if a linebreak is needed, here is a good spot
&BadBreak;	E894	if a linebreak is needed, try to avoid breaking here
&Space;	0020	one em of space in the current font
	00A0	space that is not a legal breakpoint
	200B	space of no width at all
	200A	space of width 1/18 em
	2009	space of width 3/18 em
	2005	space of width 4/18 em
	E897	space of width 5/18 em
	E898	space of width -1/18 em
	E899	space of width -3/18 em
	E89A	space of width -4/18 em
	E89B	space of width -5/18 em
⁣	E89C	used as a separator, e.g., in indices (Section 3.2.4)
⁣	E89C	short form of ⁣
⁢	E89E	marks multiplication when it is understood without a mark (Section 3.2.4)
⁢	E89E	short form of ⁢
⁡	E8A0	character showing function application in presentation tagging (Section 3.2.4)
⁡	E8A0	short form of ⁡

6.2.2 Printing Entity Listings

Since the situation concerning availability of character codes from Unicode and under ISO 9573-13 is not yet fully clear at the time of writing, we have decided to proceed conservatively.

We have taken the ISO 9573-13 proposal, as conveyed to us from Anders Berglund, and have added a number of additional aliases based in the practice of the mathematical typesetting community. Thus the main influence outside ISO has been the names to be found in the TeX community.

To facilitate comprehension of a fairly large list of names, which totals over 2000 in this case, we offer the same information in more than one form.

We have entities listed by name and sample glyphs for all of them. Each entity name is accompanied by a code for a character grouping chosen from a list given below, a short verbal description, and a Unicode hex code if there is a corresponding sample glyph to be found in ISO 10646. Those codes beginning with the hex digit E, e.g., E321, indicate assignments to the private zone of Unicode. This indicates that the character in question is not at present an official Unicode character. It is highly recommended that authors use entity names instead of Unicode values, especially for those characters in the Unicode private zone, as those values may change. It is hoped that most of these characters will become officially endorsed by Unicode and ISO under its 10646 standard in due course. In any case we expect fonts for these characters to become publicly available as the use of MathML develops. If the entity name is an alias then a reference back to the ISO form is given if there is one, and to a preferred form if not. The ISO or preferred forms have references to their alternates where they exist.

Newly Revised. The entity listings by alphabetical and Unicode order in section 6.2.4 have now been brought more into line with the corresponding ISO character sets, in that if some part of a set is included then the entire set is included. Also, ISOCHEM has been dropped. These changes have also been reflected in the entity declarations in the DTD in Appendix A.

The tables of character sets with glyphs given in section 6.2.5 have not been revised from the original tables. In cases where information section 6.2.4 and 6.2.5 conflict, the tables in 6.2.3 and the DTD should be considered normative.

6.2.3 Special Constants

To commence we list separately a few of the special characters which MathML has seen fit to be a little radical in introducing. There are two for special constants and one for calculus. They too must have private Unicode values.

Entity name	Unicode	Description
ⅅ	F74B	D for use in differentials, e.g., within integrals
ⅅ	F74B	short form of ⅅ

ⅆ	F74C	d for use in differentials, e.g., within integrals
ⅆ	F74C	short form of ⅆ
ⅇ	F74D	e for use for the exponential base of the natural logarithms
ⅇ	F74D	short form of ⅇ
&false;	E8A7	logical constant false
ⅈ	F74E	i for use as a square root of -1
ⅈ	F74E	short form of ⅈ
&NotANumber;	E8AA	used in 4.3.2.9
&true;	E8AB	logical constant true

6.2.4 Alphabetical Lists

The first table offered is a very large ASCII listing of printing entity names, ordered alphabetically, with upper-case preceding lower-case as in ASCII order. The Unicode numbers beginning with E are arbitrary assignments in the Private Area where there is presently no Unicode character available. When there is no Unicode offered at all it is because the characters listed can be thought of as font variations of common Roman alphabetic characters.

There is also an ASCII listing of printing entities ordered by Unicode number. Next we have collections of the entities in entity sets which are similar to the groupings in the corresponding ISO documents.

6.2.5 ISO Entity Set Groupings

In addition, we list the above material in the groupings used by ISO 9573-13 with an additional grouping of aliases introduced. This table makes explicit the entity groupings and provides links to ASCII listings of the groups and HTML tabular listings which display the glyphs, insofar as they are to be had, as well.

6.2.5.1 ISO Symbol Entity Sets

The symbols for mathematics that ISO have considered are organized, for both historical and mnemonic reasons into groupings with somewhat descriptive names. In the tables below we reproduce the newly proposed versions of these groups and give the corresponding Unicode sample glyphs. For each ISO 9573-13 group we give first an Extended version in ASCII listing which includes aliases, then a similar listing with sample glyphs, then the Basic ISO 9573-13 entity set and its version with included glyphs. The entries are organized alphabetically by entity name.

It should be noted that the sample glyphs given here are in GIF files intended for viewing on a monitor's screen at 72dpi. They are not suitable for printing, and in particular do not constitute a

set of fonts covering the symbols of mathematics. In addition, it is important to note that the Unicode numbers assigned in the private zone, beginning with hex digits E2 and above, are arbitrary and only used here to ensure that sample glyphs are available for display. They do not constitute suggested assignments of codes. Such a set of fonts is under development in more than one context. The MathML Working Group is engaged in ensuring that fonts will be readily publicly available.

This first block of entity sets includes mostly non-letter symbols, along with a few letters loaded with mathematical semantics. At the end of the block we have included the table MMALIAS of the aliases introduced by MathML, which mostly come from the TeX community, and MMEXTRA with the additional character entities added by MathML. Note that some of the blocks are place-holders for a possible future expansion of the tables.

Group	Descriptive Name	
ISOAMSA	Added Math Symbols: Arrows	Extended Glyphs | Basic Glyphs
ISOAMSB	Added Math Symbols: Binary Operators	Extended Glyphs | Basic Glyphs
ISOAMSC	Added Math Symbols: Delimiters	Extended Glyphs | Basic Glyphs
ISOAMSN	Added Math Symbols: Negated Relations	Extended Glyphs | Basic Glyphs
ISOAMSO	Added Math Symbols: Ordinary	Extended Glyphs | Basic Glyphs
ISOAMSR	Added Math Symbols: Relations	Extended Glyphs | Basic Glyphs
ISOTECH	General Technical	Extended Glyphs | Basic Glyphs
ISOPUB	Publishing	Extended Glyphs | Basic Glyphs
ISODIA	Diacritical Marks	Extended Glyphs | Basic Glyphs
ISONUM	Numeric and Special Graphic	Extended Glyphs | Basic Glyphs
ISOBOX	Box and Line Drawing	Basic Glyphs
MMALIAS	MathML Aliases	Basic Glyphs
MMEXTRA	MathML Additions	Basic Glyphs

6.2.5.2 ISO Math Font Entity Sets

Mathematical literature displays the common use of particular font styles. Characters representing given letters which differ only in the glyph presentation are in principle not different for the purposes of a character registry such as Unicode, which is not supposed to take into account mere font differences. However usage has meant that both ISO and Unicode, like mathematics, recognize them as different entities. Therefore we include lists for Greek, script, open face (also known as double struck or blackboard bold), and fraktur (also known as gothic or German) fonts.

Group	Descriptive Name	
ISOGRK3	Greek Symbols	ASCII Glyphs
ISOMSCR	Math Script Font	ASCII Glyphs

ISOMOPF	Math Open Face Font	ASCII Glyphs
ISOMFRK	Math Fraktur Font	ASCII Glyphs

6.2.5.3 Other ISO Font Entity Sets

For reference we provide a list of the names of several other ISO font entity sets which are really normally used for text. ISOGRK4 is actually a collection of emboldened forms of the Greek letters.

Group	Descriptive Name
ISOGRK1	Greek Letters
ISOGRK2	Monotoniko Greek
ISOGRK4	Alternative Greek Symbols
ISOCYR1	Russian Cyrillic
ISOCYR2	Non-Russian Cyrillic

6.2.6 Additional Entity Set Grouping

In addition to the above listed, for the sake of completeness, we provide a table of other entities not within the ISO lists which are referred to somewhere in this specification. It is not certain that all these characters, though of mathematical significance, will reach incorporation within Unicode. The W3C Math WG continues to wrestle with the problems of the characters of mathematics.

&LeftSkeleton;	E850	start of missing information
&RightSkeleton;	E851	end of missing information
&LeftBracketingBar;	F603	left vertical delimiter
&RightBracketingBar;	E604	right vertical delimiter
&LeftDoubleBracketingBar;	F605	left double vertical delimiter
&RightDoubleBracketingBar;	F606	right double vertical delimiter
─	E859	short horizontal line
		E85A	short vertical line
≔	E85B	assignment operator
❘	E85C	vertical separating operator
⫤	E30F	alias for ⫤
⥰	F524	right double arrow with rounded head (looks like thin superset)
⊏̸	E604	negated set-like partial order operator
⊐̸	E615	negated set-like partial order operator

⊈	2288	alias of ⊈
⊉	2289	alias of ⊉
⥐	F50B	left-down-right-down harpoon
⥞	F50E	left-down harpoon from bar
⥖	F50C	left-down harpoon to bar
⥟	F50F	right-down harpoon from bar
⥗	F50D	right-down harpoon to bar
⇤	21E4	alias for ⇤
⥎	F505	left-up-right-up harpoon
↤	21A4	alias for ↤
⥚	F509	left-up harpoon from bar
⥒	F507	left-up harpoon to bar
⇥	21E5	alias for ⇥
⥛	F50A	right-up harpoon from bar
⥓	F508	up-right harpoon to bar
⩵	F431	two consecutive equal signs
⪢	E2F7	alias for ≫
⧏	F410	not left triangle, vertical bar
⪡	E2FB	alias for ≪
≭	226D	alias for &nasymp;
≂̸	E84E	alias for ≂̸
≎̸	E616	alias for ≎̸
≏̸	E84D	alias for ≏̸
⧏̸	F412	not left triangle, vertical bar
⪢̸	F428	not double greater-than sign
⪡̸	F423	not double less-than sign
&NotPrecedesTilde;	E5DC	alias for ⪯̸
⧐̸	E870	not vertical bar, right triangle
≿̸	E837	not succeeds or similar
⧐	F411	vertical bar, right triangle
∏	220F	alias for ∏
⋄	22C4	alias for ⋄
⨯	E619	cross or vector product

□	25A1	alias for □
⤓	F504	down arrow to bar
↧	21A7	alias for ↧
⥡	F519	down-left harpoon from bar
⥙	F517	down-left harpoon to bar
⥑	F515	up-left-down-left harpoon
⥠	F518	up-left harpoon from bar
⥘	F516	up-left harpoon to bar
⥝	F514	down-right harpoon from bar
⥕	F512	down-right harpoon to bar
⥏	F510	up-right-down-right harpoon
⥜	F513	up-right harpoon from bar
⥔	F511	up-right harpoon to bar
↓	E87F	short down arrow
↑	E880	sort up arrow
⤒	F503	up arrow to bar
↥	21A5	↥
̑	0311	breve, inverted (non-spacing)
‾	00AF	over bar
⏞	F612	over brace
⎴	F614	over bracket
⏜	F610	over parenthesis
_	0332	combining low line
⏟	F613	under brace
⎵	F615	under bracket
⏝	F611	under parenthesis
▫	F530	empty very small square
▪	F529	filled very small square
◻	F527	empty small square
◼	F528	filled small square
⧴	F51F	rule-delayed (colon right arrow)

Next: The MathML Interface

Up: Table of Contents

7. The MathML Interface

- 7.1 Embedding MathML in HTML
  - 7.1.1 The Top-Level **math** Element
  - 7.1.2 Requirements for a MathML Browser Interface
  - 7.1.3 Invoking Embedded Objects as Renderers
  - 7.1.4 Invoking Other Applications
  - 7.1.5 Mixing and Linking MathML and HTML
- 7.2 Generating, Processing and Rendering MathML
  - 7.2.1 MathML Compliance
  - 7.2.2 Handling Of Errors
  - 7.2.3 An Attribute for Unspecified Data
- 7.3 Future Extensions
  - 7.3.1 Macros and Style Sheets
  - 7.3.2 XML Extensions to MathML

To be effective, MathML must work well with a wide variety of renderers, processors, translators and editors. This chapter addresses 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 HTML.

There are three kinds of interface issues that arise in embedding MathML in HTML. First, MathML must be semantically integrated into HTML. For example, there must be a mechanism for browsers to recognize MathML markup as embedded content, and not as an HTML syntax error. More generally, the semantic embedding of MathML in HTML is a special case of embedding XML in HTML, which involves issues such as name space management, and document validation.

Second, MathML rendering must be integrated into browser software. Until MathML is rendered natively by browsers, rendering will typically be done by embedded elements. However, to properly render mathematical notation in context in a Web document, improved coordination between browsers and embedded elements will be necessary. For example, embedded elements will need to be able to detect the ambient rendering environment, such as

baseline, font family and color scheme, and respond appropriately to reader input such as font size changes. Support for printing is also essential.

Third, tools for generating MathML must be developed, including editors, translators, and export capabilities in computer algebra systems, and other scientific software.. Since MathML is designed to be powerful and flexible to accommodate a wide range of applications, while at the same time remaining structured and explicit for easy processing, MathML expressions tend to be lengthy, and prone to error when entered by hand. Therefore, special emphasis must be given to insuring that MathML can be easily generated by user-friendly conversion and authoring tools.

The W3C Math working group is committed to working with software vendors to develop a wide range of equation editors and translation tools, and plans to continue to do so in the future. In particular, the working group monitors the public www-math@w3.org mailing list, and will attempt to provide support to software developers with questions about the MathML specification. The working group also intends to try to stimulate the formation of MathML developer and user groups. For current information about MathML tools, applications and user support activities, consult the W3C Math home page.

7.1 Embedding MathML in HTML

MathML specifies a single top-level **math** element, which encapsulates each instance of MathML markup within an HTML page. As such, the **math** element provides an attachment point for information which affects a MathML expression as a whole. For example, the **math** element is the logical place to attach style sheet or macro information in the future, when these facilities become available for MathML.

Ideally, the **math** element should also serve as the interface for embedding MathML in HTML. To function in this capacity, the **math** element would have to simultaneously signal the semantic inclusion of MathML (XML) content in HTML, and provide the necessary machinery for rendering its content in a browser either by invoking an embedded element, or by specifying parameters for a native renderer in the browser. Both semantic inclusion and rendering present a number of issues that extend beyond the boundaries of W3C Math. To a large extent, the issues which arise for embedding MathML in HTML are the same as those for the more general problem of embedding XML in HTML. Resolving these issues will require the efforts of a number of World Wide Web Consortium Activities, including the HTML, XML, CSS and DOM activities.

In order to produce a complete and self-contained description of MathML, this document only specifies the attributes and usage of the **math** element as a top-level element for MathML, and not as an interface element. The W3C Math working group intends to continue working closely with other World Wide Web Consortium activities to insure that emerging standards for embedding XML in HTML accommodate seamless integration of MathML in HTML. Section 7.1.2 lists requirements which an interface element for MathML would have to meet in order to

fully integrate MathML into HTML. However, it is important to note that the MathML specification is independent of the ultimate embedding mechanism.

7.1.1 The Top-Level math Element

As stated above, MathML specifies a single top-level **math** element. 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 which return subexpressions of other MathML expressions, for example as the result of a cut-and-paste operation, should always wrap them in **<math>** tags. The presence of enclosing **<math>** tags should be a reasonable heuristic test for MathML content. 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 a **mrow** element.

The attributes of the **math** element are:

class="*value*"
style="*value*"

Provided for future Cascading Style Sheet compatibility.

macros="*URL URL ...*"

This attribute provides a way of pointing to external macro definition files. Macros are not part of the MathML specification, but a macro mechanism is anticipated as a future extension to MathML.

mode="*display|inline*"

The **mode** attribute specifies whether the enclosed MathML expression should be rendered in a display style or an in-line style. The default is **mode**="inline".

7.1.2 Requirements for a MathML Browser Interface

The top-level **math** element described in the preceding section is concerned with encapsulating MathML content and defining attributes which affect the entire enclosed expression. It is, in a sense, "inward looking." However, to render MathML properly in a browser, and to integrate it properly into an HTML document, an "outward looking" interface element is also required. This interface element must be aware of its surrounding environment, and provide a mechanism for passing information between the browser, and the MathML renderer.

As noted above, the MathML interface element and the MathML top-level element should ideally be one and the same. The **math** element should not only serve to encapsulate MathML content, it should signal the semantic embedding of MathML content to an HTML processor, and admit additional attributes for controlling how the MathML renderer should interact with

the browser.

Since a general mechanism for embedding XML in HTML is anticipated in the near future which may not be compatible with using the top-level **math** element for the interface element as well, the remainder of this section describes attributes and functionality that a MathML interface element should ultimately provide. In the near term, implementors attempting to provide interim solutions for rendering MathML in browsers should try to give authors some way of passing the following interface attributes to the renderer:

type="*mime type*"

The type attribute assigns a MIME type to the tag content. This attribute should ideally be used to invoke an embedded element, such as a Java applet, plug-in or ActiveX control, to render the tag content as described in the next section.

name="*value*"

Provided for scripting.

height=*nn*
width=*nn*
baseline=*nn*

Ideally, embedded elements will soon be able to dynamically negotiate height, width and baseline alignment with browsers. However, these optional attributes are suggested as an interim solution for software vendors that want to support MathML, but are unable to provide dynamic resizing and alignment.

overflow="*scroll*|*elide*|*truncate*|*scale*"

In cases where size negotiation is not possible or fails (for example in the case of an extremely long equation), this attribute is provided to suggest an alternative processing method to the renderer.

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=*URL*
alttext="*value*"

These attributes provide graceful fall-backs for browsers that do not support embedded elements, or images respectively.

Attributes which apply to the MathML interface element necessarily take effect when the document is first loaded, and therefore suffer the limitation that they cannot change in response to reader interaction. The **height** and **width** attributes are good examples; if the reader changes the current font size, the height and width of the embedded math fragments also need to change. Therefore, in order to properly render MathML, an embedded element must be able to communicate with the browser, and react to reader input.

At present, browser support for embedded elements is too limited to provide acceptable rendering for MathML. The W3C Math working group is working closely with the Document Object Model working group in an effort to provide better communication between embedded MathML renderers and browsers. Some of the most needed improvements are:

- Embedded elements must be able to determine the ambient style parameters, including font characteristics, foreground and background colors, and link color schemes. Embedded elements must also be able to align themselves to an arbitrary baseline, rather than the existing top, middle, bottom alignment options.
- Embedded elements must be able to detect and react to reader input. In particular, embedded elements must be able to dynamically resize themselves when the ambient font size changes.
- Embedded elements must be able to print in context, and at high resolution.

7.1.3 Invoking Embedded Objects as Renderers

Until MathML is natively supported by browsers, we anticipate that MathML rendering will be carried out via embedded objects such as plug-ins, applets, or helper applications. In the near term, the W3C Math working group advocates the use of MIME types to bind embedded MathML to renderers. Mechanisms for assigning MIME types already exist in HTML, and mechanisms for registering and automatically invoking embedded elements such as plug-ins based on MIME type already exist in Web browsers.

The **type** attribute, described in the previous section as a requirement for the MathML interface element, is intended to associate a MIME type with its content. The HTML element META is proposed as a means of specifying document-wide default MIME types for an element.

We propose a simple MIME type naming convention which is flexible enough to accommodate several common situations:

- An author wishing to reach an as wide an audience as possible might like MathML to be rendered by any available renderer.

- An author targeting a specific audience might like indicate that a particular MathML be used.
- A reader might wish to specify which of several available renderers should be used.

We propose that generic MathML be assigned the MIME type `text/mathml`, and for browser registry, we suggest the standard file extension `.mml` be used. To invoke specific renderers, we suggest assigning a MIME type of the following format:

```
text/mathml-renderer
```

Example:

A user downloads and installs renderer A, and registers it with the browser for the `text/mathml` MIME type to process generic MathML. However renderer A also accepts TeX as an input syntax, and therefore during the installation process, it requests to be registered for `application/x-tex` as well. Later, the user discovers renderer B provides additional features, such as cut and paste capability. Therefore, the user downloads, installs and registers renderer B for the `text/mathml-rendererB` MIME type.

An author then creates a document that contains the the following line in the document header:

```
<META Content-math-Type="text/mathml">
```

Later, the document contains the following expressions:

```
<math>
  <msup><mi>x</mi><mn>2</mn></msup>
</math>

<math type="text/mathml-rendererB">
  <mi>&alpha;</mi><mo>=</mo><mn>0.4</mn>
</math>
```

When our hypothetical reader views this document, renderer A is invoked to process the first expression, while renderer B is invoked for the second. Later, when our hypothetical reader later views a document with MIME type `application/x-tex`, renderer A is again invoke, this time in TeX processing mode.

7.1.4 Invoking Other Applications

Although rendering MathML expressions typically 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.

At present, there is no standard way of specifying that embedded content should be rendered with one application, edited in another, and evaluated by a third. As work progresses on coordination between browsers and embedded elements and the Document Object Model (DOM), providing this kind of functionality should be a priority. Both authors and readers

should be able to indicate a preference about what MathML application to use in a given context. For example, one might imagine that some mouse gesture over a MathML expression would cause a browser to present the reader with a pop-up menu, showing the various kinds of MathML processing available on the system, and the MathML processors recommended by the author.

Since MathML will probably be widely 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 MathML specification does not explicitly contain provisions for recording authoring tool information. In some circumstances, it may be possible to include authoring tool information which applies to an entire document as meta data; interested readers are encouraged to consult the W3C Metadata Activity for current information about metadata and resource definition. For encoding authoring tool state information that applies to a particular MathML instance, readers are referred to the possible use of the **semantics** element for this purpose.

7.1.5 Mixing and Linking MathML and HTML

In order to be fully integrated into HTML, it should be possible not only to embed MathML in HTML, but also to embed HTML in MathML. However, the problem of supporting HTML in MathML presents many difficulties. Moreover, the problems are not specific to MathML; they are problems for XML applications in HTML generally. Therefore, at present, the MathML specification does not permit any HTML elements within a MathML expression, although this may be subject to change in a future revision of MathML, when.mechanisms for embedding XML in HTML have been further developed.

In most cases, HTML elements either do not apply in mathematical contexts (headings, paragraphs, lists, etc), or MathML already provides equivalent or better functionality specifically tailored to mathematical content (tables, style changes, etc). However, there are two notable exceptions.

Linking

MathML has no element which corresponds to the HTML anchor element **a**. In HTML, anchors are used both to make links, and to provide locations to link to. MathML, as an XML application, defines links by the use of the **XML-LINK** attribute. However, MathML at present does not provide a way for other documents to make links into a MathML expression. One reason for this omission is that linking into embedded XML content is better addressed as part of a general mechanism for embedding XML in HTML. Moreover, until browsers either natively implement MathML rendering, or substantially better coordination between embedded elements and browsers becomes possible, there is no reasonable way of implementing links into MathML expressions.

MathML linking elements are generic XML linking elements as described in the Extensible

Markup Language (XML): Part 2. Linking working draft. The reader is cautioned, however, that this working draft is less mature than the XML syntax working draft, and is therefore more subject to future revision. Since the MathML linking mechanism is defined in terms of the XML linking specification, the same proviso holds for it as well.

A MathML element is designated as a link by the presence of the **XML-LINK** attribute. The possible values for the this attribute are "simple", "extended", "locator", "group" and "document". Although all of these values are valid, MathML renderers need only implement "simple" XML links to be MathML compliant. How links are indicated to the reader is left to the individual MathML processing application.

Elements which specify the value of the **XML-LINK** attribute as "simple" must also specify a value for the **HREF** attribute. These two attributes fully specify a "simple" XML link. Thus, a typical MathML link might look like:

<mrow XML-LINK="simple" HREF="http://www.w3.org"> ... </mrow>

MathML designates that almost all elements can be used as an XML linking element. The only elements which cannot serve as linking elements are those such as the **<sep/>** element 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 which cannot be used as linking elements is given below in table 7.1.5.1.

<prescripts/>	<none/>	<sep/>
<power/>	<malignmark/>	<maligngroup/>

Table 7.1.5.1 MathML Elements Which Cannot Be Linking Elements

Images

The IMG element has no MathML equivalent. The decision to omit a general image inclusion mechanism in MathML was based on several factors. First, a simple mechanism for including images in MathML along the lines of the IMG element would not be more closely tied to mathematical content or notation than the HTML IMG element itself. Therefore, such an element would likely be superseded by the IMG element if it becomes possible to mix XML and HTML generally.

Another 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 while 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 by the creation of special symbol fonts, rather than simply including them as images.

Finally, apart from the introduction of new glyphs, many of the situations where one might be inclined to use an image amount to some sort of labeled diagram. For example, knot diagrams,

Venn diagrams, Dynkin diagrams, Feynman diagrams and complicated commutative diagrams all fall into this category. As such, their content would be better encoded via some combination of structured graphics and MathML markup. Because of the generality of the "labeled diagram" construction, the definition of a markup language to encode such constructions extends beyond the scope of the W3C Math activity. However, it may be possible to provide such functionality in a future extension of MathML.

7.2 Generating, Processing and Rendering MathML

Information is increasingly 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 also tied to the ability to process it with software tools.

Many different kinds of MathML editors, translators, processors and renderers will be implemented. In addition to supporting the MathML core language, it is reasonable to assume that some of these renderers will provide additional specialized capabilities. Consequently, it is important to specify what one can and cannot expect from a generic MathML compliant application, and in what ways MathML can be extended, or used to pass additional information directly to specific application that can take advantage of it.

7.2.1 MathML Compliance

It is important to clearly specify what it means to be a MathML compliant processor. Specifying MathML compliance serves two purposes. First, authors can be assured that their documents will be generally accessible if they refrain from using proprietary extensions. Second, software developers can be assured of the criteria for interoperability.

A well-formed MathML expression is a XML construct determined by the MathML DTD together with the additional requirements given in the specifications of the MathML document.

We define a "MathML processor" to mean any application that can accept, produce, or "roundtrip" a well-formed MathML expression. An 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.

We specify three forms of MathML compliance:

1. A MathML-input-compliant processor must accept all well-formed MathML expressions.

   For example a MathML-input-compliant validating parser which implements the MathML specification returns a truth value. A MathML-input-compliant renderer faithfully translates a MathML expression into application-specific form allowing native

application operations to be performed.

2. A MathML-output-compliant processor must generate well-formed MathML.
    - An embedded MathML-output-compliant processor must return well-formed MathML expressions when queried by the document object model API.
    - In the case where cut-and-paste/drag-and-drop operations are implemented, a MathML-output-compliant processor must return well-formed MathML expressions.
3. A MathML-roundtrip-compliant 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.

We note that being roundtrip-compliant may be very difficult for processors that convert MathML input into an internal form that is structurally very different from the XML expression model. The first generation of processors may very well be input-compliant and output-compliant, but not roundtrip-compliant. Nevertheless, we expect roundtrip-compliant processors to be eventually produced with the wide-spread acceptance of MathML.

Beyond the above, the MathML core specification makes no demands of individual processors. However, in order to guide developers, the MathML specification includes advisory material; for example, there are suggested rendering rules included in section 3. The remainder of this section makes additional suggestions about a number of interface issues a MathML processor should address in some fashion.

7.2.2 Handling of Errors

If a MathML-input-compliant application receives input containing one or more elements with an illegal number or type of attributes or children schemata, it should nonetheless attempt to render all the input in an intelligible way, i.e. to render normally those parts of the input which were well-formed, and to render error messages (rendered as if enclosed in an **<merror>** element) in place of ill-formed expressions.

MathML-output-compliant applications such as editors and translators may choose to generate **<merror>** expressions to signal errors in their input. This is usually preferable to generating well-formed, but possibly erroneous, MathML.

7.2.3 An Attribute for Unspecified Data

The MathML attributes described in the MathML specification are necessary for display and content markup. Ideally, the MathML attributes should be an open-ended list so that users could add specific attributes for specific renderers. However, this can't 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 nonstandard attributes while remaining strictly in compliance with the standard DTD.

To allow this, this specification also allows the attribute **other**="..." for all elements, for use as a hook to pass on renderer-specific information. In particular, it can be used as a hook for passing information to audio renderers, computer algebra systems, and for pattern matching in any future macro/extension mechanism. This idea is used in other languages. For example, Postscript comments are widely used to pass information that is not part of Postscript.

At the same time, the intent of the **other** attribute is not to encourage software developers to use this as a loophole for circumventing the MathML core markup conventions. We trust both authors and applications will use the **other** attribute judiciously.

The value of the **other** attribute should be a string containing an attribute list in valid XML format (i.e., attr1="val1" attr2="val2"; ..., with appropriate escaping of the double quotes). Renderers which accept nonstandard attributes directly should also accept them when they occur within the string value of the **other** attribute. This is not required for attributes specifically documented by the MathML standard.

7.3 Future Extensions

MathML is in its infancy; 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; as noted repeatedly in this chapter, the mechanisms for fully integrating MathML into HTML are not yet developed, and these mechanisms may have a significant impact on some aspects of MathML

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 "labeled diagram" facility. However, there will also no doubt be other desirable extensions to MathML which 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.

7.3.1 Macros and Style Sheets

The definition of a style sheet mechanism for XML is part of the ongoing XML activity at the World Wide Web Consortium. Although it is too soon to say what this mechanism will ultimately be like, it is likely that it will accommodate the needs of MathML. It is also possible that such a style sheet mechanism will be sufficiently powerful to provide basic macro capability as well.

Macros, however, play a very important and useful role in encoding mathematical content and meaning. Moreover, it is difficult to devise a coherent, general macro system for MathML, because there are so many distinct applications for MathML macros. Therefore, the W3C Math working group plans to investigate the definition of a macro mechanism specifically tailored to

MathML, in addition to participating in general ongoing XML style sheet and macro facility activities.

Some of the possible uses of MathML macros 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 applications. Narrowly defined disciplines should be able to easily introduce standardize content markup by using standard macro packages. For example, the OpenMath project could release macro packages for attaching OpenMath content markup up.
- **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 HTML Math content tags in a context sensitive way. Such a macroing 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 super 2 of f". Knowing the context to be multivariable calculus, the reader may wish to use a style sheet or macro package which instructs the renderer to render this **<msubsup>** element as "second derivative with respect to x of f".

7.3.2 XML Extensions to MathML

The set of elements and attributes specified in the MathML specification are necessary for rendering common math 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-compliant DTD, which necessarily limits the elements and attributes to those which occur in the DTD. Renderers desiring to accept nonstandard elements or attributes, and authors desiring to include these in documents, should accept or produce documents which conform to an appropriately extended XML-compliant DTD which has the standard MathML DTD as a subset.

MathML compliant renderers are allowed, but not required, to accept nonstandard elements and attributes, and to render them in any way. If a renderer does not accept some or all nonstandard tags, it is encouraged to either 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 normally.

Up: Table of Contents

Parsing MathML

MathML documents should be validated using the XML DTD below. Note in particular that the xml attribute **xml:space** is not used, so whitespace characters in element content (ie. outside the presentation token elements **mi, mo, mn, mtext, mspace, mtext, ms** , the content token elements **ci, cn** and **annotation**) are not significant.

If the MathML is parsed without a DTD (ie. as a well-formed XML fragment), it is the responsibility of the processing application to treat these whitespace characters as not significant.

An SGML parser (such as *nsgmls*) can be used to validate MathML. In this case an SGML declaration defining the constraints of XML applicable to an SGML parser must be used. See http://www.w3.org/TR/NOTE-sgml-xml .

The MathML DTD

A zip file of the full DTD including entity declarations is provided for reference. Here we give the main body of the DTD, without including the entity declarations. See Chapter 6 for a list of entity names ordered by name or by unicode value.

```
<!-- Content model for content and presentation        -->
<!--  and browser interface tags in MathML              -->
<!-- initial draft 9.May.1997          syntax = XML     -->
<!-- author = s.buswell sb@stilo.demon.co.uk            -->
<!--                                                    -->
<!-- revised 14.May.1997 by Robert Miner                -->
<!-- revised 29.June.1997 and 2.July.1997 by s.buswell  -->
<!--                                                    -->
<!-- revised 15.December.1997  by s.buswell             -->
<!-- revised 8.February.1998   by s.buswell             -->
<!-- revised 4.april.1998      by s.buswell             -->
<!-- 21.February.1999 entities and small revisions by d.carlisle    -->
<!--                                                    -->
<!-- W3C Recommendation      7 April 1998                          -->
<!-- ***********************************************  -->

<!-- general attribute definitions for class & style & id & other -->
<!-- : attributes shared by all mathml elements        -->

<!ENTITY % att-globalatts      'class CDATA #IMPLIED
                                style CDATA #IMPLIED
                                id    ID    #IMPLIED
                                other CDATA #IMPLIED'  >

<!-- ***********************************************  -->
<!-- Presentation element set                          -->
```

```
<!-- presentation attribute definitions -->

<!ENTITY % att-fontsize        'fontsize CDATA #IMPLIED'             >
<!ENTITY % att-fontweight      'fontweight (normal | bold) #IMPLIED'  >
<!ENTITY % att-fontstyle       'fontstyle (normal | italic) #IMPLIED' >
<!ENTITY % att-fontfamily      'fontfamily CDATA   #IMPLIED'         >
<!ENTITY % att-color           'color CDATA   #IMPLIED'              >

<!ENTITY % att-fontinfo      '%att-fontsize;
                             %att-fontweight;
                             %att-fontstyle;
                             %att-fontfamily;
                             %att-color;'  >

<!ENTITY % att-form        'form (prefix | infix | postfix) #IMPLIED' >
<!ENTITY % att-fence       'fence (true | false ) #IMPLIED'    >
<!ENTITY % att-separator   'separator (true | false ) #IMPLIED'    >
<!ENTITY % att-lspace      'lspace CDATA  #IMPLIED'       >
<!ENTITY % att-rspace      'rspace CDATA  #IMPLIED'       >
<!ENTITY % att-stretchy    'stretchy (true | false ) #IMPLIED'     >
<!ENTITY % att-symmetric   'symmetric (true | false ) #IMPLIED'    >
<!ENTITY % att-maxsize     'maxsize CDATA #IMPLIED'       >
<!ENTITY % att-minsize     'minsize CDATA #IMPLIED'       >
<!ENTITY % att-largeop           'largeop (true | false ) #IMPLIED' >
<!ENTITY % att-movablelimits     'movablelimits (true | false )
#IMPLIED' >
<!ENTITY % att-accent              'accent (true | false)  #IMPLIED'>

<!ENTITY % att-opinfo '%att-form;
                       %att-fence;
                       %att-separator;
                       %att-lspace;
                       %att-rspace;
                       %att-stretchy;
                       %att-symmetric;
                       %att-maxsize;
                       %att-minsize;
                       %att-largeop;
                       %att-movablelimits;
                       %att-accent;'         >

<!ENTITY % att-width             'width CDATA #IMPLIED'   >
<!ENTITY % att-height            'height CDATA #IMPLIED' >
<!ENTITY % att-depth             'depth CDATA #IMPLIED'   >

<!ENTITY % att-sizeinfo          '%att-width;
                                  %att-height;
                                  %att-depth;'          >

<!ENTITY % att-lquote              'lquote CDATA #IMPLIED'   >
<!ENTITY % att-rquote              'rquote CDATA #IMPLIED'   >
```

```
<!ENTITY % att-linethickness        'linethickness CDATA #IMPLIED' >

<!ENTITY % att-scriptlevel          'scriptlevel CDATA #IMPLIED'>
<!ENTITY % att-displaystyle         'displaystyle (true | false)
#IMPLIED'>
<!ENTITY % att-scriptsizemultiplier 'scriptsizemultiplier CDATA
#IMPLIED'  >
<!ENTITY % att-scriptminsize        'scriptminsize CDATA #IMPLIED'>
<!ENTITY % att-background           'background CDATA #IMPLIED' >

<!ENTITY % att-open                 'open CDATA #IMPLIED' >
<!ENTITY % att-close                'close CDATA #IMPLIED' >
<!ENTITY % att-separators           'separators CDATA #IMPLIED' >

<!ENTITY % att-subscriptshift       'subscriptshift CDATA #IMPLIED'>
<!ENTITY % att-superscriptshift     'superscriptshift CDATA #IMPLIED'    >

<!ENTITY % att-accentunder          'accentunder (true | false)
#IMPLIED'>

<!ENTITY % att-align                'align CDATA #IMPLIED'           >
<!ENTITY % att-rowalign             'rowalign CDATA #IMPLIED'    >
<!ENTITY % att-columnalign          'columnalign CDATA #IMPLIED'   >
<!ENTITY % att-groupalign           'groupalign CDATA #IMPLIED'    >
<!ENTITY % att-alignmentscope       'alignmentscope CDATA #IMPLIED'  >

<!ENTITY % att-rowspacing           'rowspacing CDATA #IMPLIED'      >
<!ENTITY % att-columnspacing        'columnspacing CDATA #IMPLIED'   >
<!ENTITY % att-rowlines             'rowlines CDATA #IMPLIED'           >
<!ENTITY % att-columnlines          'columnlines CDATA #IMPLIED'         >
<!ENTITY % att-frame                'frame (none | solid | dashed)
#IMPLIED'  >
<!ENTITY % att-framespacing         'framespacing CDATA #IMPLIED'       >
<!ENTITY % att-equalrows            'equalrows CDATA #IMPLIED'         >
<!ENTITY % att-equalcolumns         'equalcolumns CDATA #IMPLIED'     >

<!ENTITY % att-tableinfo           '%att-align;
                                    %att-rowalign;
                                    %att-columnalign;
                                    %att-groupalign;
                                    %att-alignmentscope;
                                    %att-rowspacing;
                                    %att-columnspacing;
                                    %att-rowlines;
                                    %att-columnlines;
                                    %att-frame;
                                    %att-framespacing;
                                    %att-equalrows;
                                    %att-equalcolumns;
                                    %att-displaystyle;'        >
```

```
<!ENTITY % att-rowspan              'rowspan CDATA #IMPLIED'         >
<!ENTITY % att-columnspan           'columnspan CDATA #IMPLIED'      >

<!ENTITY % att-edge                  'edge (left | right)  #IMPLIED ' >

<!ENTITY % att-actiontype            'actiontype CDATA #IMPLIED' >
<!ENTITY % att-selection             'selection CDATA #IMPLIED ' >

<!-- presentation token schemata with content-->

<!ENTITY % ptoken "mi | mn | mo | mtext | ms" >

<!ATTLIST mi        %att-fontinfo;
                    %att-globalatts;    >

<!ATTLIST mn        %att-fontinfo;
                    %att-globalatts;    >

<!ATTLIST mo        %att-fontinfo;
                    %att-opinfo;
                    %att-globalatts;     >

<!ATTLIST mtext     %att-fontinfo;
                    %att-globalatts;     >

<!ATTLIST ms        %att-fontinfo;
                    %att-lquote;
                    %att-rquote;
                    %att-globalatts;          >

<!-- empty presentation token schemata -->

<!ENTITY % petoken  "mspace" >
<!ELEMENT mspace     EMPTY  >

<!ATTLIST mspace    %att-sizeinfo;
                    %att-globalatts;       >

<!-- presentation general layout schemata -->

<!ENTITY % pgenschema "mrow|mfrac|msqrt|mroot|
                    mstyle|merror|mpadded| mphantom|mfenced" >

<!ATTLIST mrow      %att-globalatts;   >

<!ATTLIST mfrac     %att-linethickness;
                    %att-globalatts; >

<!ATTLIST msqrt     %att-globalatts;   >
```

```
<!ATTLIST mroot     %att-globalatts;   >

<!ATTLIST mstyle    %att-fontinfo;
                    %att-opinfo;
                    %att-lquote;
                    %att-rquote;
                    %att-linethickness;
                    %att-scriptlevel;
                    %att-scriptsizemultiplier;
                    %att-scriptminsize;
                                        %att-background;
                                        %att-open;
                                        %att-close;
                                        %att-separators;
                    %att-subscriptshift;
                    %att-superscriptshift;
                    %att-accentunder;
                    %att-tableinfo;
                    %att-rowspan;
                    %att-columnspan;
                                        %att-edge;
                                        %att-actiontype;
                                        %att-selection;
                    %att-globalatts;          >

<!ATTLIST merror    %att-globalatts;   >

<!ATTLIST mpadded   %att-sizeinfo;
                    %att-lspace;
                    %att-globalatts;     >

<!ATTLIST mphantom  %att-globalatts;     >

<!ATTLIST mfenced   %att-open;
                    %att-close;
                    %att-separators;
                    %att-globalatts;     >

<!-- presentation  layout schemata : scripts and limits -->

<!ENTITY % pscrschema  "msub|msup|msubsup|
                    munder|mover|munderover|mmultiscripts" >

<!ATTLIST msub      %att-subscriptshift;
                    %att-globalatts;     >

<!ATTLIST msup      %att-superscriptshift;
                    %att-globalatts;          >

<!ATTLIST msubsup   %att-subscriptshift;
                    %att-superscriptshift;
```

```
                    %att-globalatts;          >

<!ATTLIST munder    %att-accentunder;
                    %att-globalatts;          >

<!ATTLIST mover     %att-accent;
                    %att-globalatts;           >

<!ATTLIST munderover %att-accent;
                     %att-accentunder;
                     %att-globalatts;      >

<!ATTLIST mmultiscripts
                    %att-subscriptshift;
                    %att-superscriptshift;
                    %att-globalatts;          >

<!-- presentation layout schemata: script empty elements -->

<!ENTITY % pscreschema "mprescripts|none" >

<!ELEMENT mprescripts    EMPTY          >
<!ATTLIST mprescripts   %att-globalatts;      >

<!ELEMENT none          EMPTY          >
<!ATTLIST none          %att-globalatts;      >

<!-- presentation layout schemata: tables -->

<!ENTITY % ptabschema "mtable|mtr|mtd" >

<!ATTLIST mtable    %att-tableinfo;
                    %att-globalatts;      >

<!ATTLIST mtr       %att-rowalign;
                    %att-columnalign;
                    %att-groupalign;
                    %att-globalatts; >

<!ATTLIST mtd       %att-rowalign;
                    %att-columnalign;
                    %att-groupalign;
                    %att-rowspan;
                    %att-columnspan;
                    %att-globalatts;   >

<!ENTITY % plschema  "%pgenschema;|%pscrschema;|%ptabschema;" >

<!-- empty presentation layout schemata -->

<!ENTITY % peschema "maligngroup | malignmark" >
```

```
<!ELEMENT  malignmark  EMPTY          >

<!ATTLIST malignmark    %att-edge;
                        %att-globalatts;      >

<!ELEMENT maligngroup    EMPTY          >
<!ATTLIST maligngroup   %att-groupalign;
                        %att-globalatts;      >

<!-- presentation action schemata -->

<!ENTITY % pactions "maction" >
<!ATTLIST maction   %att-actiontype;
                    %att-selection;
                    %att-globalatts;     >

<!-- Presentation entity for substitution into content tag constructs -->
<!-- excludes elements which are not valid as expressions             -->

<!ENTITY % PresInCont    "%ptoken; | %petoken; |
                          %plschema; | %peschema; | %pactions;">

<!-- Presentation entity - all presentation constructs -->

<!ENTITY % Presentation  "%ptoken; | %petoken; | %pscreschema; |
                          %plschema; | %peschema; | %pactions;">

<!-- ************************************************  -->
<!-- Content element set                                -->
<!-- attribute definitions                              -->

<!ENTITY % att-base              'base CDATA "10"'           >
<!ENTITY % att-closure           'closure CDATA "closed"'    >
<!ENTITY % att-definition        'definitionURL CDATA ""'    >
<!ENTITY % att-encoding          'encoding CDATA ""'         >
<!ENTITY % att-nargs             'nargs CDATA "1"'           >
<!ENTITY % att-occurrence        'occurrence CDATA "function-model"'  >
<!ENTITY % att-order             'order CDATA "numeric"'         >
<!ENTITY % att-scope             'scope CDATA "local"'         >
<!ENTITY % att-type              'type CDATA #IMPLIED'         >

<!-- content leaf token elements -->

<!ENTITY % ctoken "ci | cn" >

<!ATTLIST ci   %att-type;
               %att-globalatts;     >

<!ATTLIST cn   %att-type;
               %att-base;
```

```
               %att-globalatts;     >

<!-- content elements - specials -->

<!ENTITY % cspecial "apply | reln | lambda" >

<!ATTLIST apply   %att-globalatts;     >

<!ATTLIST reln    %att-globalatts;     >

<!ATTLIST lambda  %att-globalatts;     >

<!-- content elements - others -->

<!ENTITY % cother "condition | declare | sep" >

<!ATTLIST condition       %att-globalatts;     >

<!ATTLIST declare         %att-type;
                          %att-scope;
                          %att-nargs;
                          %att-occurrence;
                          %att-definition;
                          %att-globalatts;     >

<!ELEMENT sep       EMPTY >
<!ATTLIST sep       %att-globalatts;     >

<!-- content elements - semantic mapping -->

<!ENTITY % csemantics "semantics | annotation | annotation-xml" >

<!ATTLIST semantics         %att-definition;
                            %att-globalatts;     >

<!ATTLIST annotation        %att-encoding;
                            %att-globalatts;     >

<!ATTLIST annotation-xml    %att-encoding;
                            %att-globalatts;     >

<!-- content elements - constructors -->

<!ENTITY % cconstructor "interval | list | matrix | matrixrow | set |
vector" >

<!ATTLIST interval     %att-closure;
                       %att-globalatts;     >

<!ATTLIST set          %att-globalatts;     >
```

```
<!ATTLIST list         %att-order;
                       %att-globalatts;     >

<!ATTLIST vector       %att-globalatts;     >

<!ATTLIST matrix       %att-globalatts;     >

<!ATTLIST matrixrow    %att-globalatts;     >

<!-- content elements - operators -->

<!ENTITY % cfuncop1ary "inverse | ident " >

<!ELEMENT inverse          EMPTY       >
<!ATTLIST inverse     %att-definition;
                       %att-globalatts;     >

<!ENTITY % cfuncopnary "fn |  compose" >

<!ATTLIST fn     %att-definition;
                 %att-globalatts;     >

<!ELEMENT ident        EMPTY       >
<!ATTLIST ident        %att-definition;
                       %att-globalatts;     >

<!ELEMENT compose      EMPTY       >
<!ATTLIST compose      %att-definition;
                       %att-globalatts;     >

<!ENTITY % carithop1ary "abs | conjugate | exp | factorial" >

<!ELEMENT exp          EMPTY       >
<!ATTLIST exp          %att-definition;
                       %att-globalatts;     >

<!ELEMENT abs          EMPTY       >
<!ATTLIST abs          %att-definition;
                       %att-globalatts;     >

<!ELEMENT conjugate    EMPTY       >
<!ATTLIST conjugate    %att-definition;
                       %att-globalatts;     >

<!ELEMENT factorial    EMPTY       >
<!ATTLIST factorial    %att-definition;
                       %att-globalatts;     >

<!ENTITY % carithop1or2ary "minus" >

<!ELEMENT minus        EMPTY       >
```

```
<!ATTLIST minus       %att-definition;
                      %att-globalatts;     >

<!ENTITY % carithop2ary "quotient | divide | power | rem" >

<!ELEMENT quotient       EMPTY       >
<!ATTLIST quotient       %att-definition;
                         %att-globalatts;     >

<!ELEMENT divide       EMPTY       >
<!ATTLIST divide       %att-definition;
                       %att-globalatts;     >

<!ELEMENT power          EMPTY       >
<!ATTLIST power       %att-definition;
                      %att-globalatts;     >

<!ELEMENT rem          EMPTY       >
<!ATTLIST rem        %att-definition;
                     %att-globalatts;     >

<!ENTITY % carithopnary "plus | times | max | min | gcd" >

<!ELEMENT plus          EMPTY       >
<!ATTLIST plus        %att-definition;
                      %att-globalatts;     >

<!ELEMENT max          EMPTY       >
<!ATTLIST max         %att-definition;
                      %att-globalatts;     >

<!ELEMENT min          EMPTY       >
<!ATTLIST min         %att-definition;
                      %att-globalatts;     >

<!ELEMENT times          EMPTY       >
<!ATTLIST times     %att-definition;
                    %att-globalatts;     >

<!ELEMENT gcd          EMPTY       >
<!ATTLIST gcd     %att-definition;
                  %att-globalatts;     >

<!ENTITY % carithoproot "root" >

<!ELEMENT root          EMPTY       >
<!ATTLIST root     %att-definition;
                   %att-globalatts;     >

<!ENTITY % clogicopquant "exists | forall" >
```

```
<!ELEMENT exists          EMPTY       >
<!ATTLIST exists     %att-definition;
                      %att-globalatts;     >

<!ELEMENT forall          EMPTY       >
<!ATTLIST forall     %att-definition;
                      %att-globalatts;     >

<!ENTITY % clogicopnary "and | or | xor" >

<!ELEMENT and          EMPTY       >
<!ATTLIST and     %att-definition;
                      %att-globalatts;     >

<!ELEMENT or          EMPTY       >
<!ATTLIST or     %att-definition;
                      %att-globalatts;     >

<!ELEMENT xor          EMPTY       >
<!ATTLIST xor     %att-definition;
                      %att-globalatts;     >

<!ENTITY % clogicop1ary "not" >

<!ELEMENT not          EMPTY       >
<!ATTLIST not     %att-definition;
                      %att-globalatts;     >

<!ENTITY % clogicop2ary "implies" >

<!ELEMENT implies          EMPTY       >
<!ATTLIST implies     %att-definition;
                      %att-globalatts;     >

<!ENTITY % ccalcop "log | int | diff | partialdiff" >

<!ELEMENT log          EMPTY       >
<!ATTLIST log     %att-definition;
                      %att-globalatts;     >

<!ELEMENT int          EMPTY       >
<!ATTLIST int     %att-definition;
                      %att-globalatts;     >

<!ELEMENT diff          EMPTY       >
<!ATTLIST diff     %att-definition;
                      %att-globalatts;     >

<!ELEMENT partialdiff          EMPTY       >
<!ATTLIST partialdiff     %att-definition;
                      %att-globalatts;     >
```

```
<!ENTITY % ccalcop1ary "ln" >

<!ELEMENT ln          EMPTY       >
<!ATTLIST ln     %att-definition;
                      %att-globalatts;     >

<!ENTITY % csetop2ary "setdiff" >

<!ELEMENT setdiff          EMPTY       >
<!ATTLIST setdiff     %att-definition;
                      %att-globalatts;     >

<!ENTITY % csetopnary "union | intersect" >

<!ELEMENT union          EMPTY       >
<!ATTLIST union     %att-definition;
                      %att-globalatts;     >

<!ELEMENT intersect          EMPTY       >
<!ATTLIST intersect     %att-definition;
                      %att-globalatts;     >

<!ENTITY % cseqop "sum | product | limit" >

<!ELEMENT sum          EMPTY       >
<!ATTLIST sum     %att-definition;
                      %att-globalatts;     >

<!ELEMENT product          EMPTY       >
<!ATTLIST product     %att-definition;
                      %att-globalatts;     >

<!ELEMENT limit          EMPTY       >
<!ATTLIST limit     %att-definition;
                      %att-globalatts;     >

<!ENTITY % ctrigop "sin | cos | tan | sec | csc | cot | sinh
                | cosh | tanh | sech | csch | coth
                | arcsin | arccos | arctan" >

<!ELEMENT sin          EMPTY       >
<!ATTLIST sin     %att-definition;
                      %att-globalatts;     >

<!ELEMENT cos          EMPTY       >
<!ATTLIST cos     %att-definition;
                      %att-globalatts;     >

<!ELEMENT tan          EMPTY       >
<!ATTLIST tan     %att-definition;
```

```
                    %att-globalatts;     >

<!ELEMENT sec          EMPTY       >
<!ATTLIST sec     %att-definition;
                    %att-globalatts;     >

<!ELEMENT csc          EMPTY       >
<!ATTLIST csc     %att-definition;
                    %att-globalatts;     >

<!ELEMENT cot          EMPTY       >
<!ATTLIST cot     %att-definition;
                    %att-globalatts;     >

<!ELEMENT sinh          EMPTY       >
<!ATTLIST sinh     %att-definition;
                    %att-globalatts;     >

<!ELEMENT cosh          EMPTY       >
<!ATTLIST cosh     %att-definition;
                    %att-globalatts;     >

<!ELEMENT tanh          EMPTY       >
<!ATTLIST tanh     %att-definition;
                    %att-globalatts;     >

<!ELEMENT sech          EMPTY       >
<!ATTLIST sech     %att-definition;
                    %att-globalatts;     >

<!ELEMENT csch          EMPTY       >
<!ATTLIST csch     %att-definition;
                    %att-globalatts;     >

<!ELEMENT coth          EMPTY       >
<!ATTLIST coth     %att-definition;
                    %att-globalatts;     >

<!ELEMENT arcsin          EMPTY       >
<!ATTLIST arcsin     %att-definition;
                    %att-globalatts;     >

<!ELEMENT arccos          EMPTY       >
<!ATTLIST arccos     %att-definition;
                    %att-globalatts;     >

<!ELEMENT arctan          EMPTY       >
<!ATTLIST arctan     %att-definition;
                    %att-globalatts;     >

<!ENTITY % cstatopnary "mean | sdev | variance | median | mode" >
```

```
<!ELEMENT mean          EMPTY       >
<!ATTLIST mean     %att-definition;
                      %att-globalatts;     >

<!ELEMENT sdev          EMPTY       >
<!ATTLIST sdev     %att-definition;
                      %att-globalatts;     >

<!ELEMENT variance          EMPTY       >
<!ATTLIST variance     %att-definition;
                      %att-globalatts;     >

<!ELEMENT median          EMPTY       >
<!ATTLIST median     %att-definition;
                      %att-globalatts;     >

<!ELEMENT mode          EMPTY       >
<!ATTLIST mode     %att-definition;
                      %att-globalatts;     >

<!ENTITY % cstatopmoment "moment" >

<!ELEMENT moment          EMPTY       >
<!ATTLIST moment     %att-definition;
                      %att-globalatts;     >

<!ENTITY % clalgop1ary "determinant | transpose" >

<!ELEMENT determinant          EMPTY       >
<!ATTLIST determinant     %att-definition;
                      %att-globalatts;     >

<!ELEMENT transpose          EMPTY       >
<!ATTLIST transpose     %att-definition;
                      %att-globalatts;     >

<!ENTITY % clalgopnary "selector" >

<!ELEMENT selector          EMPTY       >
<!ATTLIST selector     %att-definition;
                      %att-globalatts;     >

<!-- content elements -  relations  -->

<!ENTITY % cgenrel2ary "neq" >

<!ELEMENT neq          EMPTY       >
<!ATTLIST neq     %att-definition;
                      %att-globalatts;     >
```

```
<!ENTITY % cgenrelnary "eq | leq | lt | geq | gt" >

<!ELEMENT eq          EMPTY       >
<!ATTLIST eq     %att-definition;
                      %att-globalatts;     >

<!ELEMENT gt          EMPTY       >
<!ATTLIST gt     %att-definition;
                      %att-globalatts;     >

<!ELEMENT lt          EMPTY       >
<!ATTLIST lt     %att-definition;
                      %att-globalatts;     >

<!ELEMENT geq          EMPTY       >
<!ATTLIST geq     %att-definition;
                      %att-globalatts;     >

<!ELEMENT leq          EMPTY       >
<!ATTLIST leq     %att-definition;
                      %att-globalatts;     >

<!ENTITY % csetrel2ary "in | notin | notsubset | notprsubset" >

<!ELEMENT in          EMPTY       >
<!ATTLIST in     %att-definition;
                      %att-globalatts;     >

<!ELEMENT notin          EMPTY       >
<!ATTLIST notin     %att-definition;
                      %att-globalatts;     >

<!ELEMENT notsubset          EMPTY       >
<!ATTLIST notsubset     %att-definition;
                      %att-globalatts;     >

<!ELEMENT notprsubset          EMPTY       >
<!ATTLIST notprsubset     %att-definition;
                      %att-globalatts;     >

<!ENTITY % csetrelnary "subset | prsubset" >

<!ELEMENT subset          EMPTY       >
<!ATTLIST subset     %att-definition;
                      %att-globalatts;     >

<!ELEMENT prsubset          EMPTY       >
<!ATTLIST prsubset     %att-definition;
                      %att-globalatts;     >

<!ENTITY % cseqrel2ary "tendsto" >
```

```
<!ELEMENT tendsto          EMPTY       >
<!ATTLIST tendsto     %att-definition;
                      %att-type;
                      %att-globalatts;     >

<!-- content elements -  quantifiers  -->

<!ENTITY % cquantifier "lowlimit | uplimit | bvar | degree | logbase" >

<!ATTLIST lowlimit  %att-globalatts;     >

<!ATTLIST uplimit  %att-globalatts;     >

<!ATTLIST bvar     %att-globalatts;     >

<!ATTLIST degree   %att-globalatts;     >

<!ATTLIST logbase  %att-globalatts;     >

<!-- operator groups -->

<!ENTITY % cop1ary "%cfuncop1ary; | %carithop1ary; | %clogicop1ary;
                    | %ccalcop1ary; | %ctrigop; | %clalgop1ary; " >

<!ENTITY % cop2ary "%carithop2ary; | %clogicop2ary;| %csetop2ary; " >

<!ENTITY % copnary "%cfuncopnary; | %carithopnary; | %clogicopnary;
                    | %csetopnary; | %cstatopnary;  | %clalgopnary; " >

<!ENTITY % copmisc "%carithoproot; | %carithop1or2ary; | %ccalcop;
                    | %cseqop; | %cstatopmoment; | %clogicopquant;" >

<!-- relation groups -->

<!ENTITY % crel2ary "%cgenrel2ary;  | %csetrel2ary; | %cseqrel2ary;   " >

<!ENTITY % crelnary "%cgenrelnary; | %csetrelnary;" >

<!-- content constructs - all -->

<!ENTITY % Content "%ctoken; | %cspecial;  | %cother;  | %csemantics;
                   |%cconstructor; | %cquantifier;
                   |%cop1ary; |%cop2ary; |%copnary; |%copmisc;
                   |%crel2ary; |%crelnary;" >

<!-- content constructs for substitution in presentation structures -->

<!ENTITY % ContInPres "ci | cn  | apply  | fn | lambda | reln
                   | interval | list | matrix |matrixrow
                   | set | vector | semantics" > <!--dpc-->
```

```
<!-- ********************************************  -->

<!-- recursive definition for content of expressions            -->
<!-- include presentation tag constructs at lowest level        -->
<!-- so presentation layout schemata hold presentation or Content -->
<!-- include Content tag constructs at lowest level             -->
<!-- so Content tokens hold PCDATA or Presentation at leaf level -->
<!-- (for permitted substitutable elements in context)          -->

<!ENTITY % ContentExpression   "(%Content; | %PresInCont;)* "     >
<!ENTITY % PresExpression      "(%Presentation; | %ContInPres;)* "     >
<!ENTITY % MathExpression      "(%PresInCont; | %ContInPres;)* "     >

<!--  content token elements (may hold embedded presentation constructs)
-->

<!ELEMENT ci       (#PCDATA | %PresInCont;)*   >
<!ELEMENT cn       (#PCDATA | sep | %PresInCont;)*   >

<!--  content special elements    -->

<!ELEMENT apply      (%ContentExpression;)   >
<!ELEMENT reln       (%ContentExpression;)   >
<!ELEMENT lambda     (%ContentExpression;)   >

<!--  content other elements    -->

<!ELEMENT condition      (%ContentExpression;)   >
<!ELEMENT declare        (%ContentExpression;)   >

<!--  content semantics elements    -->

<!ELEMENT semantics              (%ContentExpression;)   >
<!ELEMENT annotation             (#PCDATA)                >
<!ELEMENT annotation-xml         (%ContentExpression;)   >

<!--  content constructor elements    -->

<!ELEMENT interval       (%ContentExpression;)   >
<!ELEMENT set            (%ContentExpression;)   >
<!ELEMENT list           (%ContentExpression;)   >
<!ELEMENT vector         (%ContentExpression;)   >
<!ELEMENT matrix         (%ContentExpression;)   >
<!ELEMENT matrixrow      (%ContentExpression;)   >

<!--  content operator element (user-defined)    -->

<!ELEMENT fn       (%ContentExpression;)   >

<!--  content quantifier elements    -->
```

```
<!ELEMENT lowlimit       (%ContentExpression;)   >
<!ELEMENT uplimit        (%ContentExpression;)   >
<!ELEMENT bvar           (%ContentExpression;)   >
<!ELEMENT degree         (%ContentExpression;)   >
<!ELEMENT logbase        (%ContentExpression;)   >

<!-- ************************************************  -->
<!-- presentation layout schema contain tokens, layout and content
schema  -->

<!ELEMENT mstyle          (%PresExpression;)                    >
<!ELEMENT merror          (%PresExpression;)                    >
<!ELEMENT mphantom        (%PresExpression;)                    >
<!ELEMENT mrow            (%PresExpression;)                    >
<!ELEMENT mfrac           (%PresExpression;)                    >
<!ELEMENT msqrt           (%PresExpression;)                    >
<!ELEMENT mroot           (%PresExpression;)                    >
<!ELEMENT msub            (%PresExpression;)                    >
<!ELEMENT msup            (%PresExpression;)                    >
<!ELEMENT msubsup         (%PresExpression;)                    >
<!ELEMENT mmultiscripts   (%PresExpression;)                    >
<!ELEMENT munder          (%PresExpression;)                    >
<!ELEMENT mover           (%PresExpression;)                    >
<!ELEMENT munderover      (%PresExpression;)                    >
<!ELEMENT mtable          (%PresExpression;)                    >
<!ELEMENT mtr             (%PresExpression;)                    >
<!ELEMENT mtd             (%PresExpression;)                    >
<!ELEMENT maction         (%PresExpression;)                    >
<!ELEMENT mfenced         (%PresExpression;)                    >
<!ELEMENT mpadded         (%PresExpression;)                    >

<!-- presentation tokens contain PCDATA or malignmark constructs -->

<!ELEMENT mi     (#PCDATA | malignmark )*        >
<!ELEMENT mn     (#PCDATA | malignmark )*        >
<!ELEMENT mo     (#PCDATA | malignmark )*        >
<!ELEMENT mtext  (#PCDATA | malignmark )*        >
<!ELEMENT ms     (#PCDATA | malignmark )*        >

<!-- ************************************************  -->
<!-- browser interface definition -->

<!-- attributes for top level math element -->

<!ENTITY  %     att-macros      'macros CDATA #IMPLIED' >
<!ENTITY  %     att-mode        'mode   CDATA #IMPLIED' >

<!ENTITY  %     att-topinfo    '%att-globalatts;
                                %att-macros;
                                %att-mode;'     >
```

```
<!-- attributes for browser interface element element -->

<!ENTITY  %     att-name        'name CDATA #IMPLIED' >
<!ENTITY  %     att-baseline    'baseline CDATA #IMPLIED' >
<!ENTITY  %     att-overflow    'overflow
(scroll|elide|truncate|scale) "scroll"' >
<!ENTITY  %     att-altimg      'altimg CDATA #IMPLIED' >
<!ENTITY  %     att-alttext     'alttext CDATA #IMPLIED' >

<!ENTITY  %     att-browif      '%att-type;
                                %att-name;
                                %att-height;
                                %att-width;
                                %att-baseline;
                                %att-overflow;
                                %att-altimg;
                                %att-alttext; '     >

<!-- the top level math  element      -->
<!-- math  contains MathML encoded mathematics -->
<!-- math has the browser info attributes iff it is the
    browser interface element also -->

<!ELEMENT math       (%MathExpression;)    >

<!ATTLIST  math         %att-topinfo;
                        %att-browif;   >


<!-- ENTITY sets -->

<!-- ISO 9573-13 -->

<!ENTITY % ent-isoamsa SYSTEM "isoamsa.ent" >
%ent-isoamsa;

<!ENTITY % ent-isoamsb SYSTEM "isoamsb.ent" >
%ent-isoamsb;

<!ENTITY % ent-isoamsc SYSTEM "isoamsc.ent" >
%ent-isoamsc;

<!ENTITY % ent-isoamsn SYSTEM "isoamsn.ent" >
%ent-isoamsn;

<!ENTITY % ent-isoamso SYSTEM "isoamso.ent" >
%ent-isoamso;

<!ENTITY % ent-isoamsr SYSTEM "isoamsr.ent" >
%ent-isoamsr;
```

```
<!ENTITY % ent-isogrk3 SYSTEM "isogrk3.ent" >
%ent-isogrk3;

<!ENTITY % ent-isogrk4 SYSTEM "isogrk4.ent" >
%ent-isogrk4;

<!ENTITY % ent-isomfrk SYSTEM "isomfrk.ent" >
%ent-isomfrk;

<!ENTITY % ent-isomopf SYSTEM "isomopf.ent" >
%ent-isomopf;

<!ENTITY % ent-isomscr SYSTEM "isomscr.ent" >
%ent-isomscr;

<!ENTITY % ent-isotech SYSTEM "isotech.ent" >
%ent-isotech;



<!-- ISO 8879 -->

<!ENTITY % ent-isobox SYSTEM "isobox.ent" >
%ent-isobox;

<!ENTITY % ent-isocyr1 SYSTEM "isocyr1.ent" >
%ent-isocyr1;

<!ENTITY % ent-isocyr2 SYSTEM "isocyr2.ent" >
%ent-isocyr2;

<!ENTITY % ent-isodia SYSTEM "isodia.ent" >
%ent-isodia;

<!ENTITY % ent-isogrk1 SYSTEM "isogrk1.ent" >
%ent-isogrk1;

<!ENTITY % ent-isogrk2 SYSTEM "isogrk2.ent" >
%ent-isogrk2;

<!ENTITY % ent-isolat1 SYSTEM "isolat1.ent" >
%ent-isolat1;

<!ENTITY % ent-isolat2 SYSTEM "isolat2.ent" >
%ent-isolat2;

<!ENTITY % ent-isonum SYSTEM "isonum.ent" >
%ent-isonum;

<!ENTITY % ent-isopub SYSTEM "isopub.ent" >
```

```
%ent-isopub;

<!-- MathML aliases for characters defined above -->

<!ENTITY % ent-mmlalias SYSTEM "mmlalias.ent" >
%ent-mmlalias;


<!-- MathML new characters -->

<!ENTITY % ent-mmlextra SYSTEM "mmlextra.ent" >
%ent-mmlextra;

<!-- end of ENTITY sets -->
<!-- end of DTD fragment -->
<!-- ************************************************  -->
```

Up: Table of Contents

Glossary

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, characters that can stretch such as parenthesis, brackets, braces, summation signs, etc., and operators 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 mechanism 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.

Character Data (CDATA)
A SGML/XML data type for raw data which 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 know 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 which explicitly specify the mathematical meaning of a portion of a MathML expression (defined in Chapter 4 of the MathML standard).

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 which surrounds that 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 subexpression (of a MathML expression "E")
A subexpression which is directly contained in E.

Directly contained (element A in element B)
A is a child of B (as defined in XML); i.e. A is contained in B, but not in any element which 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 (Member Only)) of the W3C.

Document Style Semantics and Specification Language (DSSSL)
A method of specify the formatting and transformation of SGML documents. ISO International Standard 10179:1996.

Document Type Definition (DTD)
In SGML or XML 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 which is "inferred" around the contents of certain layout schemata when they have other than exactly one argument. Defined precisely in Section 3.1.5.

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 which 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 which contains the operator itself. Defined precisely in Section 3.2.4.

Entity reference
A sequence of ASCII characters of the form &*name*; which represents some other data, typically a non-ASCII character, a sequence of characters, or an external source of data, eg. a file containing a set of standard entity definitions such as ISOLat1.

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, eg "Times Roman Bold 12 point".

Glyph
The actual shape (bit pattern, outline) of a character image.

Input syntax layer
A planned MathML extension mechanism designed to facilitate hand entry of MathML content.

Indirectly contained
A is contained in B, but not directly contained in B.

Instance of MathML
A single instance of the toplevel 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 Sections 3.3 - 3.6, other than the empty elements defined there (i.e. not the elements defined in 3.5.4 (about alignment) or the empty elements [none/] and [mprescripts/] defined in 3.4.7 (about <mmultiscripts>)). 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 argument count requirements and the requirement for a certain empty element in <mmultiscripts>).

Mathematical Markup Language (MathML)
The markup language (specified in this document) for describing mathematical expression structure, together with a mathematical context.

MathML element

An XML element which forms part of the logical structure of a MathML document

MathML expression (within some well-formed 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 Section 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 multi-media 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. (<mo>, a token element, is defined in Section 3.2.4.)

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 recognised.

Pt
Point (pt), 1 pt = 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 section 4.2.3 and used with the apply construct to build function applications.

Presentation elements
MathML tags and entities intended to express the syntactic structure of math notation (defined in Chapter 3 of the MathML standard).

Presentation layout schema
A presentation element that can have other MathML elements as content.

Presentation token elements
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.

Scope of a Declaration
The portion of a MathML document to over which a particular definition is active.

Selected subexpression (of an <maction> element)
The argument of an <maction> element (a layout schema defined in Section 3.6) which 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 Section 3.6.

Spacelike (MathML expression)
A MathML expression which 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 3.2.6.

Standard Generalized Markup Language (SGML)
An ISO standard (ISO 8879:1986) which 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.

Subexpression (of a MathML expression "E")
A MathML expression contained (directly or indirectly) in E's content.

Suggested rendering rules for MathML presentation elements
Defined throughout Chapter 3; the ones which use other terms defined here occur mainly in Section 3.2.4, but also in 3.6 and perhaps elsewhere.

TeX
A software system written by Donald Knuth for typesetting documents.

Token element
Presentation token element or a Content token element. (See above.)

Toplevel element (of MathML)
<math> (defined in Chapter 7)

Typeface
A typeface is a specific design of a set of letters, numbers and symbols, such as "Times Roman" or "Chicago".

Well-Formed MathML data
MathML data which (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; (3) Satisfies the EBNF grammar for content elements.

Width
The distance from the left edge of the box to the right edge of the box.

Up: Table of Contents

Operator Dictionary

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.

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 subexpressions 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 math, such as TeX, which don't always specify how subexpressions 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 3.2.4.

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.

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 "Default value of **form** attribute" in Section 3.2.4.

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.

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, Chapter 6 should be taken as definitive, rather than this appendix.

Notes on lspace and rspace attributes

The values for **lspace** and **rspace** given here range from 0 to 6/18 em in units of 1/18 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>&ApplyFunction;</mo>`, the total spacing (**lspace** + **rspace**) in expressions such as "sin *x*" (where the right operand doesn't start with a fence) should be greater than 0; for `<mo>&InvisibleTimes;</mo>`, the total spacing should be greater than 0 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.

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"
"&LeftBracketingBar;"               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"
"&LeftDoubleBracketingBar;"         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"
```

```
"&RightBracketingBar;"              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"
"&RightDoubleBracketingBar;"        form="postfix" fence="true" stretchy="true"
lspace="0em" rspace="0em"
"&RightFloor;"                      form="postfix" fence="true" stretchy="true"
lspace="0em" rspace="0em"
"&LeftSkeleton;"                    form="prefix"  fence="true"  lspace="0em"
rspace="0em"
"&RightSkeleton;"                   form="postfix" fence="true"  lspace="0em"
rspace="0em"

"&InvisibleComma;"                  form="infix"   separator="true"  lspace="0em"
rspace="0em"

","                                 form="infix"   separator="true"  lspace="0em"
rspace=".33333em"

"&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=".27777em"
";"                                 form="postfix" separator="true"  lspace="0em"
rspace="0em"

":="                                form="infix"    lspace=".27777em"
rspace=".27777em"
"&Assign;"                          form="infix"    lspace=".27777em"
rspace=".27777em"

"&Because;"                         form="infix"    lspace=".27777em"
rspace=".27777em"
"&Therefore;"                       form="infix"    lspace=".27777em"
rspace=".27777em"

"&VerticalSeparator;"               form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"

"//"                                form="infix"    lspace=".27777em"
rspace=".27777em"

"&Colon;"                           form="infix"    lspace=".27777em"
rspace=".27777em"

"&amp;"                             form="prefix"   lspace="0em" rspace=".27777em"
"&amp;"                             form="postfix"  lspace=".27777em" rspace="0em"

"*="                                form="infix"    lspace=".27777em"
rspace=".27777em"
"-="                                form="infix"    lspace=".27777em"
rspace=".27777em"
```

```
"+="                                form="infix"    lspace=".27777em"
rspace=".27777em"
"/="                                form="infix"    lspace=".27777em"
rspace=".27777em"

"->"                                form="infix"    lspace=".27777em"
rspace=".27777em"

":"                                 form="infix"    lspace=".27777em"
rspace=".27777em"

".."                                form="postfix"  lspace=".22222em" rspace="0em"
"..."                               form="postfix"  lspace=".22222em" rspace="0em"

"&SuchThat;"                        form="infix"    lspace=".27777em"
rspace=".27777em"

"&DoubleLeftTee;"                   form="infix"    lspace=".27777em"
rspace=".27777em"
"&DoubleRightTee;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&DownTee;"                         form="infix"    lspace=".27777em"
rspace=".27777em"
"&LeftTee;"                         form="infix"    lspace=".27777em"
rspace=".27777em"
"&RightTee;"                        form="infix"    lspace=".27777em"
rspace=".27777em"

"&Implies;"                         form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&RoundImplies;"                    form="infix"    lspace=".27777em"
rspace=".27777em"

"|"                                 form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"||"                                form="infix"    lspace=".22222em"
rspace=".22222em"
"&Or;"                              form="infix"   stretchy="true"
lspace=".22222em" rspace=".22222em"

"&amp;&amp;"                        form="infix"    lspace=".27777em"
rspace=".27777em"
"&And;"                             form="infix"   stretchy="true"
lspace=".22222em" rspace=".22222em"

"&amp;"                             form="infix"    lspace=".27777em"
rspace=".27777em"

"!"                                 form="prefix"   lspace="0em" rspace=".27777em"
"&Not;"                             form="prefix"   lspace="0em" rspace=".27777em"

"&Exists;"                          form="prefix"   lspace="0em" rspace=".27777em"
"&ForAll;"                          form="prefix"   lspace="0em" rspace=".27777em"
"&NotExists;"                       form="prefix"   lspace="0em" rspace=".27777em"

"&Element;"                         form="infix"    lspace=".27777em"
rspace=".27777em"
```

```
"&NotElement;"                      form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotReverseElement;"               form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotSquareSubset;"                 form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotSquareSubsetEqual;"            form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotSquareSuperset;"               form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotSquareSupersetEqual;"          form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotSubset;"                       form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotSubsetEqual;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotSuperset;"                     form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotSupersetEqual;"                form="infix"    lspace=".27777em"
rspace=".27777em"
"&ReverseElement;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&SquareSubset;"                    form="infix"    lspace=".27777em"
rspace=".27777em"
"&SquareSubsetEqual;"               form="infix"    lspace=".27777em"
rspace=".27777em"
"&SquareSuperset;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&SquareSupersetEqual;"             form="infix"    lspace=".27777em"
rspace=".27777em"
"&Subset;"                          form="infix"    lspace=".27777em"
rspace=".27777em"
"&SubsetEqual;"                     form="infix"    lspace=".27777em"
rspace=".27777em"
"&Superset;"                        form="infix"    lspace=".27777em"
rspace=".27777em"
"&SupersetEqual;"                   form="infix"    lspace=".27777em"
rspace=".27777em"

"&DoubleLeftArrow;"                 form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&DoubleLeftRightArrow;"            form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&DoubleRightArrow;"                form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&DownLeftRightVector;"             form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&DownLeftTeeVector;"               form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&DownLeftVector;"                  form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&DownLeftVectorBar;"               form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&DownRightTeeVector;"              form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&DownRightVector;"                 form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
```

```
"&DownRightVectorBar;"              form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LeftArrow;"                       form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LeftArrowBar;"                    form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LeftArrowRightArrow;"             form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LeftRightArrow;"                  form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LeftRightVector;"                 form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LeftTeeArrow;"                    form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LeftTeeVector;"                   form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LeftVector;"                      form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LeftVectorBar;"                   form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LowerLeftArrow;"                  form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&LowerRightArrow;"                 form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&RightArrow;"                      form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&RightArrowBar;"                   form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&RightArrowLeftArrow;"             form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&RightTeeArrow;"                   form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&RightTeeVector;"                  form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&RightVector;"                     form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&RightVectorBar;"                  form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&ShortLeftArrow;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&ShortRightArrow;"                 form="infix"    lspace=".27777em"
rspace=".27777em"
"&UpperLeftArrow;"                  form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&UpperRightArrow;"                 form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"

"="                                 form="infix"    lspace=".27777em"
rspace=".27777em"
"&lt;"                              form="infix"    lspace=".27777em"
rspace=".27777em"
">"                                 form="infix"    lspace=".27777em"
rspace=".27777em"
"!="                                form="infix"    lspace=".27777em"
rspace=".27777em"
"=="                                form="infix"    lspace=".27777em"
rspace=".27777em"
```

```
"&lt;="                              form="infix"    lspace=".27777em"
rspace=".27777em"
">="                                form="infix"    lspace=".27777em"
rspace=".27777em"
"&Congruent;"                       form="infix"    lspace=".27777em"
rspace=".27777em"
"&CupCap;"                          form="infix"    lspace=".27777em"
rspace=".27777em"
"&DotEqual;"                        form="infix"    lspace=".27777em"
rspace=".27777em"
"&DoubleVerticalBar;"               form="infix"    lspace=".27777em"
rspace=".27777em"
"&Equal;"                           form="infix"    lspace=".27777em"
rspace=".27777em"
"&EqualTilde;"                      form="infix"    lspace=".27777em"
rspace=".27777em"
"&Equilibrium;"                     form="infix"   stretchy="true"
lspace=".27777em" rspace=".27777em"
"&GreaterEqual;"                    form="infix"    lspace=".27777em"
rspace=".27777em"
"&GreaterEqualLess;"                form="infix"    lspace=".27777em"
rspace=".27777em"
"&GreaterFullEqual;"                form="infix"    lspace=".27777em"
rspace=".27777em"
"&GreaterGreater;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&GreaterLess;"                     form="infix"    lspace=".27777em"
rspace=".27777em"
"&GreaterSlantEqual;"               form="infix"    lspace=".27777em"
rspace=".27777em"
"&GreaterTilde;"                    form="infix"    lspace=".27777em"
rspace=".27777em"
"&HumpDownHump;"                    form="infix"    lspace=".27777em"
rspace=".27777em"
"&HumpEqual;"                       form="infix"    lspace=".27777em"
rspace=".27777em"
"&LeftTriangle;"                    form="infix"    lspace=".27777em"
rspace=".27777em"
"&LeftTriangleBar;"                 form="infix"    lspace=".27777em"
rspace=".27777em"
"&LeftTriangleEqual;"               form="infix"    lspace=".27777em"
rspace=".27777em"
"&le;"                              form="infix"    lspace=".27777em"
rspace=".27777em"
"&LessEqualGreater;"                form="infix"    lspace=".27777em"
rspace=".27777em"
"&LessFullEqual;"                   form="infix"    lspace=".27777em"
rspace=".27777em"
"&LessGreater;"                     form="infix"    lspace=".27777em"
rspace=".27777em"
"&LessLess;"                        form="infix"    lspace=".27777em"
rspace=".27777em"
"&LessSlantEqual;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&LessTilde;"                       form="infix"    lspace=".27777em"
rspace=".27777em"
"&NestedGreaterGreater;"            form="infix"    lspace=".27777em"
```

```
rspace=".27777em"
"&NestedLessLess;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotCongruent;"                    form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotCupCap;"                       form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotDoubleVerticalBar;"            form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotEqual;"                        form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotEqualTilde;"                   form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotGreater;"                      form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotGreaterEqual;"                 form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotGreaterFullEqual;"             form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotGreaterGreater;"               form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotGreaterLess;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotGreaterSlantEqual;"            form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotGreaterTilde;"                 form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotHumpDownHump;"                 form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotHumpEqual;"                    form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLeftTriangle;"                 form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLeftTriangleBar;"              form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLeftTriangleEqual;"            form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLess;"                         form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLessEqual;"                    form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLessFullEqual;"                form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLessGreater;"                  form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLessLess;"                     form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLessSlantEqual;"               form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotLessTilde;"                    form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotNestedGreaterGreater;"         form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotNestedLessLess;"               form="infix"    lspace=".27777em"
rspace=".27777em"
"&NotPrecedes;"                     form="infix"    lspace=".27777em"
rspace=".27777em"
```

"⪯̸" form="infix" lspace=".27777em"
rspace=".27777em"
"⋠" form="infix" lspace=".27777em"
rspace=".27777em"
"&NotPrecedesTilde;" form="infix" lspace=".27777em"
rspace=".27777em"
"⋫" form="infix" lspace=".27777em"
rspace=".27777em"
"⧐̸" form="infix" lspace=".27777em"
rspace=".27777em"
"⋭" form="infix" lspace=".27777em"
rspace=".27777em"
"⊁" form="infix" lspace=".27777em"
rspace=".27777em"
"⪰̸" form="infix" lspace=".27777em"
rspace=".27777em"
"⋡" form="infix" lspace=".27777em"
rspace=".27777em"
"≿̸" form="infix" lspace=".27777em"
rspace=".27777em"
"≁" form="infix" lspace=".27777em"
rspace=".27777em"
"≄" form="infix" lspace=".27777em"
rspace=".27777em"
"≇" form="infix" lspace=".27777em"
rspace=".27777em"
"≉" form="infix" lspace=".27777em"
rspace=".27777em"
"∤" form="infix" lspace=".27777em"
rspace=".27777em"
"≺" form="infix" lspace=".27777em"
rspace=".27777em"
"⪯" form="infix" lspace=".27777em"
rspace=".27777em"
"≼" form="infix" lspace=".27777em"
rspace=".27777em"
"≾" form="infix" lspace=".27777em"
rspace=".27777em"
"∷" form="infix" lspace=".27777em"
rspace=".27777em"
"∝" form="infix" lspace=".27777em"
rspace=".27777em"
"⇋" form="infix" stretchy="true"
lspace=".27777em" rspace=".27777em"
"⊳" form="infix" lspace=".27777em"
rspace=".27777em"
"⧐" form="infix" lspace=".27777em"
rspace=".27777em"
"⊵" form="infix" lspace=".27777em"
rspace=".27777em"
"≻" form="infix" lspace=".27777em"
rspace=".27777em"
"⪰" form="infix" lspace=".27777em"
rspace=".27777em"
"≽" form="infix" lspace=".27777em"
rspace=".27777em"
"≿" form="infix" lspace=".27777em"

```
rspace=".27777em"
"&Tilde;"                          form="infix"    lspace=".27777em"
rspace=".27777em"
"&TildeEqual;"                     form="infix"    lspace=".27777em"
rspace=".27777em"
"&TildeFullEqual;"                 form="infix"    lspace=".27777em"
rspace=".27777em"
"&TildeTilde;"                     form="infix"    lspace=".27777em"
rspace=".27777em"
"&UpTee;"                          form="infix"    lspace=".27777em"
rspace=".27777em"
"&VerticalBar;"                    form="infix"    lspace=".27777em"
rspace=".27777em"

"&SquareUnion;"                    form="infix"   stretchy="true"
lspace=".22222em" rspace=".22222em"
"&Union;"                          form="infix"   stretchy="true"
lspace=".22222em" rspace=".22222em"
"&UnionPlus;"                      form="infix"   stretchy="true"
lspace=".22222em" rspace=".22222em"

"-"                                form="infix"    lspace=".22222em"
rspace=".22222em"
"+"                                form="infix"    lspace=".22222em"
rspace=".22222em"
"&Intersection;"                   form="infix"   stretchy="true"
lspace=".22222em" rspace=".22222em"
"&MinusPlus;"                      form="infix"    lspace=".22222em"
rspace=".22222em"
"&PlusMinus;"                      form="infix"    lspace=".22222em"
rspace=".22222em"
"&SquareIntersection;"             form="infix"   stretchy="true"
lspace=".22222em" rspace=".22222em"

"&Vee;"                            form="prefix"  largeop="true"
movablelimits="true" stretchy="true"  lspace="0em" rspace=".16666em"
"&CircleMinus;"                    form="prefix"  largeop="true"
movablelimits="true"  lspace="0em" rspace=".16666em"
"&CirclePlus;"                     form="prefix"  largeop="true"
movablelimits="true"  lspace="0em" rspace=".16666em"
"&Sum;"                            form="prefix"  largeop="true"
movablelimits="true" stretchy="true"  lspace="0em" rspace=".16666em"
"&Union;"                          form="prefix"  largeop="true"
movablelimits="true" stretchy="true"  lspace="0em" rspace=".16666em"
"&UnionPlus;"                      form="prefix"  largeop="true"
movablelimits="true" stretchy="true"  lspace="0em" rspace=".16666em"
"lim"                              form="prefix"  movablelimits="true"
lspace="0em" rspace=".16666em"
"max"                              form="prefix"  movablelimits="true"
lspace="0em" rspace=".16666em"
"min"                              form="prefix"  movablelimits="true"
lspace="0em" rspace=".16666em"

"&CircleMinus;"                    form="infix"    lspace=".16666em"
rspace=".16666em"
"&CirclePlus;"                     form="infix"    lspace=".16666em"
rspace=".16666em"
```

```
"&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=".16666em"
rspace=".16666em"

"&Cap;"                              form="infix"    lspace=".16666em"
rspace=".16666em"

"&VerticalTilde;"                    form="infix"    lspace=".16666em"
rspace=".16666em"

"&Wedge;"                            form="prefix"  largeop="true"
movablelimits="true" stretchy="true"  lspace="0em" rspace=".16666em"
"&CircleTimes;"                      form="prefix"  largeop="true"
movablelimits="true"  lspace="0em" rspace=".16666em"
"&Coproduct;"                        form="prefix"  largeop="true"
movablelimits="true" stretchy="true"  lspace="0em" rspace=".16666em"
"&Product;"                          form="prefix"  largeop="true"
movablelimits="true" stretchy="true"  lspace="0em" rspace=".16666em"
"&Intersection;"                     form="prefix"  largeop="true"
movablelimits="true" stretchy="true"  lspace="0em" rspace=".16666em"

"&Coproduct;"                        form="infix"    lspace=".16666em"
rspace=".16666em"

"&Star;"                             form="infix"    lspace=".16666em"
rspace=".16666em"

"&CircleDot;"                        form="prefix"  largeop="true"
movablelimits="true"  lspace="0em" rspace=".16666em"

"*"                                  form="infix"    lspace=".16666em"
rspace=".16666em"
"&InvisibleTimes;"                   form="infix"    lspace="0em" rspace="0em"

"&CenterDot;"                        form="infix"    lspace=".16666em"
rspace=".16666em"

"&CircleTimes;"                      form="infix"    lspace=".16666em"
rspace=".16666em"

"&Vee;"                              form="infix"    lspace=".16666em"
rspace=".16666em"

"&Wedge;"                            form="infix"    lspace=".16666em"
rspace=".16666em"
```

```
"&Diamond;"                          form="infix"    lspace=".16666em"
rspace=".16666em"

"&Backslash;"                        form="infix"   stretchy="true"
lspace=".16666em" rspace=".16666em"

"/"                                  form="infix"   stretchy="true"
lspace=".16666em" rspace=".16666em"

"-"                                  form="prefix"   lspace="0em" rspace=".05555em"
"+"                                  form="prefix"   lspace="0em" rspace=".05555em"
"&MinusPlus;"                        form="prefix"   lspace="0em" rspace=".05555em"
"&PlusMinus;"                        form="prefix"   lspace="0em" rspace=".05555em"

"."                                  form="infix"    lspace="0em" rspace="0em"

"&Cross;"                            form="infix"    lspace=".11111em"
rspace=".11111em"

"**"                                 form="infix"    lspace=".11111em"
rspace=".11111em"

"&CircleDot;"                        form="infix"    lspace=".11111em"
rspace=".11111em"

"&SmallCircle;"                      form="infix"    lspace=".11111em"
rspace=".11111em"

"&Square;"                           form="prefix"   lspace="0em" rspace=".11111em"

"&Del;"                              form="prefix"   lspace="0em" rspace=".11111em"
"&PartialD;"                         form="prefix"   lspace="0em" rspace=".11111em"

"&CapitalDifferentialD;"             form="prefix"   lspace="0em" rspace=".11111em"
"&DifferentialD;"                    form="prefix"   lspace="0em" rspace=".11111em"

"&Sqrt;"                             form="prefix"  stretchy="true"  lspace="0em"
rspace=".11111em"

"&DoubleDownArrow;"                  form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&DoubleLongLeftArrow;"              form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&DoubleLongLeftRightArrow;"         form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&DoubleLongRightArrow;"             form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&DoubleUpArrow;"                    form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&DoubleUpDownArrow;"                form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&DownArrow;"                        form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&DownArrowBar;"                     form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&DownArrowUpArrow;"                 form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
```

```
"&DownTeeArrow;"                    form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LeftDownTeeVector;"               form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LeftDownVector;"                  form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LeftDownVectorBar;"               form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LeftUpDownVector;"                form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LeftUpTeeVector;"                 form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LeftUpVector;"                    form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LeftUpVectorBar;"                 form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LongLeftArrow;"                   form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LongLeftRightArrow;"              form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&LongRightArrow;"                  form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&ReverseUpEquilibrium;"            form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&RightDownTeeVector;"              form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&RightDownVector;"                 form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&RightDownVectorBar;"              form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&RightUpDownVector;"               form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&RightUpTeeVector;"                form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&RightUpVector;"                   form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&RightUpVectorBar;"                form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&ShortDownArrow;"                  form="infix"    lspace=".11111em"
rspace=".11111em"
"&ShortUpArrow;"                    form="infix"    lspace=".11111em"
rspace=".11111em"
"&UpArrow;"                         form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&UpArrowBar;"                      form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&UpArrowDownArrow;"                form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&UpDownArrow;"                     form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&UpEquilibrium;"                   form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"
"&UpTeeArrow;"                      form="infix"   stretchy="true"
lspace=".11111em" rspace=".11111em"

"^"                                 form="infix"    lspace=".11111em"
rspace=".11111em"
```

```
"&lt;>"                          form="infix"    lspace=".11111em"
rspace=".11111em"

"'"                             form="postfix"  lspace=".11111em" rspace="0em"

"!"                             form="postfix"  lspace=".11111em" rspace="0em"
"!!"                            form="postfix"  lspace=".11111em" rspace="0em"

"~"                             form="infix"    lspace=".11111em"
rspace=".11111em"

"@"                             form="infix"    lspace=".11111em"
rspace=".11111em"

"--"                            form="postfix"  lspace=".11111em" rspace="0em"
"--"                            form="prefix"   lspace="0em" rspace=".11111em"
"++"                            form="postfix"  lspace=".11111em" rspace="0em"
"++"                            form="prefix"   lspace="0em" rspace=".11111em"

"&ApplyFunction;"               form="infix"    lspace="0em" rspace="0em"

"?"                             form="infix"    lspace=".11111em"
rspace=".11111em"

"_"                             form="infix"    lspace=".11111em"
rspace=".11111em"

"&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"
"&DiacriticalLeftArrow;"        form="postfix" accent="true" stretchy="true"
lspace="0em" rspace="0em"
"&DiacriticalLeftRightArrow;"   form="postfix" accent="true" stretchy="true"
lspace="0em" rspace="0em"
"&DiacriticalLeftRightVector;"  form="postfix" accent="true" stretchy="true"
lspace="0em" rspace="0em"
"&DiacriticalLeftVector;"       form="postfix" accent="true" stretchy="true"
lspace="0em" rspace="0em"
"&DiacriticalAcute;"            form="postfix" accent="true"  lspace="0em"
rspace="0em"
"&DiacriticalRightArrow;"       form="postfix" accent="true" stretchy="true"
lspace="0em" rspace="0em"
"&DiacriticalRightVector;"      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"
```

Up: Table of Contents

Working Group Membership

The W3C Math working group is presently co-chaired by Patrick Ion of the AMS, and Angel Diaz of IBM. Contact the co-chairs if you are interested in joining the group. For the present membership see its working group home page.

At the time of release of the MathML 1.0 the working group was co-chaired by 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, in addition to people listed in the original membership below, corrections have been offered by David Carlisle, Don Gignac, Kostya Serebriany, Ben Hinkle, Sebastian Rahtz, Sam Dooley and others.

Members of the working group that proposed MathML 1.0 were:

Adobe

T. V. Raman

American Mathematical Society , Providence RI

Ralph Youngen

Design Science Inc.

Paul Topping

Elsevier Science

Nico Poppelier

Geometry Technologies, Inc.

Robert Miner

IBM Research Division, Yorktown Heights, NY

Robert S. Sutor, Angel Diaz

Mathematical Reviews (American Mathematical Society), Ann Arbor MI

Patrick Ion

Mathsoft, Cambridge, MA

Stephen Glim

University of Waterloo

Ka-Ping Yee

INRIA, Sophia Antipolis

Stéphane Dalmas

Stilo Technologies

Stephen Buswell

SoftQuad, Surrey, BC

Lauren Wood

Texterity Corporation

Ronald Whitney

University of Western Ontario

Stephen Watt

Waterloo Maple Inc.

Stan Devitt

W3C

Dave Raggett, Arnaud Le Hors

Wolfram Research

Steven Hunt, Brenda Hunt, Bruce Smith, Neil Soiffer

Up: Table of Contents

Content Markup Validation Grammar

```
Informal EBNF grammar for Content Markup structure validation
=============================================================

// Notes
//
// This defines the valid expression trees in content markup
//
// ** it does not define attribute validation -
// ** this has to be done on top
//
// Presentation_tags is a placeholder for a valid
// presentation element start tag or end tag
//
// #PCDATA is the XML parsed character data
//
// symbols beginning with '_' eg. _mmlarg are internal symbols
// (recursive grammar usually required for recognition)
//
// all-lowercase symbols eg. 'ci' are terminal symbols
// representing MathML content elements
//
// symbols beginning with Uppercase are terminals
// representating other tokens
//
// revised sb 3.nov.97, 16.nov.97 and 22.dec.1997
// revised sb 6.jan.98, 6.Feb.1998 and 4.april.1998


// whitespace  definitions  including presentation_tags

Presentation_tags ::= "presentation"        //placeholder


Space    ::= #x09 | #xoA | #xoD | #x20      //tab, lf, cr, space characters
S        ::= (Space | Presentation_tags)*   //treat presentation as space

// only for content validation
// characters

Char     ::= Space | [#x21 - #xFFFD]
             | [#x00010000 - #x7FFFFFFF]   //valid XML chars

// start and end tag functions
// start(%x) returns a valid start tag for the element %x
// end(%x) returns a valid end tag for the element %x
// empty(%x) returns a valid empty tag for the element %x
```

```
//
// start(ci)    ::= "<ci>"
// end(cn)              ::= "</cn>"
// empty(plus)  ::= "<plus/>"
//
// The reason for doing this is to avoid writing a grammar
// for all the attributes. The model below is not complete
// for all possible attribute values.

_start(%x)      ::= "<%x" (Char - '>')* ">"
// returns a valid start tag for the element %x

_end(%x)        ::= "<%x" Space* ">"
// returns a valid end tag for the element %x

_empty(%x)      ::= "<%x" (Char - '>')* "/>"
// returns a valid empty tag for the element %x

_sg(%x)         ::= S _start(%x)
// start tag preceded by optional whitespace

_eg(%x)         ::= _end(%x) S

// end tag followed by optional whitespace

_ey(%x)         ::= S _empty(%x) S
// empty tag preceded and followed by optional whitespace

// mathml content constructs
// allow declare within generic argument type so we can insert it anywhere

_mmlall         ::= _container | _relation | _operator | _qualifier | _other
_mmlarg         ::= declare* _container declare*

_container      ::= _token | _special | _constructor
_token          ::= ci | cn
_special        ::= apply | lambda | reln
_constructor    ::= interval | list | matrix | matrixrow | set | vector

_other          ::= condition |  declare | sep

_qualifier      ::= lowlimit | uplimit | bvar | degree | logbase

// relations

_relation       ::= _genrel | _setrel | _seqrel2ary
_genrel         ::= _genrel2ary | _genrelnary
_genrel2ary     ::= ne
_genrelnary     ::= eq | leq | lt | geq | gt

_setrel         ::=  _seqrel2ary | _setrelnary
_setrel2ary     ::=  in | notin | notsubset | notprsubset
_setrelnary     ::= subset | prsubset

_seqrel2ary     ::= tendsto
```

```
//operators

_operator       ::= _funcop | _sepop | _arithop | _calcop
                | _seqop | _trigop | _statop | _lalgop
                | _logicop | _setop

_funcop         ::= _funcop1ary | _funcopnary
_funcop1ary     ::= inverse     | ident
_funcopnary     ::= fn| compose   // general user-defined function is n-ary


// arithmetic operators
// (note minus is both 1ary and 2ary)

_arithop        ::= _arithop1ary | _arithop2ary | _arithopnary  | root
_arithop1ary    ::= abs | conjugate | exp | factorial | minus
_arithop2ary    ::= quotient | divide | minus | power | rem
_arithopnary    ::= plus | times | max | min | gcd

// calculus

_calcop         ::= _calcop1ary | log | int | diff | partialdiff
_calcop1ary     ::= ln

// sequences and series

_seqop          ::= sum | product | limit

// trigonometry

_trigop         ::= sin | cos | tan | sec | csc | cot | sinh
                    | cosh | tanh | sech | csch | coth
                    | arcsin | arccos | arctan

// statistics operators

_statop         ::= _statopnary | moment
_statopnary     ::= mean | sdev | variance | median | mode

// linear algebra operators

_lalgop         ::= _lalgop1ary | _lalgopnary
_lalgop1ary     ::= determinant | transpose
_lalgopnary     ::= selector

// logical operators

_logicop        ::= _logicop1ary | _logicopnary | _logicop2ary | _logicopquant
_logicop1ary    ::= not
_logicop2ary    ::= implies
_logicopnary    ::= and | or | xor
_logicopquant   ::= forall | exists

// set theoretic operators
```

```
_setop          ::= _setop2ary | _setopnary
_setop2ary      ::= setdiff
_setopnary      ::= union | intersect

// operator groups

_unaryop        ::=  _func1ary | _arithop1ary | _trigop | _lalgop1ary
                     | _calcop1ary | _logicop1ary

_binaryop       ::=  _arithop2ary | _setop2ary | _logicop2ary
_naryop         ::=  _arithopnary | _statopnary  | _logicopnary
                     | _lalgopnary | _setopnary | _funcopnary

_ispop          ::= int | sum | product
_diffop         ::= diff | partialdiff

_binaryrel      ::= _genrel2ary | _setrel2ary |  _seqrel2ary
_naryrel        ::= _genrelnary | _setrelnary

//separator

sep             ::=  _ey(sep)

// leaf tokens  and data content of leaf elements
// note _mdata includes Presentation constructs here.

_mdatai         ::= (#PCDATA | Presentation_tags)*
_mdatan         ::= (#PCDATA | sep | Presentation_tags)*

ci              ::=  _sg(ci) _mdatai _eg(ci)
cn              ::=  _sg(cn) _mdatan _eg(cn)

// condition -  constraints constraints. contains either
// a single reln (relation), or
// an apply holding a logical combination of relations, or
// a set (over which the operator should be applied)

condition       ::= _sg(condition)  reln | apply | set  _eg(condition)

// domains for integral, sum , product

_ispdomain      ::= (lowlimit  uplimit?)
                | uplimit
                | interval
                | condition

// apply construct

apply           ::= _sg(apply) _applybody _eg(apply)

_applybody      ::= ( _unaryop _mmlarg  )
//1-ary ops
                                | (_binaryop _mmlarg _mmlarg)
//2-ary ops
```

```
                                | (_naryop _mmlarg*)
//n-ary ops, enumerated arguments
                                | (_naryop bvar* condition _mmlarg)
//n-ary ops, condition defines argument list
                                | (_ispop  bvar? _ispdomain? _mmlarg)
//integral, sum, product
                                | (_diffop bvar* _mmlarg)
//differential ops
                                | (log logbase? _mmlarg)
//logs
                                | (moment degree? _mmlarg*)
//statistical moment
                                | (root degree? _mmlarg)
//radicals - default is square-root
                                | (limit bvar* lowlimit? condition? _mmlarg)
//limits
                                | (_logicopquant bvar+ condition? (reln | apply))
//quantifier with explicit bound variables


// equations and relations - reln uses lisp-like syntax (like apply)
// the bvar and condition are used to construct a "such that" or
// "where" constraint on the relation

reln            ::= _sg(reln) _relnbody _eg(reln)

_relnbody       ::= ( _binaryrel bvar* condition? _mmlarg _mmlarg )
                    | ( _naryrel bvar* condition? _mmlarg*  )

// fn construct

fn                      ::= _sg(fn) _fnbody _eg(fn)
_fnbody         ::=  Presentation_tags | container

// lambda construct     - note at least 1 bvar must be present

lambda          ::=  _sg(lambda) _lambdabody _eg(lambda)

_lambdabody     ::=  bvar+ _container  //multivariate lambda calculus

//declare construct

declare         ::= _sg(declare) _declarebody _eg(declare)
_declarebody    ::= ci (fn | constructor)?

// constructors

interval        ::=  _sg(interval) _mmlarg _mmlarg _eg(interval)
//start, end define interval

set         ::=  _sg(set) _lsbody _eg(set)
list        ::=  _sg(list) _lsbody _eg(list)

_lsbody         ::=  _mmlarg*                   //enumerated arguments
                 | (bvar* condition _mmlarg)   //condition constructs arguments
```

```
matrix          ::= _sg(matrix) matrixrow* _eg(matrix)

matrixrow       ::= _sg(matrixrow) _mmlall* _eg(matrixrow)
//allows matrix of operators

vector          ::= _sg(vector) _mmlarg* _eg(vector)

//qualifiers - note the contained _mmlarg could be a reln

lowlimit        ::= _sg(lowlimit) _mmlarg  _eg(lowlimit)
uplimit         ::= _sg(uplimit) _mmlarg _eg(uplimit)
bvar            ::= _sg(bvar) ci degree? _eg(bvar)
degree          ::= _sg(degree) _mmlarg _eg(degree)
logbase         ::= _sg(logbase) _mmlarg _eg(logbase)

//relations and operators
// (one declaration for each operator and relation element)

_relation       ::= _ey(%relation)         //eg. <eq/>  <lt/>
_operator       ::= _ey(%operator)         //eg. <exp/> <times/>

//the top level math element

math            ::= _sg(math) mmlall* _eg(math)
```

Up: Table of Contents

F. Content Element Definitions

F.1. About Content Markup Elements

Every content element must have a mathematical definition associated with it in some form. The purpose of this appendix is to provide ***default*** definitions. (An index to the definitions is provided later in this document.) For this release of MathML definitions have not been restricted to any one format. There are several reasons for allowing flexibility at this time.

1. Many mathematical constructs are not yet implemented in any computation based system. However, MathML must still allow authors to associate mathematical constructs with definitions for archival purposes and so that work on such implementations can begin.
2. The task of defining a mathematical object, and establishing an association with a particular definition does not logically depend on the existence of an implementation in a computational system. It is a perfectly legitimate mathematical activity independent of whether it is ever implemented. Providing a record of those author specified associations is integral part of the role of MathML.
3. The task of designing a machine readable language suitable for recording semantic descriptions is an onerous one that goes substantially beyond the scope of this particular recommendation. It also overlaps substantially with efforts groups such as the OpenMath Consortium. (See also: North American OpenMath Initiative, and The European OpenMath Consortium)

The feasibilty of implementing a particular object in a particular comptational system and the details of particular implementations have very little to do with the requirement that there actually be a mathematical definition. An author's decision to use content elements is a decision to work with defined objects. The definitions may be as vague as claiming that, say F, is an unknown, but differentiable function from the real numbers to the real numbers, or as complicated as requiring that F to be an elaborate new function or operation as defined in some recent research paper. The primary role of MathML content elements is to provide a mechanism for recording the fact that a particular structure has a particular mathematical meaning. If a definition is implemented in a computational system, this is a bonus.

Of course, default definitions and semantics should be chosen to be as useful as possible. Where possible they should be already implemented or easy to implement and all other things being equal, an author would be well advised to use a definition that is in common use. This is no different from noting that most well written mathematical communications (in any format) benefit substantially from the author's use of widely used and understood terms.

A requirement that there be a definition is also very different from a requirement that a definition be provided in some specific manner. Before requiring a particular approach to definitions one needs to consider such issues as:

1. providing a language for defining semantics.
2. deciding if it is reasonable to *require* the use of such a syntax. (Authors may not have the time or expertise to provide a formal description in a new and unfamiliar language.)
3. not being constrained by the limitations of existing computational systems.

In order to leave open the discussion of such fundamental issues we have deliberately limited the support for new or author defined definitions to support for specifying an appropriate "definitionURL.". The format of the content of that URL is unspecified. It might be the URL of a mathematical paper whose whole purpose is to define a new operator, or even a simple reference to a traditional text. If the author's mathematical operator matches exactly with an operator in a particular computational system, an appropriate definition might be a MathML **semantics** element establishing a correspondence between two encodings. Whatever is chosen, the only essential feature is that the definition be provided.

This rest of this appendix provides detailed descriptions of the default semantics associated with each of the MathML content elements. Since this is exactly the role intended for the encodings under development by the OpenMath Consortium and one of our goals is to foster international cooperation in suchstandardization efforts we have presented the default definitions in a format modeled on OpenMath's *content dictionaries* . While the actual details differ somewhat from the OpenMath specification, the underlying principles are the same and this is being used as input to ongoing discussions underway with the OpenMath Consortium aimed at ensuring that the OpenMath encodings will be capable of conveying the necessary information.

F.1.1. The Structure of an MMLdefinition.

Each MathML element is described using an XML format. The top element is **MMLdefinition**. The sub-elements identify the various parts of the description and include:

name

PCDATA providing the name of the MathML element.

description

A text based description of the object that an element represents. Thiscross will often include cross references to more traditional texts or papers or existing papers on the Web.

functorclass

Each MathML element must be classified according to its mathematical role.

punctuation -Some elements exist simply as an aid to parsing. For example the sep element is used to separate the CDATA defining a rational number into two parts in a manner that is easily parsed by an XML application. These objects are refered to as **punctuation**.

modifier -Some elements exist simply to modify the properties of an existing element or mathematical object. For example the declare construct is used to reset the default attribute values, or to associate a name with a specific instance of an object. These kinds of elements are referred to as **modifiers** and the result is of the same type, but with different attributes.

constructor - The remaining objects which "contain" sub-elements are all object *constructors* of some sort or another. They combine the sub-elements into a compound mathematical object such as a constant, set, list, or an expression representing a function application. For example, the **lambda** element is actually a constructor which *constructs* a function definition from a list of variables and an expression, while the **fn** element is a constructor that, in effect, sets the type of an object to function and if necessary, provides an external definition. Any use of apply produces an object of type *apply* whose sub-type is determined by the first operand and its properties. The signature of a constructor indicates the type of its sub-elements and the type (and sometimes subtype) of the resulting object.

function (operator) - The MathML objects represented by empty XML elements are functions or operators. These *function* definitions are parameterized by their XML attribute values and are used as the first argument to an **apply** or **reln**. Functions are classified according to how they are used.. For example the empty **<sin/>** element represents the **unary** mathematical function sine. In every case, element attributes may be used to further qualify the object. The **<plus/>** element is an **nary** operator. The result of using a function or operator is an expression which represents an object in a certain algebraic domain.

parameter -Another class of objects are the named *parameters*. For example, these named objects are used to identify bounds of integration, or differentiation variables.

MMLattribute

Some of the XML attributes of a MathML content element have a direct impact on the mathematical semantics of the object. For example the **type** attribute of the **cn** element is used to determine what type of constant (integer, real, etc.) is being constructed. Only those attributes that affect the mathematical properties of an object are listed here and typically they also appear explicitly in the signature.

signature

The signature is systematic representation which associates the different possible combinations of attributes and function arguments to the different kinds of mathematical objects that are constructed. The possible combinations of parameter and argument types (the left-hand side) each result in an object of some type (the right-hand side). It in effect describes how to resolve operator overloading.

For constructors (including parameters), the left-hand side of the signature describes the types of the child elements and the right-hand side describes the type of object that is constructed. For functions, the left-hand side of the signature indicates the types of the parameters and arguments that would be expected when it is applied, or used to construct a relation, and the right-hand side represents the mathematical type of the object constructed by the <apply>. Modifiers modify the the attributes of an existing object.. For example a ***symbol*** might become a ***symbol of type vector***.

The signature must be able to record specific attribute values and argument types on the left, and and parameterized types on the right.. The syntax used for signatures is of the general form:

```
[<attribute name>=<attributevalue>]( <list of argument types> )
--> <mathematical result type>(<mathematical subtype>).
```

The MMLattributes, if any, appear in the form **<attribute name>** = **<attribute value>**. They are separated notationally from the rest of the arguments by square braces. The possible values are usually taken from an enumerated list, and the signature is usually affected by selection of a specific value.

For the actual function arguments and named parameters on the left, the focus is on the mathematical types involved. The function argument types are presented in a syntax similar to that used for a DTD, with the one main exception.. The types of the named parameters appear in the signature as **<elementname>=<type>** in a manner analogous for that used for attribute values. For example, if the argument is named (e.g., bvar) then it is represented in the signature by an equation as in:

```
[<attribute name>=<attributevalue>]( bvar=symbol,<argument list> ) -->
<mathematical result type>(<mathematical subtype>)
```

No mathematical evaluation ever takes place in MathML. Every MathML content element either refers to a defined object such as a mathematical function or it combines such objects in some way to build a new object. For purposes of the signature, the constructed object represents an object of a certain type parameterized type. For example the result of applying <plus/> to arguments is an expression which respresents a sum. The type of the resulting expression depends on the types of the operands, and the values of the MathML attributes.

example

Examples of the use of this object in MathML and possibly other syntax are included in these elements.

property

This element describes the mathematical properties of such objects.. For simple associations of values with specific instances of an object, the first child is an instance of the object being defined. The second is a **value** or **approx** (approximation) element which contains a MathML description of this particular value. More elaborate conditions on the object are expressed using the MathML syntax.

MathML Dictionary: Version 1.0
February 10, 1998

- **F. Content Element Definitions**
  - **F.1. About Content Markup Elements**
    - F.1.1. The Structure of an MMLdefinition.
  - **F.2. Definitions of MathML Content Elements**
    - F.2.1. Leaf Elements
      - F.2.1.1. <cn>
      - F.2.1.2. <ci>
    - F.2.2. Basic Content Element
      - F.2.2.1. <apply>
      - F.2.2.2. <reln>
      - F.2.2.3. <fn>
      - F.2.2.4. <interval>
      - F.2.2.5. <inverse>
      - F.2.2.6. <sep>
      - F.2.2.7. <condition>
      - F.2.2.8. <declare>
      - F.2.2.9. <lambda>
      - F.2.2.10. <compose/>
      - F.2.2.11. <ident/>
    - F.2.3. Arithmetic, Algebra and Logic
      - F.2.3.1. <quotient/>
      - F.2.3.2. <exp/>
      - F.2.3.3. <factorial/>
      - F.2.3.4. <divide/>
      - F.2.3.5. <max/>

- F.2.3.6. <min/>
- F.2.3.7. <minus/>
- F.2.3.8. <plus/>
- F.2.3.9. <power/>
- F.2.3.10. <rem/>
- F.2.3.11. <times/>
- F.2.3.12. <root/>
- F.2.3.13. <gcd/>
- F.2.3.14. 13<and/>
- F.2.3.15. <or/>
- F.2.3.16. <xor/>
- F.2.3.17. <not/>
- F.2.3.18. <implies/>
- F.2.3.19. <forall/>
- F.2.3.20. <exists/>
- F.2.3.21. <abs/>
- F.2.3.22. <conjugate/>

- F.2.4. Relations
  - F.2.4.1. <eq/>
  - F.2.4.2. 2<neq/>
  - F.2.4.3. 3<gt/>
  - F.2.4.4. 4<lt/>
  - F.2.4.5. 5<geq/>
  - F.2.4.6. 6<leq/>
- F.2.5. Calculus
  - F.2.5.1. <ln/>
  - F.2.5.2. <log/>
  - F.2.5.3. <int/>
  - F.2.5.4. <diff/>
  - F.2.5.5. <partialdiff/>
  - F.2.5.6. <lowlimit/>
  - F.2.5.7. <uplimit/>
  - F.2.5.8. <bvar/>
  - F.2.5.9. <degree/>
- F.2.6. Theory of Sets
  - F.2.6.1. <set>
  - F.2.6.2. <list>
  - F.2.6.3. <union/>
  - F.2.6.4. <intersect/>
  - F.2.6.5. <in/>
  - F.2.6.6. <notin/>
  - F.2.6.7. <subset/>
  - F.2.6.8. <prsubset/>
  - F.2.6.9. <notsubset/>

- F.2.6.10. <notprsubset/>
- F.2.6.11. <setdiff/>
- F.2.7. Sequences and Series
  - F.2.7.1. <sum/>
  - F.2.7.2. <product/>
  - F.2.7.3. <limit/>
  - F.2.7.4. <tendsto/>
- F.2.8. Trigonometry
  - F.2.8.1. <sin/>
  - F.2.8.2. <cos/>
  - F.2.8.3. <tan/>
  - F.2.8.4. <sec/>
  - F.2.8.5. <csc/>
  - F.2.8.6. <cot/>
  - F.2.8.7. <sinh/>
  - F.2.8.8. <cosh/>
  - F.2.8.9. <tanh/>
  - F.2.8.10. <sech/>
  - F.2.8.11. <csch/>
  - F.2.8.12. <coth/>
  - F.2.8.13. <arcsin/>
  - F.2.8.14. <arccos/>
  - F.2.8.15. <arctan/>
- F.2.9. Statistics
  - F.2.9.1. <mean/>
  - F.2.9.2. <sdev/>
  - F.2.9.3. <variance/>
  - F.2.9.4. <median/>
  - F.2.9.5. <mode/>
  - F.2.9.6. <moment/>
- F.2.10. Lineary Algebra
  - F.2.10.1. <vector>
  - F.2.10.2. <matrix>
  - F.2.10.3. <matrixrow>
  - F.2.10.4. <determinant/>
  - F.2.10.5. <transpose/>
  - F.2.10.6. <selector/>

F.2. Definitions of MathML Content Elements

F.2.1. Leaf Elements

F.2.1.1. <cn>

```
<MMLdefinition>
```

```
<name> cn </name>
<description>
    A numerical constant.  The mathematical type of number
    is given as an attribute.  The default type is "real".
    Numbers such as rational, complex or real, require two
    parts for a complete specification.  The parts of such
    a number are separated by an empty "sep" element.

    There are a number of pre-defined constants including:
      &pi; &Exponential; &ComplexI &true; &false; &NaN;
    the properties of some of which are outlined below.

     The &NaN; is IEEE's "Not a Number", as defined in
     IEEE 854 standard for Floating point arithmetic.
</description>
<functorclass> constant </functorclass>
<MMLattribute>
    <name> type </name>
    <value> integer | rational | complex-cartesian
             | complex-polar | real
    </value>
    <default> real </default>
</MMLattribute>
<MMLattribute>
    <name> base </name>
    <value> positive_integer </value>
    <default> 10 </default>
</MMLattribute>
<signature> [type=integer](numstring) -> constant(integer) </signature>
<signature> [base=basevalue](numstring) -> constant(integer) </signature>
<signature> [type=rational](numstring,numstring) -> constant(rational) </signature>
<signature> [type=complex-cartesian](numstring,numstring) -> constant(complex)
</signature>
<signature> [type=rational](numstring,numstring) -> constant(rational) </signature>
<signature> [type=real](&pi;) -> constant(real) </signature>
<signature> [definition](numstring,numstring) -> constant(userdefined) </signature>
<signature> (&gamma;) -> constant</signature>
<example> <cn> 245 </cn> </example>
<example> <cn type="integer"> 245 </cn> </example>
<example> <cn type="integer" base="16"> A </cn></example>
<example> <cn type="rational"> 245 <sep> 351 </cn> </example>
<example> <cn type="complex-cartesian"> 1 <sep/> 2 </cn> </example>
<example> <cn> 245 </cn> </example>

<property> <approx>
  <cn> &pi; </cn>
  <cn> 3.141592654 </cn>
</approx></property>

<property> <approx>
  <cn> &gamma; </cn>
  <cn> .5772156649 </cn>
</approx> </property>

<property> <reln><identity/>
  <cn>&ImaginaryI; </cn>
  <apply><root><cn>-1</cn><cn>2</cn></apply>
</reln>
</property>
```

```
<property> <reln><approx>
<cn> &ExponentialE; </cn><cn>2.718281828 </cn>
</reln> </property>
<property> <apply><forall/>
  <bvar><ci type=boolean>p</ci></bvar>
 apply><and/>
    <ci>p</ci><cn>&true;</cn></apply>
    <ci>p</ci>
  </apply>
</property>
<property> <apply><forall/>
  <bvar><ci type=boolean>p</ci></bvar>
  <apply><or/>
    <ci>p</ci><cn>&true;</cn></apply>
    <cn>&true;</cn>
  </apply>
</property>

  <bvar><ci type=boolean>p</ci></bvar>
  <apply><or/>
    <ci>p</ci><cn>&true;</cn></apply>
    <cn>&true;</cn>
  </apply>
</property>

<property>
    <identity>
      <apply><not/><cn> &true; </apply>
      <cn> &false; </cn>
    </identity>
</property>

<property> <reln><identity/>
  <cn base="16"> A </cn> <cn> 10 </cn> </reln> </property>
<property> <reln><identity/>
  <cn base="16"> B </cn> <cn> 11 </cn> </reln></property>
<property> <reln><identity/>

  <cn base="16"> C </cn> <cn> 12 </cn> </reln></property>
<property> <reln><identity/>
  <cn base="16"> D </cn> <cn> 13 </cn> </reln></property>
<property> <reln><identity/>
  <cn base="16"> E </cn> <cn> 14 </cn> </reln></property>
<property> <reln><identity/>
  <cn base="16"> F </cn> <cn> 15 </cn> </reln></property>
</MMLdefinition>
```

F.2.1.2. <ci>

```
<MMLdefinition>
<name> ci </name>
<description>
  A symbolic name constructor. The type attribute can
  be set to any valid MathML type.
</description>
<functorclass> constructor , unary </functorclass>
<MMLattribute>
  <name> type </name>
  <value> constant | matrix | set | vector | list | MathMLtype </value>
  <default> real </default>
</MMLattribute>
```

```
<signature> ({string|mmlpresentation}) -> symbol(constant) </signature>
<signature> [type=MathMLType]({string|mmlpresentation}) -> symbol(MathMLType)
</signature>
<example><ci> xyz </ci> </example>
<example><ci> type="vector"> V </ci> </example>
</MMLdefinition>
```

F.2.2. Basic Content Element

F.2.2.1. <apply>

```
<MMLdefinition>
<name> apply </name>
<description>
  This is the MathML constructor for function application.
  The first argument is applied to the remaining arguments.
  It may be the case that some of the child elements are
  named elements. (See the signature.)
</description>
<functorclass> constructor , nary </functorclass>
<signature> (function,anything*) -> application </signature>
<example><apply><plus/><ci>x</ci><cn>1</cn></apply></example>
<example><apply><sin/><ci>x</ci></apply></example>
</MMLdefinition>
```

F.2.2.2. <reln>

```
<MMLdefinition>
<name> reln </name>
<description>
  This is the MathML constructor for expressing a relation between
  two or more mathematical objects.  The first argument indicates
  the type of "relation" between the remaining arguments. (See the signature.)
  No assumptions are made about the truth value of such a relation.
  Typically, the relation is used as a component in the construction
  of some logical assertion.   Relations may be combined into
         sets, etc. just like any other mathematical object.

</description>
<functorclass> constructor </functorclass>
<signature> (function,anything*) -> reln </signature>
<example><reln><and/><ci>P</ci><ci>Q</ci></reln></example>
<example><reln><lt/><ci>x</ci><ci>y</ci></reln></example>
</MMLdefinition>
```

F.2.2.3. <fn>

```
<MMLdefinition>
<name> fn </name>
<description>
  This is the MathML constructor for building new function
  names.   The "name" can be a general MathML content element.
  It identifies that object as "usable" in a function
  context.

  By setting its definitionURL value, you can
         associate it with a particular function definition.

  Use the MathML Declare to associate a name with a lambda
  construct.
</description>
```

```
<MMLattribute>
  <name>definitionURL</name>
  <value> URL </value>
  <default> none </default>
</MMLattribute>
<functorclass> constructor </functorclass>
<signature> (anything) -> function </signature>
<signature> [definitionURL=functiondef](anything) ->
  function(definitionURL=functiondef)
</signature>
<example><fn><ci>F</ci></fn></example>
<example><fn definitionURL="http://www.w3c/...">
  <lt/><ci>G</ci></fn>
</example>

<!--Declaring Id to be the identity function.-->

<example>
  <declare><fn><ci>Id</ci></fn><lambda><ci>x</ci><ci>x</ci></declare>
</example>
</MMLdefinition>
```

F.2.2.4. <interval>

```
<MMLdefinition>
<name> interval </name>
<description>
  This is the MathML constructor element for building an interval
  on the real line.  While an interval could be expressed by
  combining relations appropriately, they occur explicitly because
  of their frequence of occurrence in common use.
</description>
<MMLattribute>
  <name>type</name>
  <value> closed | open | open-closed | closed-open </value>
  <default> closed </default>
</MMLattribute>
<functorclass> constructor , binary </functorclass>
<signature> [type=intervaltype](expression,expression) -> interval </signature>
<example><reln><and/><ci>x</ci><cn>1</cn></reln></example>
<example><reln><lt/><ci>x</ci></reln></example>
</MMLdefinition>
```

F.2.2.5. <inverse>

```
<MMLdefinition>
<name> inverse </name>
<description>
  This MathML element is applied to a function in order to
  construct a new function that is to be interpreted as the
  inverse function of the original function.  For a particular
  function F,  inverse(F) composed with F behaves like the
  identity map on the domain of F and F composed with inverse(F)
  should be an identity function on a suitably restricted
  subset of the Range of F.

  The MathML definitionURL attribute should be used to resolve
         notational ambiguities, or to restrict the inverse to a
  particular domain or make it one-sided.
</description>
<MMLattribute>
```

```
  <name>definitionURL</name>
  <value> CDATA </value>
  <default> none </default>

<!--none corresponds to using the default MathML definition ...-->

</MMLattribute>
<functorclass> operator, unary </functorclass>
<signature> (function) -> function </signature>
<signature> [definitionURL=URL](function) ->
    function(definition) </signature>
<example><apply><inverse/><sin/></apply></example>
<example>
  <apply>
  <inverse definitionURL="www.w3c.org/MathML/Content/arcsin"/>
  <sin/>
  </apply>
</example>
<property><apply><forall/>
  <bvar><ci>y</ci></bvar>
  <apply><sin/>
    <apply>
      <apply><inverse/><sin/></apply>
      <ci>y</ci>
    </apply>
  </apply>
  <value><ci>y</ci></value>
</apply>
</property>
<property>
<apply>
  <apply><inverse/><sin/></apply>
  <apply>
    <sin/>
    <ci>x</ci>
  </apply>
</apply>
<value><ci>x</ci></value>
</property>
<property>F(inverse(F)(y))<value>y</value></property>
</MathMLdefinition>
```

F.2.2.6. <sep>

```
<MathMLdefinition>
<name> sep </name>
<description>
  This is the MathML infix constructor used to sub-divide PCDATA into
  separate components. for example, this is used in the description of
  a multipart number such as a rational or a complex number.
</description>
<functorclass> punctuation </functorclass>
<example><cn type="complex-polar">123<sep/>456</cn></example>
<example><cn>123</cn></example>
</MathMLdefinition>
```

F.2.2.7. <condition>

```
<MathMLdefinition>
<name> condition </name>
<description>
```

```
  This is the MathML constructor for building conditions.
  A condition differs from a relation in how it is used.
  A relation is simply an expression, while a condition
  is  used as a predicate to place a conditions on a bound
  variables.

  For a compound condition use relations or apply
  operators such as "and" or "or" or a set of
  relations).
</description>
<functorclass> constructor, unary </functorclass>
<signature> ({reln|apply|set}) -> predicate </signature>
<example>
<condition>
  <reln><lt/>
    <apply><power/>
      <ci>x</ci><cn>5</cn>
    </apply>
    <cn>3</cn>
  </reln>
</condition>
</example>
</MathMLdefinition>
```

F.2.2.8. <declare>

```
<MathMLdefinition>
<name> declare </name>
<description>
  This is the MathML constructor for redefining the properties and
  values with mathematical objects.  For example V may be a name
  delcared to be a vector, or V may be a name which stands for a
  particular vector.

  The attribute values of the declare statement are assigned as the
  corresponding default attribute values of the first object.

</description>
<functorclass> modifier , (unary | binary) </functorclass>
<MMLattribute>
<name>definitionURL</definition>
<value> Any valid URL </value>
</MMLattribute>
<MMLattribute>
<name>type</name><value> MathMLType </value>
</MMLattribute>
<MMLattribute>
<name>nargs</name><value> number of arguments for an object of type fn </value>
</MMLattribute>

<signature> [attributename=attributevalue](anything)
  -> anything(attributevalue) </signature>

<!-- The two argument form updates  the properties of the first
object to be those of the second.  The attribute values override the
properties of the "value".
-->

<signature> [attributename=attributevalue](anything,anything)
  -> anything(attributevalue) </signature>
```

```
<example><reln><and/><ci>x</ci><cn>1</cn></reln></example>
<example><reln><lt/><ci>x</ci></reln></example>
</MathMLdefinition>
```

F.2.2.9. <lambda>

```
<MathMLdefinition>
  <name> lambda </name>
  <description> The operation of lambda calculus that makes a
  function from an expression and a variable.  The definition
  at this level uses only one variable.  Lambda is a binary
  function, where the first argument is the variable and
  the second argument is a the expression.
  Lambda( x, F ) is written as \lambda x [F] in the lambda
  calculus literature.
  The lambda function can be viewed as the inverse of function
  application.

  Although the expression F may contain x, the lambda expression
  is interpreted to be free of x.  That is, the x variable is
  a variable local to the environment of the definition of
  the function or operator.  Formally, lambda(x,F) is free of
  x, and any substitutions, evaluations or tests for x in
  lambda(x,F) should not happen.

  A lambda expression on an arbitrary function applied to a
  simple argument is equivalent to the arbitrary function.
  E.g.  lambda(x, f(x)) == f.  This is a common shortcut.

</description>
  <functorclass> Nary , Constructor </functorclass>
  <property>
    <lambda><ci>x</ci>
      <apply><fn><ci>F</ci></fn><ci>x</ci></apply>
    </lambda>
     <value> <fn><ci>F</ci></fn> </value>
  </property>

<!-- Constructing a variant of the sine function -->

   <example>
      <lambda>
        <ci> x </ci>
        <apply><sin/>
          <apply><plus/>
            <ci> x </ci>
            <cn> 3 </cn>
        </apply>
      </lambda>
   </example>

<!-- the identity operator  -->

  <example>
    <lambda><ci> x </ci> <ci> x </ci> </lambda>
  </example>

  <property>
  <reln><identity/>
    <lambda><ci>x</ci>
      <apply><fn><ci>F</ci></fn><ci>x</ci></apply>
```

```
      </lambda>
      <fn><ci>F</ci></fn>
    </reln>
    </property>
  <MathMLdefinition>
```

F.2.2.10. <compose/>

```
  <MathMLdefinition>
  <name> compose </name>
  <description>
    This is the MathML constructor for composing functions.
    In order for a composition to be meaningful, the range of
    the first function must be the domain of the second function,
    etc.  .

    The result is a new function whose domain is the domain of
    the first function and whose range is the range of the last
    function and whose definition is equivalent to applying
    each function to the previous outcome in turn as in:

    (f @ g )( x )   ==  f( g(x) ).

          This function is often denoted by a small circle infix
   operator.
  </description>

  <functorclass> Nary , Operator </functorclass>

  <signature> (fn*) -> fn </signature>

  <example>
  <apply><compose/>
         <fn><ci> f </ci></fn>
         <fn><ci> g </ci></fn>
       </apply></example>

  <property>
  <apply><forall>
    <bvar><ci>x</ci></bvar>
    <reln><eq/>
      <apply>
        <apply><compose/>
          <ci>f</ci>
          <ci>g</ci>
        </apply>
        <ci>x</ci>
      </apply>
      <apply><ci>f</ci>
        <apply><ci>g</ci>
          <ci>x</ci>
        </apply>
      </apply>
    </reln>
  </apply>
  </property>
  </MathMLdefinition>
```

F.2.2.11. <ident/>

```
<MathMLdefinition>
<name> ident </name>
<description>
  This is the MathML constructor for the identity function.
  This function has the property that

       f( x ) = x,  for all x in its domain.

</description>

<functorclass> Nary , Operator </functorclass>

<signature> (symbol) -> symbol </signature>

<example>
<apply><ident/>
       <ci> f </ci>
       <ci> x </ci>
</apply>
</example>

<property>
<apply><forall>
  <bvar><ci>x</ci></bvar>
  <reln><eq/>
    <apply><ident/>
        <ci>f</ci>
        <ci>x</ci>
    </apply>
    <ci>x</ci>
    </reln>
</apply>
</property>
</MathMLdefinition>
```

F.2.3. Arithmetic, Algebra and Logic

F.2.3.1. <quotient/>

```
<MMLdefinition>
<name> quotient </name>
    <description> Integer quotient, the result of integer
      division.  For arguments a and b, it returns q,
      where a = b*q+r,  |r| < |b|  and  a*r &ge; 0  (or
      the sign of r is the same as the sign of a).
     </description>
       <functorclass> Binary, Function </functorclass>
       <signature> (integer, integer) -> integer </signature>
       <signature> (symbolic, symbolic) -> symbolic </signature>

<!--
ForAll(bvar(a,b),identity(a ,b*Quotient(a,b) + Remainder(a,b))
-->
       <property>
       <apply><forall/>
         <bvar><ci>a</ci></bvar>
         <bvar><ci>b</ci></bvar>
         <reln/><eq/>
             <ci>a</ci>
             <apply><plus/>
               <apply><times/>
```

```
                  <ci>b</ci>
                  <apply><quotient/><ci>a</ci><ci>b</ci></apply>
                </apply>
                <apply><rem/><ci>a</ci><ci>b</ci></apply>
              </apply>
            <reln>
          </apply>
          </property>

       <!-- 1 = quotient(5,4) -->

       <property>
        <apply><identity/>
          <apply><quotient/>
            <ci>5</ci>
            <ci>4</ci>
          </apply>
          <ci>1</ci>
       <apply>
       </property>
    </MMLdefinition>
```

F.2.3.2. <exp/>

```
    <MMLdefinition>
      <name> exp </name>
        <description> The exponential function.
                <Reference> M. Abramowitz and I. Stegun, Handbook of
                    Mathematical Functions, [4.2]
                </Reference>
        </description>
            <functorclass> Unary, Function </functorclass>
            <signature> real -> real </signature>
            <signature> symbolic -> symbolic </signature>
            <property><reln><eq/>
              <apply><exp/><cn>0</cn></apply>
              <cn>1</cn></reln>
            </property>
            <property><apply><identity/>
              <apply><exp/><ci>x</ci></apply>
              <apply><power/>
                <cn>ExponentialE;</cn><ci>x</ci>
              </apply>
            </apply>
            </property>
            <property> exp(x) = limit( (1+x/n)^n, n, infinity ) </property>

    </MMLdefinition>
```

F.2.3.3. <factorial/>

```
    <MMLdefinition>
      <name>
        factorial
      </name>
      <description>
        This element is used to construct factorials
        as in n! = n * (n-1) * (n-2 ) ... 1 .
      </description>
      <functorclass> Unary , function </functorclass>
      <signature> ( algebraic ) -> algebraic </signature>
```

```
<example> <apply><factorial/><ci>n</ci></apply> </example>

<!-- for all n > 0, n! = n*(n-1)! -->

<property><apply><forall/>
  <bvar><ci>n<ci></bvar>
  <condition>
    <reln><gt/><ci>n</ci><cn>0</cn></reln>
  </condition>
  <reln><eq/>
    <apply><factorial/><ci>n</ci></apply>
    <apply><times/>
      <ci>n</ci>
      <apply><factorial/>
        <apply><minus/><ci>n</ci><cn>1</cn></apply>
      </apply>
    </apply>
  </reln>
</property>
</MMLdefinition>
```

F.2.3.4. <divide/>

```
<MMLdefinition>
  <name> divide </name>
  <description>
    The MathML operator that is used to construct
    a "divided by" b.  If a and b are from an algebraic
    domain with a non-commutative times then this is defined
    as a * (b)^(-1).  The result is from the same algebraic
    domain as the operands.
  </description>
  <MMLattribute>
    <name> type </name>
    <value> non-commutative </name>
    <default> none </default>
  </MMLattribute>
  <functorclass> binary , function </functorclass>
  <signature> (complex, complex) -> complex </signature>
  <signature> (real, real) -> real </signature>
  <signature> (rational, rational) -> rational </signature>
  <signature> (integer, integer) -> rational </signature>
  <signature> (symbolic, symbolic) -> symbolic </signature>
  <example>
    <apply> <divide/>
     <ci> a </ci>
     <ci> b </ci>
    </apply>
  </example>
  <property>
    <apply><forall/>
      <bvar>a</bvar>
      <reln><eq/>
      <apply> <divide/>
        <ci> a </ci>
        <ci> 0 </ci>
      <ci>Error, Division by 0</ci>
    </apply>
  </property>
  </MMLdefinition>
```

F.2.3.5. <max/>

```
<MMLdefinition>
  <name> max  </name>
  <description>
    Represent the maximum of a set of elements.  The elements
    may be given explicitly or described by membership in
    some set.  To be well defined, the elements must all be
    comparable.    </description>
  <functorclass> function </functorclass>
  <signature> ( ordered_set_element * ) -> ordered_set_element </signature>
  <signature> ( condition ) ->  ordered_set_element </signature>
  <example>
    <apply><max/><cn>2</cn><cn>3</cn> <cn>5</cn> </apply>
  </example>
  <example>
    <apply><max/>
       <condition>
         <bvar><ci>x</ci></bvar>
         <reln> <notin/>
           <ci> x </ci>
           <ci type="set"> B </ci>
         </reln>
       </condition>
     </apply>
  </example>
  </MMLdefinition>
```

F.2.3.6. <min/>

```
<MMLdefinition>
  <name> min  </name>
  <description>
    Represent the minimum of a set of elements.  The elements
    may be given explicitly or described by membership in
    some set.  To be well defined, the elements must all be
    comparable.    </description>
  <functorclass> function </functorclass>
  <signature> ( ordered_set_element * ) -> ordered_set_element </signature>
  <signature> ( condition ) ->  ordered_set_element </signature>
  <example>
    <apply><min/><cn>2</cn><cn>3</cn> <cn>5</cn> </apply>
  </example>
  <example>
    <apply><min/>
       <condition>
         <bvar><ci>x</ci></bvar>
         <reln> <notin/>
           <ci> x </ci>
           <ci type="set"> B </ci>
         </reln>
       </condition>
     </apply>
  </example>
  </MMLdefinition>
```

F.2.3.7. <minus/>

```
<MMLdefinition>
  <name> minus  </name>
  <description>
```

```
    The subtraction operator of a group.  </description>
  <MMLattribute>
    <name> definitionURL </name>
    <value> URL </name>
    <default> none </default>
  </MMLattribute>
  <functorclass>
    Operator , (Unary | Binary )
  </functorclass>
  <signature>(real,real) -> real</signature>
  <signature>(integer,integer) -> integer</signature>
  <signature>(rational,rational) -> rational</signature>
  <signature>(complex,complex) -> complex</signature>

<!--
  Note that complex-cartesian is a data input format,
  but the resulting data type is complex. !
-->

  <signature> (vector,vector) -> vector</signature>
  <signature>(matrix,matrix) -> matrix</signature>
  <signature>(real) -> real </signature>
  <signature>(integer) -> integer </signature>
  <signature>(complex) -> complex </signature>
  <signature>(rational) -> rational </signature>
  <signature>(vector) -> vector </signature>
  <signature>(matrix) -> matrix </signature>
 <example>
    <apply><minus/><cn>3</cn><cn>5</cn></apply>
 </example>
 <example>
    <apply><minus/><cn>3</cn></apply>
 </example>

<!-- Definition of the unary operator (-1) = -( 1 ) -->

<property>
    <reln><eq/>
       <bvar><ci>n</ci>
       <apply><minus/><cn>1</cn></apply>
       <cn>-1</cn>
    </reln>
  </property>
  </MMLdefinition>
```

F.2.3.8. <plus/>

```
<MMLdefinition>
<name> plus </name>
<description> The N-ary addition operator of an
algebraic structure.

If no operands are provided, the expression represents
the additive identity.

If one operand a is provided, the expression represents
a.

If two or more operands are provided, the expression
represents the group element corresponding to a left
associative binary pairing of the operands.
```

```
Issues with regard to the "value" of mixed operands
are left up to the target system.  If the author wishes
to refer to specific type coercion rules, then
the definitionURL attribute should be used to refer
to a suitable specification.
</description>

<functorclass>  Operator , Nary </functorclass>
<signature>(real,real) -> real</signature>
<signature>(integer,integer) -> integer</signature>
<signature>(rational,rational) -> rational</signature>
<signature> (vector,vector) -> vector</signature>
<signature>(matrix,matrix) -> matrix</signature>
<signature>(complex,complex) -> complex</signature>
<signature>(symbolic,symbolic) -> symbolic </signature>

<signature> real -> real </signature>
<signature> rational -> rational </signature>
<signature> integer -> integer </signature>
<signature> symbolic -> symbolic </signature>
<signature>(real) -> real </signature>
<signature>(integer) -> integer </signature>
<signature>(complex) -> complex </signature>
<signature>(rational) -> rational </signature>
<signature>(vector) -> vector </signature>
<signature>(matrix) -> matrix </signature>

<example><apply><plus/><cn>3</cn></apply></example>
<example><apply><plus/><cn>3</cn><cn>5</cn></apply></example>
<example><apply><plus/><cn>3</cn><cn>5</cn><cn>7</cn></apply></example>

<!-- The properties for more restricted algebraic structures should
be defined using a definitionURL
-->

<property> +() = 0 </property>
<property> +(a) = a </property>
<property> ForAll(a,Commutative, a + b = b + a)</property>
</MMLdefinition>
```

F.2.3.9. <power/>

```
<MMLdefinition>
<name> power </name>
<description> The powering operator </description>
<functorclass> binary, operator </functorclass>
<signature> (complex complex) -> complex </signature>
<signature> (real real) -> complex </signature>
<signature> (rational rational) -> complex </signature>
<signature> (rational integer) -> rational </signature>
<signature> (integer integer) -> rational </signature>
<signature> (symbolic symbolic) -> symbolic </signature>
<property> ForAll(a,Condition(a<>0),a^0=1) </property>
<property> ForAll(a,a^1=a) </property>
<property> ForAll(a,1^a=1) </property>
<property>ForAll(a,0^0=Undefined)</property>
    </MMLdefinition>
```

F.2.3.10. <rem/>

```
<MMLdefinition>
<name> rem </name>
<description> Integer remainder, the result of integer
division.  For arguments a and b, it returns r,
where a = b*q+r,  |r| < |b|  and  a*r &ge; 0  (the
sign of r is the same as the sign of a when both are
non-zero).
</description>
<functorclass> binary, function </functorclass>
<signature> (integer integer) -> integer </signature>
<signature> (symbolic symbolic) -> symbolic </signature>
<property> a = b*rem(a,b) + rem(a,b) </property>
<property>rem(a,0) = Division_by_Zero</property>
</MMLdefinition>
```

F.2.3.11. <times/>

```
<MMLdefinition>
<name> times </name>
<description> The multiplication operator of any
ring.
</description>
<functorclass> N-ary, Operator </functorclass>
<signature> (complex complex) -> complex </signature>
<signature> (real, real) -> real </signature>
<signature> (rational, rational) -> rational </signature>
<signature> (integer, integer) -> integer </signature>
<signature> (symbolic, symbolic) -> symbolic </signature>
<property>ForAll(bvars(a,b),condition(in({a,b},Commutative)),a*b=b*a)</property>
<property>ForAll(bvars(a,b,c),Associative,a*(b*c)=(a*b)*c), associativity </property>
<property> a*1=a </property>
<property> 1*a=a </property>
<property> a*0=0 </property>
<property> 0*a=0 </property>
</MMLdefinition>
```

F.2.3.12. <root/>

```
<MMLdefinition>
  <name> root  </name>
  <description>
    Construct the nth root of an object.
    The first argument "a" is the object and the
    second object "n" denotes the root, as in

              ( a ) ^ (1/n)

  </description>
  <MMLattribute>
    <name> type </name>
    <value> real | complex | principle_branch </name>
    <default> real </default>
  </MMLattribute>
  <functorclass> binary , function </functorclass>
  <signature> ( anything , symbol ) -> root </signature>
  <example>
    <apply> <root/>
        <ci> a </ci>
        <ci> n </ci>
     </apply>
  </example>
```

```
  <property> Forall(bvars(a,n),root(a,n) = a^(1/n)) </property>
  </MMLdefinition>
```

F.2.3.13. <gcd/>

```
<MMLdefinition>
  <name> gcd  </name>
  <description>
    This represents the greatest common divisor
    of its arguments.
  </description>
  <MMLattribute>
    <name> type </name>
    <value> anything </name>
    <default> integer </default>
  </MMLattribute>
  <functorclass> Function , Nary </functorclass>
  <signature> [type=typevalue](typevalue*) -> typevalue </signature>
  <example>
    <apply><gcd/><cn>12</cn> <cn>17</cn></apply>
  </example>
  <property>Forall(p,q,(is(p,prime) and is(q,prime)) , gcd(p,q)=1 </property>
  </MMLdefinition>
```

F.2.3.14. <and/>

```
<MMLdefinition>
  <name> and  </name>
  <description>
    This is the logical "and" operator.
  </description>
  <functorclass> function, Nary </functorclass>
  <signature> (boolean*) -> boolean </signature>
  <example> <apply><and/><ci>p</ci><ci>q</ci></apply> </example>
  <property> identity(true and p , p ) </property>
  <property> identity(p and q , q and p ) </property>
  </MMLdefinition>
```

F.2.3.15. <or/>

```
<MMLdefinition>
<name> or </name>
<description> The logical "or" operator.
</description>
<functorclass> Binary, Function </functorclass>
<signature> (boolean,boolean) -> boolean </signature>
<signature> [type=boolean](symbolic symbolic)
     -> symbolic </signature>
<property> identity(true or p , true ) </property>
        ...
</MMLdefinition>
```

F.2.3.16. <xor/>

```
<MMLdefinition>
<name> or </name>
<description> The logical "xor" operator.
</description>
<functorclass> Binary, Function </functorclass>
<signature> (boolean,boolean) -> boolean </signature>
<signature> [type=boolean](symbolic symbolic)
```

```
     -> symbolic </signature>
<property> ...</property>
</MMLdefinition>
```

F.2.3.17. <not/>

```
<MMLdefinition>
<name> not </name>
<description> The logical "not" operator.
</description>
<functorclass> Unary, Function </functorclass>
<signature> (boolean) -> boolean </signature>
<signature> [type=boolean](symbolic)
     -> symbolic </signature>
<property> ... </property>
</MMLdefinition>
```

F.2.3.18. <implies/>

```
<MMLdefinition>
<Name> implies </Name>
<description> The implies operator.  This represents
the construction  "A implies B".
</description>
<functorclass> Binary, relation </functorclass>
<signature> (boolean,boolean) -> boolean </signature>

<property> <apply></forall>
   <bvar><ci>A</ci></bvar>
   <bvar><ci>B</ci></bvar>
   <reln><eq/>
      <apply><implies/>
        <ci>A</ci>
        <ci>B</ci>
      </apply>
      <apply><or/>
        <ci>B</ci>
        <apply><not/>
          <ci> A </ci>
        </apply>
      </apply>
    </reln>
 </property>
</MMLdefinition>
```

F.2.3.19. <forall/>

```
<MMLdefinition>
<name> forall </name>
<description> The logical "For all" quantifier.
</description>
<functorclass> Nary, Operator </functorclass>
<signature> (bvar*,condition?,(reln|apply)) -> boolean </signature>
<property> ... </property>
</MMLdefinition>
```

F.2.3.20. <exists/>

```
<MMLdefinition>
```

```
<name> exists </name>
<description> The logical "There exists" quantifier.
</description>
<functorclass> Nary, Operator </functorclass>
<signature> (bvar*,condition?,(reln|apply)) -> boolean </signature>
<property> ... </property>
</MMLdefinition>
```

F.2.3.21. <abs/>

```
<MMLdefinition>
<name> exists </name>
<description> The absolute value of a number.
</description>
<functorclass> Unary, Operator </functorclass>
<signature> (algebraic) -> algebraic </signature>
<property> ... </property>
</MMLdefinition>
```

F.2.3.22. <conjugate/>

```
<MMLdefinition>
<name> conjugate </name>
<description> The "conjugate" arithmetic operator
used to represent the conjugate of a complex number.
</description>
<functorclass> Unary, Operator </functorclass>
<signature> (algebraic) -> algebraic </signature>
<property> ... </property>
</MMLdefinition>
```

F.2.4. Relations

F.2.4.1. <eq/>

```
<MMLdefinition>
<Name> eq </Name>
<description> The equality operator. </description>
<functorclass> Nary, relation </functorclass>
<property> Commutative </property>
<signature> (symbolic symbolic) -> boolean </signature>
</MMLdefinition>
```

F.2.4.2. 2<neq/>

```
<MMLdefinition>
<Name> neq </Name>
<description> The notequals operator. </description>
<functorclass> Nary, relation </functorclass>
<property> Commutative </property>
<signature> (symbolic symbolic) -> boolean </signature>
</MMLdefinition>
```

F.2.4.3. <gt/>

```
<MMLdefinition>
<Name> gt </Name>
```

```
<description> The equality operator. </description>
<functorclass> binary, relation </functorclass>
<property> Commutative </property>
<signature> (symbolic symbolic) -> boolean </signature>
</MMLdefinition>
```

F.2.4.4. <lt/>

```
<MMLdefinition>
<Name> lt </Name>
<description> The inequality equality operator "<" </description>
<functorclass> binary, relation </functorclass>
<property> Commutative </property>
<signature> (symbolic, symbolic*) -> boolean </signature>
</MMLdefinition>
```

F.2.4.5. <geq/>

```
<MMLdefinition>
<Name> geq </Name>
<description> The inequality operator. >= </description>
<functorclass> Nary, relation </functorclass>
<signature> (symbolic, symbolic*) -> boolean </signature>
<property> ... Commutative ? ... </property>
</MMLdefinition>
```

F.2.4.6. <leq/>

```
<MMLdefinition>
<Name> leq </Name>
<description> The inequality operator  </description>
<functorclass> Nary, relation </functorclass>
<property> Commutative </property>
<signature> (symbolic symbolic) -> boolean </signature>
</MMLdefinition>
```

F.2.5. Calculus

F.2.5.1. <ln/>

```
    <MMLdefinition>
        <Name> ln </Name>
        <description> The logarithmic function.  Also called
            the natural logarithm.
            The inverse of the exponential function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.1]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <property>
            Error( "logarithm has a singularity at 0" )
        </property>
```

```
        <signature> Intersect(real,positive) -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property> ln(1) = 0 </property>
        <property> ln(exp(x)) = x, "for real x" </property>
        <property> exp(ln(x)) = x, always </property>
    </MMLdefinition>
```

F.2.5.2. <log/>

```
    <MMLdefinition>
        <Name> log </Name>
        <description> The logarithmic function (base 10), or any
         any other user specified base.  Also called
            the natural logarithm.
            The inverse of the exponential function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.1]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> (real,logbase) -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property>
            Error( "logarithm has a singularity at 0" )
        </property>
      </MMLdefinition>
```

F.2.5.3. <int/>

```
  <MMLdefinition>
        <Name> int </Name>
        <description>
            The definite or indefinite integral of a function or algebraic
            expression.

            There are several forms of calling sequences depending on
            the nature of the areguments, and whether or not it is a
            definite integral.

        </description>
        <functorclass> Binary , Function </functorclass>
        <signature> (function) -> function </signature>
        <signature> (algebraic,bvar) -> algebraic </signature>
        <signature> (algebraic,bvar,interval) -> algebraic </signature>
        <signature> (algebraic,bvar,condition) -> algebraic </signature>
    </MMLdefinition>
```

F.2.5.4. <diff/>

```
    <MMLdefinition>
        <Name> diff </Name>
        <description>
            For expressions, this represents the derivative of
            its first argument evaluated at the second argument.
            For Unary functions (only one argument) it represents
            f'.
        </description>
        <functorclass> (Unary | Binary) , Function </functorclass>
```

```
            <signature> (algebraic,bvar) -> algebraic </signature>
            <property>Forall(x,diff( sin(x) , x ) = cos(x)) </property>
            <property>Forall(x,diff( x , x ) = 1 ) </property>
            <property>Forall(x,diff( x^2 , x ) = 2x) </property>
            <property>identity( diff(sin) , cos ) </property>
        </MMLdefinition>
```

F.2.5.5. <partialdiff/>

```
       <MMLdefinition>
            <Name> partialdiff </Name>
            <description>
                For expressions, this represents the derivative of
                its first argument evaluated at the second argument.
                For Unary functions (only one argument) it represents
                f'.
            </description>
            <functorclass> (Binary) , Function </functorclass>
            <signature> (algebraic,bvar) -> algebraic </signature>
            <property>Forall(x,diff( sin(x*y) , x ) = cos(x)) </property>
            <property>Forall(x,y,diff( x*y , x ) = diff(x,x)*y + diff(y,x)*x )
    </property>
            <property>Forall(x,a,b,diff( a + b , x ) = diff(a,x) + diff(b,x) )
    </property>
            <property>identity( diff(sin) , cos ) </property>
        </MMLdefinition>
```

F.2.5.6. <lowlimit/>

```
        <MMLdefinition>
            <Name> lowlimit </Name>
            <description> Construct a lower limit.  Limits
             are used in some integrals as alternative way
             of describing the region over which an integral
             is computed. (i.e., a connected component of the
             real line.)
             </description>
            <functorclass> Constructor </functorclass>
            <signature> (anything*) -> list </signature>
        </MMLdefinition>
```

F.2.5.7. <uplimit/>

```
        <MMLdefinition>
            <Name> uplimit </Name>
            <description> Construct a an upper limit.  Limits
             are used in some integrals as alternative way
             of describing the region over which an integral
             is computed. (i.e., a connected component of the
             real line.)
             </description>
            <functorclass> Constructor </functorclass>
            <signature> (anything*) -> list </signature>
        </MMLdefinition>
```

F.2.5.8. <bvar/>

```
        <MMLdefinition>
            <Name> bvar </Name>
            <description>
    The bvar element is the container element
    for the "bound variable" of an operation.
    For example, in an integral it specifies the
    variable of integration. In a derivative, it
    indicates which variable with respect to
    which a function is being differentiated.

    When the bvar element is used to quantifiy a derivative,
    the bvar element may contain a child degree element which
    specifies the order of the derivative with respect to that
    variable. The bvar element is also used for the internal
    variable in sums and products.
        </description>
            <functorclass> Constructor </functorclass>
            <signature> (symbol) -> symbol </signature>
            <example> <bvar><ci>x</ci></bvar></example>
        </MMLdefinition>
```

F.2.5.9. <degree/>

```
        <MMLdefinition>
            <Name> degree </Name>
            <description> A parameter used by some
            MathML data-types to specify that, for example,
            a bound variable is repeated several times.
            </description>
            <functorclass> Constructor </functorclass>
            <signature> (algebraic) -> algebraic </signature>
            <example> <degree><ci>x</ci></degree></example>

            <property> ... </property>
        </MMLdefinition>
```

F.2.6. Theory of Sets

F.2.6.1. <set>

```
        <MMLdefinition>
            <Name> set </Name>
            <description> Construct a set. </description>
            <functorclass> Nary, Constructor </functorclass>
            <signature> (anything*) -> set </signature>
        </MMLdefinition>
```

F.2.6.2. <list>

```
        <MMLdefinition>
            <Name> list </Name>
            <description> Construct a list. </description>
            <functorclass> Nary, Constructor </functorclass>
            <signature> (anything*) -> list </signature>
        </MMLdefinition>
```

F.2.6.3. <union/>

```
<MMLdefinition>
    <Name> union </Name>
    <description> The union of two sets. </description>
    <functorclass> Binary, Function </functorclass>
    <signature> (set*) -> set </signature>
</MMLdefinition>
```

F.2.6.4. <intersect/>

```
<MMLdefinition>
    <Name> intersection </Name>
    <description> The intersection of two sets. </description>
    <functorclass> Binary, Function </functorclass>
    <signature> (set set) -> set </signature>
</MMLdefinition>
```

F.2.6.5. <in/>

```
<MMLdefinition>
    <Name> in </Name>
    <description>
        The membership testing operation (also commonly
        called "in" or "including").  Returns true if the first
        argument is part of the second argument.  The second
        argument must be a set.
    </description>
    <functorclass> Binary, Function </functorclass>
    <signature> (anything, set) -> boolean </signature>
</MMLdefinition>
```

F.2.6.6. <notin/>

```
<MMLdefinition>
<Name> notin </Name>
<description>
    The membership exclusion operation (also commonly
    called "notin" or "including").
    It is defined as "not in".
</description>
    <functorclass> Binary, Function </functorclass>
    <signature> (anything set) -> boolean </signature>
</MMLdefinition>
```

F.2.6.7. <subset/>

```
<MMLdefinition>
<Name> subset </Name>
<description>
  Boolean function whose value is determined by
  whether or not one set is a subset of another.
 </description>
    <functorclass> Binary, Function </functorclass>
    <signature> (set*) -> boolean </signature>
</MMLdefinition>
```

F.2.6.8. <prsubset/>

```
<MMLdefinition>
<Name> prsubset </Name>
<description>
  Boolean function whose value is determined by
  whether or not one set is a proper subset of another.
 </description>
    <functorclass> Binary, Function </functorclass>
    <signature> (set, set) -> boolean </signature>
    <property>...</property>
</MMLdefinition>
```

F.2.6.9. <notsubset/>

```
<MMLdefinition>
<Name> notsubset </Name>
<description>
  Boolean function whose value is the complement
  of "subset".
 </description>
    <functorclass> Binary, Function </functorclass>
    <signature> (set, set) -> boolean </signature>
    <property>...</property>
</MMLdefinition>
```

F.2.6.10. <notprsubset/>

```
<MMLdefinition>
<Name> notprsubset </Name>
<description>
  Boolean function whose value is the complement
  of "proper subset".
 </description>
    <functorclass> Binary, Function </functorclass>
    <signature> (set, set) -> boolean </signature>
    <property>...</property>
</MMLdefinition>
```

F.2.6.11. <setdiff/>

```
<MMLdefinition>
<Name> setdiff </Name>
<description>
  Function indicating the difference of two sets.
</description>
<functorclass> Binary, Function </functorclass>
<signature> (set, set) -> set </signature>
<property>...</property>
</MMLdefinition>
```

F.2.7. Sequences and Series

F.2.7.1. <sum/>

```
<MMLdefinition>
<Name> sum </Name>
<description>
```

```
The sum element denotes the summation operator. Upper and lower
limits for the sum, and more generally a domains for the bound variables
are specified using uplimit, lowlimit or a condition on the bound
variables. The index for the summation is specified by a bvar element.

The sum element takes the attribute definition which can be used to
override the default semantics.
</description>
<functorclass> Unary, Function </functorclass>
<signature> (bvar*,((lowlimit,uplimit)|condition),algebraic) -> sum </signature>
<signature> ... </signature>
</MMLdefinition>
```

F.2.7.2. <product/>

```
<MMLdefinition>
<Name> product </Name>
<description>
The product element denotes the product operator. Upper and lower
limits for the product, and more generally a domains for the bound
variables are specified using uplimit, lowlimit or a condition on the
bound variables. The index for the product is specified by a bvar
element.

The product element takes the attribute definition which can be used
to override the default semantics.
</description>
<functorclass> Unary, Function </functorclass>
<signature> (bvar*,((lowlimit,uplimit)|condition),algebraic)
   -> product </signature>
<signature> ... </signature>
<signature> ... </signature>
</MMLdefinition>
```

F.2.7.3. <limit/>

```
<MMLdefinition>
<Name> limit </Name>
<description>
The sum element denotes the summation operator.
Upper and lower limits for the sum, and more
generally a domains for the bound variables are
specified using uplimit, lowlimit or a condition
on the bound variables. The index for the summation is
specified by a bvar element.
</description>
<functorclass> Nary, Function </functorclass>
<signature> (bvar*,(lowlimit | condition*),algebraic)
 -> limit </signature>
</MMLdefinition>
```

F.2.7.4. <tendsto/>

```
<MMLdefinition>
<Name> tendsto </Name>
<description> tendsto is used to specify how a limit is
computed. It accepts a type attribute that determines the
manner in which it tends to a value.
```

```
</description>
<functorclass> binary, Function </functorclass>
<signature> (symbol,anything) -> condition(limit) </signature>
<signature> [type=direction](symbol,anything) ->
   condition(limit) </signature>
</MMLdefinition>
```

F.2.8. Trigonometry

F.2.8.1. <sin/>

```
<MMLdefinition>
    <Name> sin </Name>
    <description> The circular trigonometric function sine
        <Reference> M. Abramowitz and I. Stegun, Handbook of
            Mathematical Functions, [4.3]
        </Reference>
    </description>
    <functorclass> Unary, Function </functorclass>
    <signature> real -> real </signature>
    <signature> symbolic -> symbolic </signature>
    <property> sin(0) = 0 </property>
    <property> sin(integer*Pi) = 0 </property>
    <property> sin((Z+1/2)*Pi) = (-1)^Z, "for integer Z" </property>
    <property> -1 <= sin(real) </property>
    <property> sin(real) <= 1 </property>
    <property> sin(3*x)=-4*sin(x)^3+3*sin(x), "triple angle formula"
        <Reference> ditto, [4.3.27] </Reference>
    </property>
</MMLdefinition>
```

F.2.8.2. <cos/>

```
<MMLdefinition>
    <Name> cos </Name>
    <description> The cosine function.
        <Reference> M. Abramowitz and I. Stegun, Handbook of
            Mathematical Functions, [4.3]
        </Reference>
    </description>
    <functorclass> Unary, Function </functorclass>
    <signature> real -> real </signature>
    <signature> symbolic -> symbolic </signature>
    <property> cos(0) = 1 </property>
    <property> cos(integer*Pi+Pi/2) = 0 </property>
    <property> cos(Z*Pi) = (-1)^Z, "for integer Z" </property>
    <property> -1 <= cos(real) </property>
    <property> cos(real) <= 1 </property>
</MMLdefinition>
```

F.2.8.3. <tan/>

```
<MMLdefinition>
    <Name> tan </Name>
    <description> The tangent function.
        <Reference> M. Abramowitz and I. Stegun, Handbook of
            Mathematical Functions, [4.3]
        </Reference>
```

```
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> real -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property> tan(integer*Pi) = 0 </property>
        <property> tan(x) = sin(x)/cos(x) </property>
    </MMLdefinition>
```

F.2.8.4. <sec/>

```
    <MMLdefinition>
        <Name> sec </Name>
        <description> The secant function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.3]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> real -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property> sec(x) = 1/cos(x) </property>
    </MMLdefinition>
```

F.2.8.5. <csc/>

```
    <MMLdefinition>
        <Name> csc </Name>
        <description> The cosecant function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.3]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> real -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property> csc(x) = 1/sin(x) </property>
    </MMLdefinition>
```

F.2.8.6. <cot/>

```
    <MMLdefinition>
        <Name> cot </Name>
        <description> The cotangent function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.3]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> real -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property> cot(integer*Pi+Pi/2) = 0 </property>
        <property> cot(x) = cos(x)/sin(x) </property>
    </MMLdefinition>
```

F.2.8.7. <sinh/>

```
    <MMLdefinition>
```

```
        <Name> sinh </Name>
        <description> The hyperbolic sine function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.3]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> real -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property>...</property>
    </MMLdefinition>
```

F.2.8.8. <cosh/>

```
    <MMLdefinition>
        <Name> sinh </Name>
        <description> The hyperbolic sine function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.3]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> real -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property>...</property>
    </MMLdefinition>
```

F.2.8.9. <tanh/>

```
    <MMLdefinition>
        <Name> tanh </Name>
        <description> The hyperbolic tangent function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.3]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> real -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property>...</property>
    </MMLdefinition>
```

F.2.8.10. <sech/>

```
    <MMLdefinition>
        <Name> sech </Name>
        <description> The hyperbolic secant function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.3]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> real -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property>...</property>
    </MMLdefinition>
```

F.2.8.11. <csch/>

```
<MMLdefinition>
    <Name> csch </Name>
    <description> The hyperbolic cosecant function.
        <Reference> M. Abramowitz and I. Stegun, Handbook of
            Mathematical Functions, [4.3]
        </Reference>
    </description>
    <functorclass> Unary, Function </functorclass>
    <signature> real -> real </signature>
    <signature> symbolic -> symbolic </signature>
    <property>...</property>
</MMLdefinition>
```

F.2.8.12. <coth/>

```
<MMLdefinition>
    <Name> coth </Name>
    <description> The hyperbolic cotangent function.
        <Reference> M. Abramowitz and I. Stegun, Handbook of
            Mathematical Functions, [4.3]
        </Reference>
    </description>
    <functorclass> Unary, Function </functorclass>
    <signature> real -> real </signature>
    <signature> symbolic -> symbolic </signature>
    <property>...</property>
</MMLdefinition>
```

F.2.8.13. <arcsin/>

```
<MMLdefinition>
    <Name> arcsin </Name>
    <description> The inverse of the sine function.
        <Reference> M. Abramowitz and I. Stegun, Handbook of
            Mathematical Functions, [4.4]
        </Reference>
    </description>
    <functorclass> Unary, Function </functorclass>
    <signature> real -> real </signature>
    <signature> symbolic -> symbolic </signature>
    <property> sin(arcsin(x)) = x </property>
    <property> arcsin(sin(x)) = x, "for x between -Pi/2 and Pi/2" </property>
</MMLdefinition>
```

F.2.8.14. <arccos/>

```
<MMLdefinition>
    <Name> arccos </Name>
    <description> The inverse of the cosine function.
        <Reference> M. Abramowitz and I. Stegun, Handbook of
            Mathematical Functions, [4.4]
        </Reference>
    </description>
    <functorclass> Unary, Function </functorclass>
    <signature> real -> real </signature>
    <signature> symbolic -> symbolic </signature>
    <property> cos(arccos(x)) = x </property>
```

```
        <property> arccos(cos(x)) = x, "for x between 0 and Pi" </property>
    </MMLdefinition>
```

F.2.8.15. <arctan/>

```
    <MMLdefinition>
        <Name> arctan </Name>
        <description> The inverse of the tangent function.
            <Reference> M. Abramowitz and I. Stegun, Handbook of
                Mathematical Functions, [4.4]
            </Reference>
        </description>
        <functorclass> Unary, Function </functorclass>
        <signature> real -> real </signature>
        <signature> symbolic -> symbolic </signature>
        <property> tan(arctan(x)) = x </property>
        <property> arctan(tan(x)) = x, "for x between -Pi/2 and Pi/2" </property>
    </MMLdefinition>
```

F.2.9. Statistics

F.2.9.1. <mean/>

```
<MMLdefinition>
  <Name> mean </Name>
  <description>
    Given  k unspecified scalar arguments they are treated as equiprobable
    values of a random variable and the mean is computed as:

      mean( a1, a2, ...  an)   Sum( ai, i=1... n )/ n.

    (see section 7.7 in CRC's Standard Mathematical tables and Formulae).

    More generally, if the first argument is a symbol X of type
    "discrete_random_variable", this is the 1st moment of the
    random variable X and is defined as

    E[ X ] = Sum( x*f(x), x in S )

    where the probability that x = x_i is  P( x = x_i) = f(x_i) .

    The arguments are either all data, all discrete random variables,
    or all continuous random variables.

    The generalizes to continuous distributions and
    k dimenions following the definitions provided in the reference:

    <Reference> CRC Standard Mathematical Tables and Formulae,
      editor: Dan Zwillinger, CRC Press Inc., 1996, [7.1.2] and [7.7]
    </Reference>
  </description>
  <MMLattribute>
    <name>type</name>
    <values> random_variable | continuous_random_variable | data </value>
    <default> data </default>
  </MMLattribute>
  <functorclass>Nary , Operator </functorclass>
  <signature>(scalar*) -> scalar</signature>
  <signature>(scalar(type=data)*) -> scalar</signature>
  <signature>(symbol(type=random_variable)*) -> scalar</signature>
```

```
  <signature>(symbol(type=continuous_random_variable)*) -> scalar</signature>
  <property> </property>
</MMLdefinition>
```

F.2.9.2. <sdev/>

```
<MMLdefinition>
  <Name> sdev </Name>
  <description>
    This represents the standard deviation.

    Given  k unspecified scalar arguments they are treated as equiprobable
    values of a random variable and the "standard deviation" is
    computed as the square root of the second moment about the mean  U.

      sdev( a1, a2, ...  an)^2  = E( (X - U)^2 ).

    If the first argument is a symbol X of type
    "discrete_random_variable", then all arguments are treated as
    discrete random variables, instead of data and the second moment
    about the mean is computed as

      Sum( ( x_i - U )^2 * f(x_i) , x_i  in S )
    as


    where the probability that x = x_i is  P( x = x_i) = f(x_i) .

    The arguments are either all data, all discrete random variables,
    or all continuous random variables.

    The generalizes to continuous distributions and to
    k dimenions following the definitions found in:

    <Reference> CRC Standard Mathematical Tables and Formulae,
      editor: Dan Zwillinger, CRC Press Inc., 1996, [7.1.2] and [7.7]
    </Reference>
  </description>
  <MMLattribute>
    <name>type</name>
    <values> random_variable | continuous_random_variable | data </value>
    <default> data </default>
  </MMLattribute>
  <functorclass>Nary , Operator </functorclass>
  <signature>(scalar*) -> scalar</signature>
  <signature>(scalar(type=data)*) -> scalar</signature>
  <signature>(symbol(type=discrete_random_variable)*) -> scalar</signature>
  <signature>(symbol(type=continuous_random_variable)*) -> scalar</signature>
  <property> </property>
</MMLdefinition>
```

F.2.9.3. <variance/>

```
<MMLdefinition>
  <Name> variance </Name>
  <description>
    This computes the second centered moment, also known as the variance.

    Given  k unspecified scalar arguments they are treated as equiprobable
    values of a random variable and the "variance" is
    computed as the second moment about the mean  U.
```

```
      variance( a1, a2, ...  an)  = E( (X - U)^2 ).

    If the first argument is a symbol X of type
    "discrete_random_variable", then all arguments are treated as
    discrete random variables, instead of data and the second moment
    about the mean is computed as  in section [7.7] (see reference below.)

      Sum( ( x_i - U )^2 * f(x_i) , x_i  in S )
    as


    where the probability that x = x_i is  P( x = x_i) = f(x_i) .

    The arguments are either all data, all discrete random variables,
    or all continuous random variables.

    The generalizes to continuous distributions and to
    k dimenions following the definitions found in:

    <Reference> CRC Standard Mathematical Tables and Formulae,
      editor: Dan Zwillinger, CRC Press Inc., 1996, [7.1.2] and [7.7]
    </Reference>
  </description>
  <MMLattribute>
    <name>type</name>
    <values> random_variable | continuous_random_variable | data </value>
    <default> data </default>
  </MMLattribute>
  <functorclass>Nary , Operator </functorclass>
  <signature>(scalar*) -> scalar</signature>
  <signature>(scalar(type=data)*) -> scalar</signature>
  <signature>(symbol(type=discrete_random_variable)*) -> scalar</signature>
  <signature>(symbol(type=continuous_random_variable)*) -> scalar</signature>
</MMLdefinition>
```

F.2.9.4. <median/>

```
<MMLdefinition>
  <Name> median </Name>
  <description>
    This represents the median of n data values.
    If n =2k + 1  then the mode is x_k.
    If n = 2k then the median is (x_k + x_(k+1)/2).
    (Note this discription assumes that the data has been
     sorted into ascending order.)

    <Reference> CRC Standard Mathematical Tables and Formulae,
      editor: Dan Zwillinger, CRC Press Inc., 1996,  [7.7]
    </Reference>
  </description>
  <functorclass>Nary , Operator</functorclass>
  <signature>(scalar*) -> scalar</signature>
</MMLdefinition>
```

F.2.9.5. <mode/>

```
<MMLdefinition>
  <Name> mode </Name>
  <description>
    This represents the mode of  n data values.
```

```
      The mode is the data value that occurs with the
      greatest frequency.

      <Reference> CRC Standard Mathematical Tables and Formulae,
        editor: Dan Zwillinger, CRC Press Inc., 1996,  [7.7]
      </Reference>
    </description>
    <functorclass>Nary , Operator</functorclass>
    <signature>(scalar*) -> scalar</signature>
  </MMLdefinition>
```

F.2.9.6. <moment/>

```
  <MMLdefinition>
    <Name> moment </Name>
    <description>
      This computes the ith moment of a set of data, or a random variable..

      Given  k  scalar arguments of unspecified type, they are treated
      as equiprobable  values of a random variable. and the "moments" are
      computed as the second moment about the mean  U.

        moment( degree=i, scalar*)= E( X^i ).

      If the first data argument x1 is a symbol X of type
      "discrete_random_variable", then all arguments are treated as
      discrete random variables, instead of data and the ith moment
      about the mean is computed as

        Sum( (x)^i * f(x) , x in S )

  where the probability that x = x_i is P( x = x_i) = f(x_i) .

      The arguments are either all data, all discrete random variables,
      or all continuous random variables.

      The generalizes to continuous distributions and to
      k dimenions following the definitions found in:

      <Reference> CRC Standard Mathematical Tables and Formulae,
        editor: Dan Zwillinger, CRC Press Inc., 1996, [7.1.2]
      </Reference>
    </description>
    <MMLattribute>
      <name>type</name>
      <values> random_variable | continuous_random_variable | data </value>
      <default> data </default>
    </MMLattribute>
    <functorclass>Nary , Operator </functorclass>
    <signature>(degree,scalar*) -> scalar</signature>
    <signature>(degree,scalar(type=data)*) -> scalar</signature>
    <signature>(degree,symbol(type=discrete_random_variable)*) -> scalar</signature>
    <signature>(degree, symbol(type=continuous_random_variable)*) -> scalar</signature>
  </MMLdefinition>
```

F.2.10. Lineary Algebra

F.2.10.1. <vector>

```
  <MMLdefinition>
    <Name> vector </Name>
```

```
  <description>
    A vector is an ordered n-tuple of values
    representing an element of an n-dimensional
    vector space.  The "values" are all from the
    same ring, typically real or complex.  They may
           be numbers, symbols, or general algebraic expressions.

    The type attribute can be used to specify the type of
    vector that is represented.
    <Reference> CRC Standard Mathematical Tables and Formulae,
      editor: Dan Zwillinger, CRC Press Inc., 1996, [2.4]
    </Reference>
  </description>
  <MMLattribute>
    <name> type </name>
    <value>  real | complex | symbolic | anything </value>
    <default> real </default>
  </MMLattribute>
  <MMLattribute>
    <name> other </name>
    <value>  row | column </value>
    <default> row </default>
  </MMLattribute>
  <functorclass> constructor , N-ary </functorclass>
  <signature>
    ((cn|ci|apply)*) -> vector(type=real)
  </signature>
  <signature>
    [type=vectortype]((cn|ci|apply)*) -> vector(type=vectortype)
  </signature>
  <property>  <!-- scalar multiplication-->
      <apply><forall/>
        <bvar><ci>b</ci></bvar>
        <bvar><ci>v1</ci></bvar>
        <bvar><ci>v2</ci></bvar>
        <reln>
          <apply><times/>
            <ci>ci>b</ci>
            <vector><ci>ci>v1</ci><ci>ci>v2</ci></vector>
            </apply>
            <vector>
              <apply><ci>b</ci><ci>v1</ci></apply>
              <apply><ci>b</ci><ci>v2</ci></apply>
            </vector>
        </reln>
      </apply>
  </property>
  <property> vector addition </property>
  <property> distributive over scalars</property>
  <property> associativity.</property>
  <property> Matrix * column vector </property>
  <property> row vector * Matrix </property>
  </property>
</MMLdefinition>
```

F.2.10.2. <matrix>

```
<MMLdefinition>
  <Name> matrix </Name>
  <description>
```

```
      This is the constructor for a matrix.  The matrix is
      constructed from matrix rows.  The type and properties
      spell out the normal interaction with vectors and
      scalars.
      <Reference> CRC Standard Mathematical Tables and Formulae,
        editor: Dan Zwillinger, CRC Press Inc., 1996, [2.5.1]
      </Reference>

    </description>
    <MMLattribute>
      <name>type</name>
      <value>real | complex | integer | symbolic | anything </value>
      <default> real </default>
    </MMLattribute>
    <functorclass>constructor , N-ary </functorclass>
    <signature>(matrixrow*) -> matrix</signature>
    <signature>
      [type=matrixtype](matrixrow*) ->
         matrix(type=matrixtype)</signature>
    <property>scalar multiplication </property>
    <property>Matrix*column vector</property>
    <property>Addition</property>
    <property>Matrix*Matrix</property>
  </MMLdefinition>
```

F.2.10.3. <matrixrow>

```
  <MMLdefinition>
    <Name> matrixrow </Name>
    <description>
      This is a constructor for describing the rows of a matrix.
      This only occurs inside a matrix.  Its "type" is determined
      from the containing matrix element.
    </description>
    <functorclass>constructor , N-ary</functorclass>
    <signature>(cn|ci|apply)->matrixrow </signature>
  </MMLdefinition>
```

F.2.10.4. <determinant/>

```
  <MMLdefinition>
    <Name>determinant</Name>
    <description>The "determinant" of a matrix.
      <Reference> CRC Standard Mathematical Tables and Formulae,
        editor: Dan Zwillinger, CRC Press Inc., 1996, [2.5.4]
      </Reference>
    </description>
    <functorclass>Unary, operator</functorclass>
    <signature>(matrix)-> scalar </signature>
    </MMLdefinition>
```

F.2.10.5. <transpose/>

```
  <MMLdefinition>
    <Name> transpose </Name>
    <description>The transpose of a matrix or vector.
      <Reference> CRC Standard Mathematical Tables and Formulae,
        editor: Dan Zwillinger, CRC Press Inc., 1996, [2.4] and [2.5.1]
      </Reference>
    </description>
    <functorclass>Unary, Operator</functorclass>
```

```
  <signature>(vector)->vector(other=row)</signature>
  <signature>[other=column](vector)->vector(other=row)</signature>
  <signature>[other=row](vector)->vector(other=column)</signature>
  <signature>(matrix)->matrix</signature>
  <property>transpose(transpose(A))= A</property>
  <property>transpose(transpose(V))= V</property>
</MMLdefinition>
```

F.2.10.6. <selector/>

```
<MMLdefinition>
  <Name> selector </Name>
  <description>
    The operator used to extract sub-objects from vectors, matrices
    matrix rows and lists.

    Elements are accessed by providing one index element for each
    dimension.  For Matrices, sub-matrices are selected by providing
    one fewer index items.  For a matrix A and a column vector V :

     select( i,j , A ) is the  i,j th element of A.
     select(i , A )  is the matrixrow formed from the ith row of A.
     select( i , V ) is the ith element of V.
     select( V ) is the sequence of all elements of V.
     select(A) is the sequence of all elements of A, extracted row
     by row.
     select(i,L) is the ith element of a list.
     select(L) is the sequence of elements of a list.
  </description>
  <functorclass>N-ary, operator)</functorclass>
  <signature>(scalar,scalar,matrix)->scalar</signature>
  <signature>(scalar,matrix)->matrixrow</signature>
  <signature>(matrix)->scalar* </property>
  <signature>(scalar,(vector|list|matrixrow))->scalar</signature>
  <signature>(vector|list|matrixrow)->scalar*</signature>
  <property>
    Forall(
       bvar(A(type=matrix)),bvar(V(type=vector)),
       select(A) = select(V)
     )
  </property>
  <property>For all vectors V, V = vector(select(V))</property>
</MMLdefinition>
```

Up: Table of Contents

MathML 1.0 Changes

Changes from the 7 April 1998 Specification

Editorial changes

Title page and abstract

Errata and Translatation pages were added. The available formats section was moved and expanded. A link to MathML 1.0 Changes (this document) was added. The phrase "upon which MathML is based" has been added to the sentence "The fundamental eXtensible Markup Language (XML) 1.0 specification has been adopted as a W3C Recommendation" for clarification.

Section 4.2.3.4

Missing discussions of the **max**, **min**, **forall** and **exists** operators which belong in this section were added.

Section 4.2.5

The categorical assertion that a **condition** element must always be accompanied by one or more **bvar** elements was modified to take into account the exceptional usage with **min** and **max**. The first two examples were extended to show the surrounding **apply** elements so that they would be complete MathML expressions instead of fragments.

Section 4.4.10.6

A note was added explaining that even though **select** is classified as an *n*-ary operator, it can only take one, two or three arguments. The definition of the ordering of elements in a sequence of matrix elements was also clarified.

Chapter 6

A new section 6.2.6, "Additional Entity Set Grouping", and corresponding table of contents entry were added. This table collects together entities referred to in the MathML 1.0 specification, but which are not included in the ISO entity lists.

Section 6.2

In order to provide complete and technically valid entity declarations in the MathML 1.0 DTD, entities without current Unicode points have been assigned values in the Unicode Private Zone. The text and tables of values has been amended accordingly.

The entity lists in Section 6.2.4 have been updated to be more in line with the ISO character sets, in that if some part of a set is included then the entire set is included. Also,

ISOCHEM has been dropped. These changes have also been reflected in the entity declarations in the DTD in Appendix A.

Section 6.2.6

This section was added for completeness, as described above.

Appendix A. The MathML DTD

A parseable, online version of the DTD has been added in addition to the preformatted HTML version. In addition, complete entity declarations have been added.

Appendix B. Glossary

The definition of "Attribute" was reworded to be more technically correct, and less misleading. A broken link to a fonts FAQ was removed.

Error Corrections

Section 2.2.2

A bogus **over** element was changed to **divide**, and a spurious *occurrence* attribute was removed, and a missing '/' character in an end **apply** element tag in the second example.

Section 4.2.1.4

A bad link to appendix F was fixed.

Section 4.2.1.6

Quotes were added around the *vector* attribute in the first example.

Section 4.2.1.7

A missing '/' character was added to a **bvar** end tag.

Section 4.2.1.8

A bad link to section 4.2.3.4 was fixed.

Section 4.2.2.3

Missing '/' characters were added to end tags for **bvar**, **uplimit** and **lowlimit**.

Section 4.2.3

<var> was changed to <variance>, and <select> was changed to <selector>.

Section 4.2.3.4

A missing '/' character was added to an **apply** element end tag in the first example. A missing '/' character was added to an **degree** element end tag in the **diff** example, and a missing **ci** element was added around the function identifier *f*. The sentence "Qualifier schemata are always optional." was removed, as the quantifiers **forall** and **exists** require a bound variable.

Section 4.2.6

The outdated **AM_APP** element was replaced with the more current **OMA** element.

Section 4.3.2.4

A misspelling of **xml-annotation** was corrected.

Section 4.4

The **var** and **select** elements were renamed to **variance** and **selector** throughout, to avoid a namespace collision with HTML. Missing trailing '/' characters denoting an empty element were added for **ln**, **log**, **inverse**, **mean**, **sdev**, **variance**, **median**, **mode** and **moment**. The same error was corrected in the corresponding section heading when necessary.

Section 4.4.1.1

Missing quotes around attribute values were added.

Section 4.4.2.8

A missing **bvar** element was added to the lambda construct in the fourth example. A missing '>' character was added to the function declaration in the first example.

Section 4.4.2.9

Misplaced '/' characters were corrected in **lambda** elements in the examples. Out of order qualifiers for the **int** element in the example were corrected by moving the **apply** element to the end.

Section 4.4.2.10

Missing '/' characters were added to the end tags of **apply** elements in the last two examples. An extra '<' character was removed from the fourth example.

Section 4.4.3.18

The textual equivalent of the second example was changed to use the words "such that" in place of "where" for clarity and for agreement with 4.2.5. A missing **power** element was added in the second example. A missing '/' was added to an **apply** end tag, a nesting problem in the **reln** element, and a missing **apply** construct in the second **condition** were corrected in the third example.

Section 4.4.3.21

A missing '/' was added to the **conjugate** tag in the first example.

Section 4.4.4.6

A garbled **reln** tag was corrected in the example.

Section 4.4.5.3

Missing '/' characters were added to the end tags of qualifier schemata in all three examples.

Section 4.4.5.4

A misleading omission of reference to the fact that the **diff** operator can take an optional **degree** element was corrected to be as in 4.4.5.5. The default rendering was improved.

Section 4.4.5.5

Missing '/' characters were added to **degree**, **bvar** and **ci** end tags in the first example. A missing exponent in the numerator of the first differential operator in the sample equation has been added. An incorrect exponent in the default rendering was corrected.

Section 4.4.5.6

Missing '/' characters were added to the end tags of qualifier schemata in the example.

Section 4.4.5.7

A missing '/' character was added to the end tag of the **bvar** in the example.

Section 4.4.5.8

An incorrect **diff** element was changed to an **int** element in the second example.

Section 4.4.5.9

An incorrect **diff** element was changed to a **partialdiff** element in the example.

Section 4.4.7.2

A missing '/' was added to a **bvar** tag in the second example.

Section 4.4.9.3

As noted above, **var** has been renamed to **variance** to avoid a namespace collision with HTML.

Section 4.4.9.6

An incorrectly place **degree** element was moved in the example.

Section 4.4.10.6

As noted above, **select** has been renamed to **selector** to avoid a namespace collision with HTML. A missing '/' was added to the end tag for the **selector** element.

Section 4.4.11.2

The outdated **AM_APP** element was replaced with the more current **OMA** element.

Section 5.1.2

A garbled **msup** element in the first example was corrected. A missing '/' character was added to the **eq** element in the fourth example. An outdated **OMSymbol** element was replaced with the more current **OMS** element. A missing '/' character was added to the **times** element in the fourth example.

Section 5.3

An outdated **OMSymbol** element was replaced with the more current **OMS** element. Missing '/' characters were added to the **transpose** and **times** elements.

Section 7.1.3

Missing '/' characters were added to the **mi** element in the mathml-rendererB example.

Section 7.1.5

Missing emboldening of the anchor tag name was added.

Appendix B. Glossary

Some inconsistent capitalization was corrected in the definitions of Pt, Ex and Em. An extra '(' was removed from the SGML definition. Several instances "schema" were replaced by the plural "schemata". The phrase "free variable" was changed to "variable" in the definition of lambda expression, since the variables referred to are not free. The second use of the word "construct" in the definition of Pre-defined function was changed to "build". The second use of the word "sizes" in the definition of Pt was changed to "objects". The word "of" was changed to "or" in the definition of Token element. Definitions for Box and Bounding Box were added.

Appendix D. Working Group Membership

The first paragraph has been modified to include a reference to the current working group co-chairs, and a pointer to the current working group membership. A new paragraph has been added crediting a number of people who helped identify the errors corrected in this revision. Finally, a broken link to the American Mathematical Society has been corrected.

Appendix F

The **var** and **select** elements were renamed to **variance** and **selector** throughout, to avoid a namespace collision with HTML. Missing '/' characters were added to **selector** elements where they were missing from the corresponding **select** elements.

Section F.2.1.1

A missing ';' was added to the "complex i" entity in the description. A missing '/' was added to the **sep** tag in the second example. A missing '/' was added to the **root** element in the third property, a missing '<' character was added to an **apply** in the fifth property, an erroneous **apply** end tag was removed from the sixth property and a partial duplication of that property was corrected, and a missing **cn** end tag was added in the seventh property. Also, missing quotes were added to the "boolean" type attribute throughout.

Section F.2.2.3

A missing end tag was for the **lambda** element in the last example.

Section F.2.2.5

Garbled shorthand was expanded into proper markup in the last property.

Section F.2.2.9

A missing **apply** end tag was added to the variant sine function example.

Section F.2.2.10

A missing '/' was added to the **forall** element in the first property.

Section F.2.2.11

A missing '/' was added to the **forall** element in the first property.

Section F.2.3.1

Reversed start and end tags for the **reln** element were switched. A missing '/' was added to the last **apply** end tag in the second property.

Section F.2.3.2

A missing end tags for the **apply** element was added, and a missing '/' was added to a **ci** end tag, in the property.

Section F.2.3.4

Missing **ci** tags were added withing the **bvar** in the property, and garbled nesting of the **reln** element was corrected.

Section F.2.3.7

A missing **bvar** end tag was added in the property.

Section F.2.3.18

A misplaced '/' character in the **forall** element was corrected, and a missing **apply** end tag was added in the property.

Section F.2.9.3

<var> was changed to <variance>.

Section F.2.10.6

<select> was changed to <selector>.

Section F.2.10.5

A missing **name** sub-element was added to the **MMLdefinition** for **transpose**.

Up: Table of Contents

References

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

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 (pbk.)

Carroll, Lewis [Rev. C. L. Dodgson]; "Through the Looking Glass and What Alice Found There", Macmillian & Co., 1871

Chaudry,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.

Drucker,Peter; Forbes, 10 Mar 1997 [quoted by Gene Klotz]

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

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

Lie, Håkon Wium and Bert Bos; Cascading Style Sheets, level 1, W3C Recommendation, 17 Dec 1996, http://www.w3.org/pub/WWW/TR/REC-CSS1

OpenMath Release 1, December 1996; www.openmath.org

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

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.

Raggett, Dave, Arnaud Le Hors and Ian Jacobs; HTML 4.0 Specification, 18 Dec 1997, http://www.w3.org/TR/REC-html40/; section on data types

Spivak, M. D.; The Joy of T_EX, A gourmet guide to typesetting with the AMS-T_EX macro package. American Mathematical Society, Providence, RI, MA 1986, xviii + 290 pp. ISBN: 0-8218-2999-8

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

Up: Table of Contents