MathJax Documentation

Release 3.1

Davide Cervone, Volker Sorge, Peter Krautzberger, Robert Miner

Apr 28, 2021

The Basics

MathJax is an open-source JavaScript display engine for LaTeX, MathML, and AsciiMath notation that works in all modern browsers, with built-in support for assistive technology like screen readers.

Version 3.0 of MathJax is a complete rewrite of MathJax from the ground up, and its usage and configuration is significantly different from that of MathJax version 2. Use the green menu at the bottom of the sidebar on the left to access the version 2 documentation if you need it.

CHAPTER 1

What is MathJax?

MathJax is an open-source JavaScript display engine for LaTeX, MathML, and AsciiMath notation that works in all modern browsers. It was designed with the goal of consolidating the recent advances in web technologies into a single, definitive, math-on-the-web platform supporting the major browsers and operating systems, including those on mobile devices. It requires no setup on the part of the user (no plugins to download or software to install), so the page author can write web documents that include mathematics and be confident that users will be able to view it naturally and easily. One simply includes MathJax and some mathematics in a web page, and MathJax does the rest.

MathJax uses web-based fonts to produce high-quality typesetting that scales and prints at full resolution, unlike mathematics included as bitmapped images. With MathJax, mathematics is text-based rather than image-based, and so it is available for search engines, meaning that your equations can be searchable, just like the text of your pages. MathJax allows page authors to write formulas using TeX and LaTeX notation, MathML (a World Wide Web Consortium standard for representing mathematics in XML format), or AsciiMath notation. MathJax can generate output in several formats, including HTML with CSS styling, or scalable vector graphics (SVG) images.

MathJax includes the ability to generate speakable text versions of your mathematical expressions that can be used with screen readers, providing accessibility for the visually impaired. The assistive support in MathJax also includes an interactive expression explorer that helps these users to "walk through" an expression one piece at a time, rather than having to listen to a complex expression all at once, and the ability to "collapse" portions of the expressions to allow a more simplified expression to be read, and only expanded if more detail is desired.

MathJax is modular, so it can load components only when necessary, and can be extended to include new capabilities as needed. MathJax is highly configurable, allowing authors to customize it for the special requirements of their web sites. Unlike earlier versions of MathJax, version 3 can be packaged into a single file, or included as part of larger bundles for those sites that manage their javascript assets in that way.

Finally, MathJax has a rich application programming interface (API) that can be used to make the mathematics on your web pages interactive and dynamic. Version 3 has been rewritten in ES6 using Typescript (a version of javascript that includes type-checking and the ability to transpile to ES5). It was designed to be used as easily on a server (as part of a `node.js` application) as it is in a browser. This makes pre-processing of web pages containing mathematics much easier than with version 2, so web sites can perform all the math processing once up front, rather than having the browser do it each time the page is viewed.

CHAPTER 2

Accessibility Features

MathJax's mission is to provide the best tools for mathematics on the web. Naturally, this means for everyone and thus accessibility is an important concern for us.

2.1 MathJax User Interface

The MathJax user interface currently consists of the *MathJax Menu* and the various MathJax messages, such as syntax error messages from the TeX input processor.

The user interface for version 2 was localized to over 20 languages and many more partial localizations thanks to the fantastic support of the community at TranslateWiki.net. Localization is not yet available in version 3, but is on the roadmap for a future version.

The MathJax Menu follows WCAG 2.0 guidelines. Each MathJax fragment is included in the tab order; the menu can be triggered via the space or menu key; and navigation in the menu is possible using the arrow keys.

2.2 MathJax Accessibility Extensions

The *MathJax Accessibility extensions* provide several tools and features that enable universal rendering of mathematics on the web. They enhance rendering both visually and aurally. In particular:

- An innovative responsive rendering of mathematical content through collapsing and exploration of subexpressions.
- An aural rendering tool providing on-the-fly speech-text for mathematical content and its subexpressions using various rule sets.
- Tactile rendering tool enabling Nemeth Braille output on a connecte Braille displays.
- An exploration tool, allowing for meaningful exploration of mathematical content including multiple highlighting features, magnification and synchronized aural rendering.

The Accessibility Extensions support the widest selection of browsers, operating systems, and assistive technologies as they only require the use of well-supported web standards such as WAI-ARIA, in particular labels and live regions.

The Accessibility Extensions can be enabled using the MathJax Contextual Menu (right-click on any typeset expression), and are loaded automatically when enabled. The contextual menu code is included in all the combined MathJax components, such as `tex-chtml` and `mml-svg`. If you are making a custom configuration, you can include `ui/menu` to enable the contextual menu, or you can include any of the *a11y extensions* explicitly.

See the *Accessibility Extensions Options* section for details about how to configure the extensions.

2.3 Screen Reader Support

Some screen readers support MathML, MathJax's internal format. Screenreaders like ChromeVox, JAWS (on IE), and TextHelp support MathJax directly (most only version 2); other screenreaders are supported by the `assistive-mml` extension as of version 3.0.1.

The `assistive-mml` extension embeds visually hidden MathML alongside MathJax's visual rendering while hiding the visual rendering from assistive technology (AT) such as screenreaders. This allows most MathML-enabled screenreaders to read out the underlying mathematics. It's important to note that Presentation MathML is usually not expressive enough to voice the mathematics properly in all circumstances, which is why screenreaders have to rely on heuristics to analyze the MathML semantically.

The quality of MathML support in screenreaders varies greatly, with different levels of MathML feature support, different speech rule sets, and different voicing technologies.

The expected result for MathJax given the current state of technology is roughly the following:

- The visually-hidden MathML is read out correctly by AT (i.e., not just the character strings but, e.g., `<mfrac>` leads to "fraction"; this will vary with the MathML support of the screenreader).
- The visual rendering is not read out by AT
- The MathJax Menu triggers AT to say "clickable" before each math element.
  - This allows keyboard users to enter the MathJax Menu via space or menu key.
- The visually hidden MathML does not get an outline (usually placed at an odd location due to the target of the outline being visually hidden).
  - except in iOS VoiceOver, where this allows the user to hook into VoiceOver's exploration features.

2.4 More Information

2.4.1 Accessibility Extension

MathJax offers accessibility support via its own built-in extension that provides a choice of support options as well as a high degree of personalization. The extension can be activated either via the context menu, which itself is fully accessible, or by default using configuration options. Similarly its various features and options are best selected via the *MathJax Menu* or programmatically using the *accessibility options*. We discuss the different features of the accessibility tool at the hand of the context menu, roughly in the order in which they appear.

Most features of the Accessibility extensions are based on technology provided by the Speech Rule Engine. For some more details and information please also see there.

MathJax's supports the widest selection of browsers, operating systems, and assistive technologies as they only require the use of well-supported web standards such as WAI-ARIA, in particular labels and live regions.

Interactive Exploration

The main feature is an interactive exploration mode that allows a reader to traverse and explore sub-expressions step-by-step. The explorer is activated in the context menu by checking the *Activate* item in the *Accessibility* sub-menu.

Once a math expression is focused, the explorer can be started by pressing the Enter key. The cursor keys then allow traversal of the expression.

Keyboard Explorer Commands

The keyboard explorer is used to interact with a mathematical expression using keyboard commands. Interaction allows a reader to traverse an expression in a mathematical meaningful way, examining sub-expressions and diving into details as they see fit.

The keyboard explorer supports multiple types of output: Speech and Braille output for the sub-expression that is explored, magnification of that sub-expression, and synchronised highlighting with the navigation.

Navigation can be started when a MathJax expression is focused and quit at any time during the exploration. When navigation is restarted, the application will continue where the user has left off within the expression.

Overview of key bindings

Essential Keys

An earcon is played as indicator that the boundary of an expression has been reached in either direction.

Advanced Options

Special key combinations for navigating tables

Special Notes

Note: Depending on the implementation quality of the particular browser/screenreader/OS combination (especially Chrome and IE), users might have to disable screenreader reading modes (e.g., "browse mode" in NVDA, "virtual cursor" in JAWS) before being able to launch the MathJax explorer application.

During traversal, focused sub-expressions are highlighted and optionally magnified. In addition, an aural rendering is pushed to a screen reader, if one is available, and a tactile rendering can be read on a Braille display, if one is connected.

Speech & Braille Support

Both aural and tactile rendering can be controlled via the options in the *Speech* sub-menu. *Speech Output* and *Braille Output*, respectively, control whether or not speech or Braille output is generated. If speech is generated, it is by default also displayed in *Speech Subtitles*, which can be switched off and hidden. Braille on the other hand is by default hidden but can be displayed by switching on the *Braille Subtitles*.

Speech is generally generated with respect to the currently chosen locale (if it is available). In addition, there are a number of different rule sets that can be chosen for translating math to text, where each can have a number of different preferences for how a particular expression is spoken. By default, MathJax uses the *MathSpeak* rule set in *Verbose*

mode; however, the menu allows this to be changed to either the *ClearSpeak* or *ChromeVox*. Each rule set has several different preference settings; three in the case of MathSpeak, for example, which primarily influence the length of produced text. ClearSpeak on the other hand has a large number of preferences that allow very fine-tuned control over how different types of expressions are spoken. The MathJax menu allows a smart choice of preferences by only displaying the preferences that are currently relevant for the sub-expression that is currently explored. The *Select Preferences* option opens a selection box for all possible ClearSpeak preference choices.

Some rule-set and preference settings can also be controlled by keyboard commands. This allows the user to have the same expression read in different variants without having to leave the exploration mode. The > key switches rule sets between MathSpeak and ClearSpeak if both are available for the current locale. The < key cycles preferences for the currently active rule set. For ClearSpeak rules, preference cycling depends on the type of the currently explored sub-expression, similar to smart selection of menu entries.

The speech language can be adjusted in the *Language* sub-menu in the *Speech* options. MathJax currently only supports speech in English, French, German, and Spanish. The only available Braille output is Nemeth. We are hoping to add more in the future.

In addition to voicing expressions, the explorer allows for queries on sub-expression, such as getting positional information with respect to the context, as well as summaries of the sub-expression currently explored.

Abstraction

In addition to textual summaries of expressions, MathJax offers the possibility to abstract certain sub-expressions so that the entire sub-expression is visually replaced by a placeholder symbol and interactive traversal treats it as a single element. This allows the reader to abstract away details and to better observe the overall structure of a formula.

Abstraction can be triggered either via mouse click on a collapsible expression or via pressing the Enter key during keyboard exploration. Expressions that can be abstracted can also be discovered using some of the highlighting features.

Highlight

During interactive exploration, the sub-expression that is explorered is automatically highlighted, by default with a blue background color. The highlighting can be customized by changing *Background* or *Foreground* colors in in the *Highlight* sub-menu of the MathJax contextual menu. In addition, the opacity of both *Background* and *Foreground* can be adjusted by two slider bars underneath the respective sub-menus.

The *Highlight* sub-menu also provides a choice of highlighters for marking collapsible sub-expressions: The *Flame* highlighter permanently colors collapsible sub-expressions while successively darkening the background for nested collapsible expressions. The *Hover* highlighter colors each collapsible sub-expression only when hovering over it with the mouse pointer.

A final highlighting feature is *Tree Coloring*, in which expressions are visually distinguished by giving neighbouring symbols different, ideally contrasting foreground colors.

Magnification

During exploration, the accessibility extension can optionally magnify the sub-expression that is currently explored. The zoomed version of the expression is overlayed on the original one when traversing the formula. For keyboard exploration, this can be switched on in the *Magnification* sub-menu by selecting the *Keyboard* option.

A similar effect can be achieved by exploring an expression with the mouse. When using the *Mouse* option in the *Magnification* sub-menu, the sub-expression over which the mouse pointer hovers is zoomed.

The zoom factor of the magnification can also be adjusted. The values available in the context menu are *200%*, *300%*, *400%*, and *500%*.

Semantic Info

The *Semantic Info* sub-menu contains a number of options that allow the reader to see the semantic classifications MathJax applies to a particular sub-expression, by hovering over it with the mouse pointer. The choices here are

- *Type* is an immutable property of an expression that is independent of its particular position in a formula. Note, however that types can change depending on the subject area of a document.
- *Role* is dependent on the context of a sub-expression in the overall expression.
- *Prefix* is information pertaining to the position of a sub-expression. Examples are `'exponent'`, `'radicand'`, etc. These would also be spoken during interactive exploration.

For more details on all of these concepts, see also the documentation of the Speech Rule Engine.

2.4.2 Legacy Assistive Support in v2

Interactions between screen readers and MathJax are delicate and vary from browser to broswer, operating system to operating system, and screen reader to screen reader. The following information was gathered over time for version 2 of MathJax and various broser/operating-system/screen-reader combinations. The information is several years old, and my no longer be completely accurate, as features in browsers and screen readers change regularly. Because this inforamtion changes regularly with updates to browsers and screen readers, we are unable to maintin a table like this for version 3.

Support Matrix (AssistiveMML.js)

Below is a summary of results for MathML enabled screenreaders and the legacy AssistiveMML extension, based on tests as well as user reports.

Notes on Apple VoiceOver

- **VoiceOver** on OSX
  - *Safari*. The visually-hidden MathML is read out and gets an outline. Visual rendering is ignored correctly. VoiceOver somtimes drops parts of the equation due to its partial MathML support.
  - *Chrome*. The visually-hidden MathML is detected but VoiceOver does not read it correctly (only e.g., "4 items detected; math"; this seems like a VO bug); an outline is added. Visual rendering is ignored correctly.
  - *Firefox*. The visually-hidden MathML is only read as a string of contained characters; an outline is added. Visual rendering is ignored correctly.
- **VoiceOver** on iOS
  - The "slide two fingers from top to read screen" method will read the visually-hidden MathML. Visual rendering is ignored correctly.
  - Manual exploration.
    - Exploration by swiping left/right will read the visually-hidden MathML. Visual rendering is ignored correctly.
    - Tapping on an equation does not work due to the visually-hidden MathML being placed in a 1px box.

Notes on MathPlayer 4 and Internet Explorer 11

Design Science suggests that you always use IE's Enterprise mode for MathPlayer in IE11, see their documentation. However, it seems that this is only required for MathPlayer's visual rendering to work and this additionally requires the MathPlayer BrowserHelperAddon to be active in IE.

Unfortunately, the MathPlayer BrowserHelperAddon can lead to crashes. E.g., if you switch MathJax's output to the NativeMML output, MathPlayer will crash IE11; you'll have to clear the MathJax cookie to reset things. Also, in a plain MathML sample (without MathJax), clicking on the MathPlayer rendering will crash IE11.

Using IE's Enterprise mode should work with NVDA and the AssistiveMML extension but they don't seem to work with NVDA and plain MathML pages.

We suggest you do not switch on IE's Enterprise mode on pages using MathJax and we also have to strongly suggest that you **not** use the BrowserHelperAddon with MathJax on IE11.

CHAPTER 3

Writing Mathematics for MathJax

3.1 Putting mathematics in a web page

To put mathematics in your web page, you can use TeX and LaTeX notation, MathML notation, AsciiMath notation, or a combination of all three within the same page; the MathJax configuration tells MathJax which you want to use, and how you plan to indicate the mathematics when you are using TeX/LaTeX or AsciiMath notation. These three formats are described in more detail below.

3.1.1 TeX and LaTeX input

Mathematics that is written in TeX or LaTeX format is indicated using *math delimiters* that surround the mathematics, telling MathJax what part of your page represents mathematics and what is normal text. There are two types of equations: ones that occur within a paragraph (in-line mathematics), and larger equations that appear separated from the rest of the text on lines by themselves (displayed mathematics).

The default math delimiters are `$$...$$` and `\[...\]` for displayed mathematics, and `\(...\)` for in-line mathematics. Note in particular that the `$...$` in-line delimiters are **not** used by default. That is because dollar signs appear too often in non-mathematical settings, which could cause some text to be treated as mathematics unexpectedly. For example, with single-dollar delimiters, "... the cost is $2.50 for the first one, and $2.00 for each additional one ..." would cause the phrase "2.50 for the first one, and" to be treated as mathematics since it falls between dollar signs. See the section on *TeX and LaTeX Math Delimiters* for more information on using dollar signs as delimiters.

Here is a complete sample page containing TeX mathematics (see the MathJax Web Demos Repository for more).

```
<!DOCTYPE html>
<html>
<head>
<title>MathJax TeX Test Page</title>
<script src="https://polyfill.io/v3/polyfill.min.js?features=es6"></script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js">
</script>
```

(continues on next page)

(continued from previous page)

```
</head>
<body>
When \(a \ne 0\), there are two solutions to \(ax^2 + bx + c = 0\) and they are
$$x = {-b \pm \sqrt{b^2-4ac} \over 2a}.$$
</body>
</html>
```

Since the TeX notation is part of the text of the page, there are some caveats that you must keep in mind when you enter your mathematics. In particular, you need to be careful about the use of less-than signs, since those are what the browser uses to indicate the start of a tag in HTML. Putting a space on both sides of the less-than sign should be sufficient, but see *TeX and LaTeX support* for more details.

If you are using MathJax within a blog, wiki, or other content management system, the markup language used by that system may interfere with the TeX notation used by MathJax. For example, if your blog uses Markdown notation for authoring your pages, the underscores used by TeX to indicate subscripts may be confused with the use of underscores by Markdown to indicate italics, and the two uses may prevent your mathematics from being displayed. See *TeX and LaTeX support* for some suggestions about how to deal with the problem.

There are a number of extensions for the TeX input processor that are loaded by combined components that include the TeX input format (e.g., `tex-chtml.js`), and others that are loaded automatically when needed. See *TeX and LaTeX Extensions* for details on TeX extensions that are available.

3.1.2 MathML input

For mathematics written in MathML notation, you mark your mathematics using standard `<math>` tags, where `<math display="block">` represents displayed mathematics and `<math display="inline">` or just `<math>` represents in-line mathematics.

MathML notation will work with MathJax in HTML files, not just XHTML files, even in older browsers and that the web page need not be served with any special MIME-type. Note, however, that in HTML (as opposed to XHTML), you should **not** include a namespace prefix for your `<math>` tags; for example, you should not use `<m:math>` except in an XHTML file where you have tied the `m` namespace to the MathML DTD by adding the `xmlns:m="http://www.w3.org/1998/Math/MathML"` attribute to your file's `<html>` tag.

In order to make your MathML work in the widest range of situations, it is recommended that you include the `xmlns="http://www.w3.org/1998/Math/MathML"` attribute on all `<math>` tags in your document (and this is preferred to the use of a namespace prefix like `m:` above, since those are deprecated in HTML5), although this is not strictly required.

Here is a complete sample page containing MathML mathematics (see the MathJax Web Demos Repository for more).

```
<!DOCTYPE html>
<html>
<head>
<title>MathJax MathML Test Page</title>
<script src="https://polyfill.io/v3/polyfill.min.js?features=es6"></script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/mml-chtml.js">
</script>
</head>
<body>

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

(continues on next page)

(continued from previous page)

```
  <mi>a</mi><mo>&#x2260;</mo><mn>0</mn>
</math>,
there are two solutions to
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mi>a</mi><msup><mi>x</mi><mn>2</mn></msup>
  <mo>+</mo> <mi>b</mi><mi>x</mi>
  <mo>+</mo> <mi>c</mi> <mo>=</mo> <mn>0</mn>
</math>
and they are
<math xmlns="http://www.w3.org/1998/Math/MathML" display="block">
  <mi>x</mi> <mo>=</mo>
  <mrow>
    <mfrac>
      <mrow>
        <mo>&#x2212;</mo>
        <mi>b</mi>
        <mo>&#x00B1;</mo>
        <msqrt>
          <msup><mi>b</mi><mn>2</mn></msup>
          <mo>&#x2212;</mo>
          <mn>4</mn><mi>a</mi><mi>c</mi>
        </msqrt>
      </mrow>
      <mrow>
        <mn>2</mn><mi>a</mi>
      </mrow>
    </mfrac>
  </mrow>
  <mtext>.</mtext>
</math>
</p>

</body>
</html>
```

When entering MathML notation in an HTML page (rather than an XHTML page), you should **not** use self-closing tags, as these are not part of HTML, but should use explicit open and close tags for all your math elements. For example, you should use

```
<mspace width="5pt"></mspace>
```

rather than `<mspace width="5pt" />` in an HTML document. If you use the self-closing form, some browsers will not build the math tree properly, and MathJax will receive a damaged math structure, which will not be rendered as the original notation would have been. Typically, this will cause parts of your expression to not be displayed. Unfortunately, there is nothing MathJax can do about that, since the browser has incorrectly interpreted the tags long before MathJax has a chance to work with them.

See the *MathML* page for more on MathJax's MathML support.

3.1.3 AsciiMath input

MathJax v2.0 introduced a new input format, AsciiMath notation, by incorporating ASCIIMathML. This input processor has not been fully ported to MathJax version 3 yet, but there is a version of it that uses the legacy version 2 code to patch it into MathJax version 3. None of the combined components currently include it, so you would need to specify it explicitly in your MathJax configuration in order to use it. See the *AsciiMath* page for more details.

By default, you mark mathematical expressions written in AsciiMath by surrounding them in "back-ticks", i.e., ` . . . `.

Here is a complete sample page containing AsciiMath notation:

```
<!DOCTYPE html>
<html>
<head>
<title>MathJax AsciiMath Test Page</title>
<script>
MathJax = {
  loader: {load: ['input/asciimath', 'output/chtml']}
}
</script>
<script src="https://polyfill.io/v3/polyfill.min.js?features=es6"></script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/startup.js">
</script>
<body>

<p>When `a != 0`, there are two solutions to `ax^2 + bx + c = 0` and
they are</p>
<p style="text-align:center">
  `x = (-b +- sqrt(b^2-4ac))/(2a) .`
</p>

</body>
</html>
```

See the *AsciiMath support* page for more on MathJax's AsciiMath support and how to configure it.

3.2 Putting Math in Javascript Strings

If your are using javascript to process mathematics, and need to put a TeX or LaTeX expression in a string literal, you need to be aware that javascript uses the backslash (\) as a special character in strings. Since TeX uses the backslash to indicate a macro name, you often need backslashes in your javascript strings. In order to achieve this, you must double all the backslashes that you want to have as part of your javascript string. For example,

```
var math = '\\frac{1}{\\sqrt{x^2 + 1}}';
```

This can be particularly confusing when you are using the LaTeX macro \\, which must both be doubled, as \\\\. So you would do

```
var array = '\\begin{array}{cc} a & b \\\\ c & d \\end{array}';
```

to produce an array with two rows.

CHAPTER 4

The MathJax Community

If you are an active MathJax user, you may wish to become involved in the wider community of MathJax users. The MathJax project maintains forums where users can ask questions about how to use MathJax, make suggestions about future features for MathJax, and present their own solutions to problems that they have faced. There is also a bug-tracking system where you can report errors that you have found with MathJax in your environment.

4.1 Mailing Lists

If you need help using MathJax or you have solutions you want to share, please post to the MathJax Users Google Group. We try hard to answer questions quickly, and users are welcome to help with that as well. Also, users can post code snippets showing how they have used MathJax, so it may be a good place to find the examples you are looking for.

If you want to discuss MathJax development, please use the MathJax Dev Google Group. We made this group to discuss anything beyond what an end-user might be interested in, so if you have any suggestions or questions about MathJax performance, technology, or design, feel free to submit it to the group.

The community is only as good as the users who participate, so if you have something to offer, please take time to make a post on one of our groups.

4.2 Issue tracking

Found a bug or want to suggest an improvement? Post it to our issue tracker. We monitor the tracker closely, and work hard to respond to problems quickly.

Before you create a new issue, however, please search the issues to see if it has already been reported. You could also be using an outdated version of MathJax, so be sure to *upgrade your copy* to verify that the problem persists in the latest version.

See the section on *Reporting Issues* for more detailed instructions.

4.3 Documentation

The source for this documentation can be found on github. You can file bug reports on the documentation's bug tracker and actively contribute to the public documentation wiki.

4.4 "Powered by MathJax"

If you are using MathJax and want to show your support, please consider using our *"Powered by MathJax" badge*.

CHAPTER 5

Reporting Issues

If you come across a problem with MathJax, please report it so that the development team and other users are aware and can look into it. It is important that you report your problem following the steps outlined here because this will help us to rapidly establish the nature of the problem and work towards a solution effectively.

To report a problem, please follow these steps:

- Have you cleared your browser cache, quit your browser, and restarted it? If not, please do so first and check if the problem persists. These instructions tell you how to clear your cache on the major browsers.
- Have you turned off other extensions and plugins in your browser, and restarted it?
- Have a look at the math rendering examples on www.mathjax.org to see if you experience problems there as well. This might help you to determine the nature of your problem.
- If possible, check whether the problem has been solved in the latest MathJax release.
- Search through the MathJax User Group and the MathJax issue tracker to see if anyone else has come across the problem before.
- Found a real and new problem? Please report it to the MathJax issue tracker including the following information:
  - A detailed description of the problem. What exactly is not working as you expected? What do you see?
  - The MathJax version you are working with, your operating system, and full browser information including all version information.
  - If at all possible, a pointer to a webpage that is publicly available and exhibits the problem. This makes sure that we can reproduce the problem and test possible solutions. You can create minimal examples using such tools as jsfiddle, jsbin, codepen, or codesandbox.

CHAPTER 6

Getting Started with MathJax Components

MathJax allows you to include mathematics in your web pages, either using LaTeX, MathML, or AsciiMath notation, and the mathematics will be processed using JavaScript to produce HTML or SVG for viewing in any modern browser.

6.1 MathJax Components

To make using MathJax easier in web pages, the various pieces that make up MathJax have been packaged into separate files called "components". For example, there is a component for MathML input, and one for SVG output, and the various TeX extensions are packaged as separate components. You can mix and match the various components to customize MathJax to suit your particular needs (this is described in detail in the section on *Configuring MathJax* below); the individual component files that you specify are loaded when MathJax starts up.

There are also components that combine several others into one larger file that loads everything you need to run MathJax all at once. These represent some of the standard combinations of input and output formats, and you will probably find one of these that suits your needs. You can *configure* the various components in order to customize how they run, even when they are loaded as part of a combined component. For example, you can set the delimiters to be used for in-line and displayed math for the TeX input component whether the TeX component was loaded individually, or as part of the `tex-chtml` component.

It is even possible for you to create your own components or custom builds of MathJax, or incorporate the MathJax components into larger files that contain other assets your website might need (see the section on *Making a Custom Build of MathJax* for more details).

6.2 Ways of Accessing MathJax

There are two ways to access MathJax for inclusion in web pages: link to a content delivery network (CDN) like `cdn.jsdelivr.net` to obtain a copy of MathJax, or download and install a copy of MathJax on your own server (for network access) or hard disk (for local use without a network connection). The first method is described below, while the second is discussed in the section on *Hosting Your Own Copy of MathJax*.

This page gives the quickest and easiest ways to get MathJax up and running on your web site, but you may want to read the details in the linked sections in order to customize the setup for your pages.

6.2.1 Using MathJax from a Content Delivery Network (CDN)

The easiest way to use MathJax is to link directly to a public installation available through a Content Distribution Network (CDN). When you use a CDN, there is no need to install MathJax yourself, and you can begin using MathJax right away. The CDN will automatically arrange for your readers to download MathJax files from a fast, nearby server.

To use MathJax from a CDN, you need to do three things:

1. Include a MathJax configuration in your page (this may be optional in some cases).
2. Link to MathJax in the web pages that are to include mathematics.
3. Put mathematics into your web pages so that MathJax can display it.

There are many free CDN services that provide copies of MathJax. Most of them require you to specify a particular version of MathJax to load, but some provide "rolling releases", i.e., links that update to the latest available version upon release (note that we also provide a means of obtaining the latest version automatically, described below).

- jsdelivr.com [latest or specific version] (recommended)
- unpkg.com [latest or specific version]
- cdnjs.com
- raw.githack.com
- gitcdn.xyz
- cdn.statically.io

To jump start using `jsdelivr`, you accomplish the first two steps by putting

```
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-chtml.js">
</script>
```

into the `<head>` block of your document. (It can also go in the `<body>` if necessary, but the head is to be preferred.) This will load the latest 3.x.x version of MathJax from the distributed server, configure it to recognize mathematics in both TeX and MathML notation, and ask it to generate its output using HTML with CSS (the CommonHTML output format) to display the mathematics.

> **Warning:** The `tex-mml-chtml.js` file includes all the pieces needed for MathJax to process these two input formats and produce this output format. There are several other choices with different input/output combinations, and and you can even configure MathJax to load components individually.
>
> We list this file here because it will get you started quickly with MathJax without having to worry too much about configurations; but since it is one of the most general of the combined component files, it is also one of the largest, so you might want to consider a smaller one that is more tailored to your needs. See the section on *Configuring and Loading MathJax* for more details on how this is done, and on *The MathJax Components* for information about the components themselves.

If you use the code snippet given above, you will not need to change the URL whenever MathJax is updated and the version changes, because `jsdelivr` offers the `mathjax@3` notation for obtaining the `tex-mml-chtml.js` file from the latest version (3.x.x) available on the CDN.

6.2.2 Getting the Latest Version

Although `jsdelivr` provides a means of getting the latest version automatically, as described above, not all CDNs have a mechanism for that. For such CDNs, MathJax provides a `latest.js` file that can be used to obtain the latest (3.x.x) version of MathJax. For example, `cdnjs` doesn't have a mechanism for getting the latest 3.x.x version automatically, so you can use

```
<script type="text/javascript" id="MathJax-script" async
  src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.0.0/es5/latest?tex-mml-chtml.
↪js">
</script>
```

to obtain the latest (3.x.x) version of the `tex-mml-chtml` component from `cdnjs`; even though you have started by asking for version 3.0.0, the `latest.js` script will switch to the latest 3.x.x version automatically.

6.2.3 Getting a Specific Version

It is also possible to always use a specific version, regardless of the current version of MathJax. To do this, simply give the full version number in the URL; for example:

```
<script id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3.0.0/es5/tex-mml-chtml.js">
</script>
```

will always load version 3.0.0 of the `tex-mml-chtml.js` combined component file.

Other CDNs have slightly different formats for how to specify the version number. For example, `cdnjs` uses the following:

```
<script type="text/javascript" id="MathJax-script" async
  src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.0.0/es5/tex-mml-chtml.js">
</script>
```

to get the same file.

6.2.4 Browser Compatibility

MathJax supports all modern browsers (Chrome, Safari, Firefox, Edge), and most mobile browsers. Include the polyfill library in order to support earlier browser versions (see their browser support page for details). In particular, to allow MathJax version 3 to work with IE11, include the line

```
<script src="https://polyfill.io/v3/polyfill.min.js?features=es6"></script>
```

before the script that loads MathJax.

6.3 Configuring MathJax

The combined component files, like `tex-mml-chtml.js`, include default settings for the various options available in MathJax. You may need to adjust those to suit your needs. For example, the TeX input component does not enable single dollar signs as delimiters for in-line mathematics because single dollar signs appear frequently in normal text,

e.g. "The price is $50 for the first one, and $40 for each additional one", and it would be confusing the have "50 for the first one, and" be typeset as mathematics.

If you wish to enable single dollar signs as in-line math delimiters, you need to tell MathJax that by providing an explicit MathJax configuration. That is accomplished by using a `<script>` tag to set the `MathJax` global variable to hold a configuration for MathJax and placing that script before the one that loads the MathJax component file that you are using. For example

```
<script>
MathJax = {
  tex: {
    inlineMath: [['$', '$'], ['\\(', '\\)']]
  }
};
</script>
<script id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js">
</script>
```

configures MathJax's TeX input component to use `$...$` and `\(...\)` as delimiters for inline-math (this enabling single dollar signs as math delimiters), and then loads the `tex-chtml.js` component for TeX input and CommonHTML output.

There are many options that can be set in this way. See the section on *Configuring and Loading MathJax* for more details, and on *Configuring MathJax* for information on the available options for the various components.

6.4 Putting Mathematics in a Web Page

Once MathJax is configured and loaded, it will look through your web page for mathematics for it to process. There are three available formats for that mathematics: TeX/LaTeX, MathML, and AsciiMath. The TeX/LaTeX and AsciiMath formats are plain text formats that use special delimiter characters to separate the mathematics from the rest of the text of your document, while the MathML format is an XML format that uses "tags" (similar to HTML tags) to represent the mathematics. TeX and AsciiMath are often written by hand, but MathML usually is generated by mathematical software or specialized editors.

See the section on *Writing Mathematics for MathJax* for more details about how to enter mathematics in these three formats.

Note that once MathJax has processed the page, it will not run again without you explicitly telling it to. For example, if you add new mathematics to the page after MathJax has already run, that math will not be processed by MathJax until you request that to happen. See the section on *MathJax in Dynamic Content* for details of how to do that.

6.5 Where to Go from Here?

If you have followed the instructions above, you should now have MathJax installed and configured on your web server, and you should be able to use it to write web pages that include mathematics. At this point, you can start making pages that contain mathematical content!

You could also read more about the details of how to *customize MathJax*.

You can also check out the *MathJax examples* for illustrations of using MathJax.

If you are working on dynamic pages that include mathematics, you might want to read about the *MathJax Application Programming Interface* (its API), so you know how to include mathematics in your interactive pages.

Finally, if you have questions or comments, or want to help support MathJax, you could visit the *MathJax community forums* or the *MathJax bug tracker*.

CHAPTER 7

Configuring and Loading MathJax

The configuration, loading, and startup processes for MathJax version 3 are different from those of version 2 in a number of ways. Where version 2 had several different methods for configuring MathJax, version 3 streamlines the process and has only one, as described below. In version 2, you always loaded `MathJax.js`, and added a `config=...` parameter to provide a combined configuration file, but in version 3 you load one of several different files, depending on your needs (so you can avoid multiple file transfers, and also use MathJax synchronously, which was not possible in version 2).

If you use one of the combined component files in version 3, like `mml-chtml`, you may not need to do any configuration at all.

7.1 Configuring MathJax

To configure MathJax, you use a global object named *MathJax* that contains configuration data for the various components of MathJax. For example, to configure the TeX input component to use single dollar signs as in-line math delimiters (in addition to the usual `\(...\)` delimiters) and the SVG output component to use a global font cache for all expressions on the page, you would use

```
MathJax = {
  tex: {
    inlineMath: [['$', '$'], ['\\(', '\\)']]
  },
  svg: {
    fontCache: 'global'
  }
};
```

The sections below describe the different places you could put such a configuration. For information on the options that you can set for each of the components, see the *Configuring MathJax* pages.

7.1.1 Configuration Using an In-Line Script

The easiest way to configure MathJax is to place the `MathJax` object in a `<script>` tag just before the script that loads MathJax itself. For example:

```
<script>
MathJax = {
  tex: {
    inlineMath: [['$', '$'], ['\\(', '\\)']]
  },
  svg: {
    fontCache: 'global'
  }
};
</script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-svg.js">
</script>
```

This will configure the TeX input component to use single dollar signs as in-line math delimiters, and the SVG output component to use a global font cache (rather than a separate cache for each expression on the page), and then loads the latest version of the `tex-svg` component file from the `jsdelivr` CDN. This will typeset any TeX mathematics on the page, producing SVG versions of the expressions.

7.1.2 Using a Local File for Configuration

If you are using the same MathJax configuration over multiple pages, you may find it convenient to store your configuration in a separate JavaScript file that you load into the page. For example, you could create a file called `mathjax-config.js` that contains

```
window.MathJax = {
  tex: {
    inlineMath: [['$', '$'], ['\\(', '\\)']]
  },
  svg: {
    fontCache: 'global'
  }
};
```

and then use

```
<script src="mathjax-config.js" defer></script>
<script type="text/javascript" id="MathJax-script" defer
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-svg.js">
</script>
```

to first load your configuration file, and then load the `tex-svg` component from the `jsdelivr` CDN.

Note that here we use the `defer` attribute on both scripts so that they will execute in order, but still not block the rest of the page while the files are being downloaded to the browser. If the `async` attribute were used, there is no guarantee that the configuration would run first, and so you could get instances where MathJax doesn't get properly configured, and they would seem to occur randomly.

7.1.3 Configuring and Loading in One Script

It is possible to have the MathJax configuration file also load MathJax as well, which would be another way to handle the problem of synchronizing the two scripts described above. For example, you could make the file `load-mathjax.js` containing

```
window.MathJax = {
  tex: {
    inlineMath: [['$', '$'], ['\\(', '\\)']]
  },
  svg: {
    fontCache: 'global'
  }
};

(function () {
  var script = document.createElement('script');
  script.src = 'https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-svg.js';
  script.async = true;
  document.head.appendChild(script);
})();
```

and then simply link to that file via

```
<script src="load-mathjax.js" async></script>
```

This script can be `async` because it doesn't have to synchronize with any other script. This will allow it to run as soon as it loads (since it is small, there is little cost to that), meaning the script to load MathJax itself will be inserted as soon as possible, so that MathJax can begin downloading as early as possible. (If this script were loaded with `defer`, it would not run until the page was ready, so the script to load MathJax would not be inserted until then, and you would have to wait for MathJax to be downloaded before it could run.)

7.1.4 Converting Your v2 Configuration to v3

Because the version 3 configuration options are somewhat different from their version 2 counterparts, we provide an automated configuration conversion tool to help you move from version 2 to version 3. Simply paste your current `MathJax.Hub.Config()` call into the converter, press `Convert` and you should get the equivalent version 3 configuration, and comments about any options that could not be translated to version 3 (some options are not yet implements, others no longer make sense in version 3). See the instructions on the linked page for more details.

7.2 Loading MathJax

Once you have configured MathJax, you then load the MathJax component file that you want to use. Most often, this will mean you load a combined component that loads everything you need to run MathJax with a particular input and output format. For example, the `tex-svg` component would allow you to process TeX input and produce SVG output. To do so, use a script like the following

```
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-svg.js">
</script>
```

to get the latest (3.x.x) version of the `tex-svg` component in ES5 format (the only one currently available) from the `jsdelivr` CDN. This takes advantage of the feature of `jsdeliver` that allows you to get the latest version using the `mathjax@3` notation. For a specific version, you would use

```
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3.0.0/es5/tex-svg.js">
</script>
```

to always get the 3.0.0 version of the `tex-svg` component.

Other CDNs have slightly different formats for how to specify the version number. For example, `cdnjs` uses the following:

```
<script type="text/javascript" id="MathJax-script" async
  src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.0.0/es5/tex-svg.js">
</script>
```

Some CDNs don't provide a means of getting the lastest version automatically. For these, MathJax provides a `latest.js` file that will do that for you. For example, `cdnjs` doesn't have a mechanism for getting the latest 3.x.x version automtically. If you want to do that using `cdnjs`, then use

```
<script type="text/javascript" id="MathJax-script" async
  src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.0.0/es5/latest?tex-svg.js">
</script>
```

to obtain the latest (3.x.x) version of the `tex-svg` component.

See *The MathJax Components* for a list of the various components you can choose and descriptions of their contents. See the *list of CDNs* for the URLs for a number of CDNs that serve MathJax.

Note that the script that loads the MathJax component file should *follow* the script the configures MathJax (otherwise MathJax will not know what configuration you need). If you use one of the combined component files in version 3, you may not need to do any configuration at all.

7.2.1 Loading Components Individually

If none of the combined component files suits your needs, you can specify the individual components you want by setting the `load` array in the `loader` section of your MathJax configuration and loading the `startup` component.

For example

```
<script>
MathJax = {
  loader: {
    load: ['input/tex-base', 'output/svg', 'ui/menu', '[tex]/require']
  },
  tex: {
    packages: ['base', 'require']
  }
};
</script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3.0.0/es5/startup.js">
</script>
```

would cause the base TeX input, the SVG output, the contextual menu code, and the TeX `\require` macro extension components to be loaded (and would tell TeX to use the `require` extension in addition to the base TeX macros).

In this way, you can load exactly the components you want. Note, however, that each component will be loaded as a separate file, so it is better to use a combined component file if possible.

7.2.2 Loading Additional Components

You can use the `load` array described in the previous section to load additional components even if you are using one of the combined components. For example

```
<script>
MathJax = {
  loader: {
    load: ['[tex]/colorv2']
  },
  tex: {
    packages: {'[+]': 'colorv2'},
    autoload: {color: []}
  }
};
</script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3.0.0/es5/tex-chtml.js">
</script>
```

would load the version-2-compatible `\color` macro, inform TeX to add that to the packages that it already has loaded, and not autoload the default version 3 `color` (the LaTeX-compatible one). This is done on top of the `tex-chtml` combined configuration file, so the TeX input and CommonHTML output formats are already included (as are the contextual menu, and several TeX packages; see *The MathJax Components* for details).

7.3 Performing Actions During Startup

MathJax allows you several ways to hook into the MathJax startup process so that you can do additional configuration, perform actions after the initial typesetting, and so on. Because MathJax version 3 uses *promises* for its synchronization, they are what MathJax provides in order for you to hook into the startup process. There are two main hooks that you can set in the `startup` block of your configuration: the `ready()` function and the `pageReady()` function.

The `ready()` function is what MathJax calls when all the components of MathJax have been loaded. It builds the internal structures needed by MathJax, creates functions in the `MathJax` object to make typesetting and format conversion easy for you, performs the initial typesetting call, and sets up a promise for when that is complete. You can override the `ready()` function with one of your own to override the startup process completely, or to perform actions before or after the usual initialization. For example, you could do additional setup before MathJax created the objects it needs, or you could hook into the typesetting promise to synchronize other actions with the completion of the initial typesetting. Examples of these are given below.

The `pageReady()` function is performed when MathJax is ready (all its components are loaded, and the internal objects have been created), and the page itself is ready (i.e., it is OK to typeset the page). The default is for `pageReady()` to perform the initial typesetting of the page, but you can override that to perform other actions instead, such as delaying the initial typesetting while other content is loaded dynamically, for example. The `ready()` function sets up the call to `pageReady()` as part of its default action.

The return value of `pageReady()` is a promise that is resolved when the initial typesetting is finished (it is the return value of the initial `MathJax.typesetPromise()` call). If you override the `pageReady()` method, your function should return a promise as well. If your function calls `MathJax.startup.defaultPageReady()`, then you should return the promise that it returns (or a promise obtained from its `then()` or `catch()` methods).

The `MathJax.startup.promise` will resolve when the promise you return is resolved; if you don't return a promise, `MathJax.startup.promise` will resolve immediately, which may mean that it resolves too early.

Using these two functions separately or in combination gives you full control over the actions that MathJax takes when it starts up, and allows you to customize MathJax's startup process to suit your needs. Several examples are given below for common situations.

7.3.1 Performing Actions During Initialization

If you want to perform actions after MathJax has loaded all the needed components, you can set the `ready()` function to a function that does the needed actions, and then calls `MathJax.startup.defaultReady()` to perform the usual startup process.

Actions coming before the `MathJax.startup.defaultReady()` call are run before any initialization has been done. In particular, this is before any input or output jax are created, so this is where customization of the MathJax object definitions could be performed. For example, you could modify the configuration blocks at this point, or you could create subclasses of the MathJax objects that override some of their methods to produce custom behavior, and then register those subclasses with MathJax so they will be used in place of the originals.

Actions coming after the `MathJax.startup.defaultReady()` call are run after initialization is complete. In particular, all the internal objects used by MathJax (e.g., the input and output jax, the math document, the DOM adaptor, etc) will have been created, and the typesetting and conversion methods will have been created in the `MathJax` object. Also the `MathJax.startup.promise` value will hold a promise that is resolved when the initial typesetting is complete, but note that the typesetting has not yet been performed at this point.

```
window.MathJax = {
  startup: {
    ready: () => {
      console.log('MathJax is loaded, but not yet initialized');
      MathJax.startup.defaultReady();
      console.log('MathJax is initialized, and the initial typeset is queued');
    }
  }
};
```

The next section shows how to use the `MathJax.startup.promise` to synchronize with the initial typesetting action.

7.3.2 Performing Actions After Typesetting

Often, you may need to wait for MathJax to finish typesetting the page before you perform some action. To accomplish this, you can override the `ready()` function, having it perform the `MathJax.startup.defaultReady()` action, and then use the `MathJax.startup.promise` to queue your actions; these will be performed after the initial typesetting is complete.

```
window.MathJax = {
  startup: {
    ready: () => {
      MathJax.startup.defaultReady();
      MathJax.startup.promise.then(() => {
        console.log('MathJax initial typesetting complete');
      });
    }
  }
};
```

As an alternative, you can override the `pageReady()` function, and use the promise returned from the `MathJax.startup.defaultPageReady()` function:

```
window.MathJax = {
  startup: {
    pageReady: () => {
      return MathJax.startup.defaultPageReady().then(() => {
        console.log('MathJax initial typesetting complete');
      });
    }
  }
};
```

Be sure that you return the promise that you obtain from `then()` method, otherwise `MathJax.startup.promise` will resolve before the initial typesetting (and your code) has been performed.

7.4 Configuring MathJax After it is Loaded

The global variable `MathJax` is used to store the configuration for MathJax. Once MathJax is loaded, however, MathJax changes the `MathJax` variable to contain the various methods needed to control MathJax. The initial configuration that you provided is moved to the `MathJax.config` property so that its contents doesn't conflict with the new values provides in `MathJax`. This occurs when the MathJax component you have requested is loaded (and before the `ready()` function is called).

Once MathJax has created the objects that it needs (like the input and output jax), changes to the configuration may not have any effect, as the configuration values were used during the creation of the objects, and that is already complete. Most objects make a copy of their configuration from your original `MathJax` object, so changing the values in `MathJax.config` after the objects are created will not change their configurations. (You can change `MathJax.config` values for objects that haven't been created yet, but not for ones that have.)

For some objects, like input and output jax, document handlers, and math documents, the local copies of the configuration settings are stored in the `options` property of the object, and you may be able to set the value there. For example, `MathJax.startup.output.options.scale` is the scaling value for the output, and you can set that at any time to affect any subsequent typeset calls.

Note that some options are moved to sub-objects when the main object is created. For example, with the TeX input jax, the `inlineMath` and similar options are used to create a `FindTeX` object that is stored at `MathJax.startup.input[0].findTeX`; but in this case, the `FindTeX` object uses the configuration once when it is created, so changing `MathJax.startup.input[0].findTeX.options` will not affect it. (There is a `getPatterns()` method if the `FindTeX` object that could be used to refresh the object if the options are changed, however.)

If you need to change the configuration for an object whose options can't be changed once it is created, then you will need to create a new version of that object after you change the configuration. For example, if you change `MathJax.config.tex.inlineMath` after MathJax has started up, that will not affect the TeX input jax, as described above. In this case, you can call `MathJax.startup.getComponents()` to ask MathJax to recreate all the internal objects (like `MathJax.startup.input`). This will cause them to be created using the new configuration options. Note, however, that MathJax will no longer know about any mathematics that has already been typeset, as that data was stored in the objects that have been discarded when the new ones are created.

CHAPTER 8

The MathJax Components

In order to make it possible to customize what parts of MathJax you include in your web pages, the MathJax code has been broken into individual pieces, called "components". These are designed to share common code, so that you don't download the same thing more than once, while still making it possible to only download the parts that you need. There are individual components for the various input and output processors in MathJax, for the various TeX extensions, for the contextual menu, and for other specialized pieces, such as the assistive technology support. These can be mixed and matched in whatever combinations you need.

There are some obvious combinations of components, for example, TeX input together with SVG output, or MathML input with CommonHTML output. MathJax provides a number of these common combinations as complete packages that contain everything you need to run mathjax in your page in a single file, though you can also configure additional extensions to be loaded as well.

Components provide a great deal of flexibility in determining the pieces of MathJax that you use. You can even make your own custom builds of MathJax that package exactly the pieces and that you want to use. See *Making a Custom Build of MathJax* for more details about how to do that.

See the *Loading MathJax* section for details about how to specify and load MathJax components.

See the *Configuring MathJax* section for details about how to configure the various MathJax components.

8.1 Combined Components

Currently there are eight combined components, whose contents are described below:

- *tex-chtml*
- *tex-chtml-full*
- *tex-svg*
- *tex-svg-full*
- *tex-mml-chtml*

- *tex-mml-svg*
- *mml-chtml*
- *mml-svg*

The combined components include everything needed to run MathJax in your web pages. Each includes at least one input processor, an output processor, the data needed for the MathJax TeX font, the contextual menu code, and the *startup* component.

Unlike the other components, these combined components should be loaded directly via a <*script*> tag, not through the `load` array in your MathJax configuration. So a typical use would be

```
<script>
MathJax = {
  // your configuration here, if needed
};
</script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js">
</script>
```

to load the *tex-chtml* component, for example.

8.1.1 tex-chtml

The *tex-chtml* component loads the *input/tex* component and the *output/chtml*, along with the contextual menu component, and the startup component.

The *input/tex* component loads the *ams*, *newcommand*, *require*, *autoload*, *configmacros*, and *noundefined* extensions, which that means most other extensions will be loaded automatically when needed, or you can use the `\require` macro to load them explicitly.

8.1.2 tex-chtml-full

The *tex-chtml-full* component loads the *input/tex-full* component and the *output/chtml*, along with the contextual menu component, and the startup component.

The *input/tex-full* component loads the the code for all the TeX extensions, and configures TeX to use all but the *physics* and *colorv2* extensions.

8.1.3 tex-svg

The *tex-svg* component loads the *input/tex* component and the *output/svg*, along with the contextual menu component, and the startup component.

The *input/tex* component loads the *ams*, *newcommand*, *require*, *autoload*, *configmacros*, and *noundefined* extensions, which that means most other extensions will be loaded automatically when needed, or you can use the `\require` macro to load them explicitly.

8.1.4 tex-svg-full

The *tex-svg-full* component loads the *input/tex-full* component and the *output/svg*, along with the contextual menu component, and the startup component.

The *input/tex-full* component loads the the code for all the TeX extensions, and configures TeX to use all but the *physics* and *colorv2* extensions.

8.1.5 tex-mml-chtml

The *tex-mml-chtml* component loads the *input/tex* and *input/mml* components and the *output/chtml*, along with the contextual menu component, and the startup component.

The *input/tex* component loads the *ams*, *newcommand*, *require*, *autoload*, *configmacros*, and *noundefined* extensions, which that means most other extensions will be loaded automatically when needed, or you can use the `\require` macro to load them explicitly.

8.1.6 tex-mml-svg

The *tex-mml-svg* component loads the *input/tex* and *input/mml* components and the *output/svg*, along with the contextual menu component, and the startup component.

The *input/tex* component loads the *ams*, *newcommand*, *require*, *autoload*, *configmacros*, and *noundefined* extensions, which that means most other extensions will be loaded automatically when needed, or you can use the `\require` macro to load them explicitly.

8.1.7 mml-chtml

The *mml-chtml* component loads the *input/mml* component and the *output/chtml*, along with the contextual menu component, and the startup component.

8.1.8 mml-svg

The *mml-svg* component loads the *input/mml* component and the *output/svg*, along with the contextual menu component, and the startup component.

8.2 Input Components

Currently there are three MathJax input formats, each packaged into its own component.

- *input/tex*
- *input/mml*
- *input/asciimath*

These are described in more detail below. See the *Input Processor Options* section for details about configuring these components.

8.2.1 input/tex

The TeX input format is packaged in three different ways, depending on which extensions are included in the component. This gives you several possible trade-offs between file size and feature completeness. See the *TeX and LaTeX input* section for details about the TeX input processor.

When you include one of the TeX input components, MathJax will define a function to convert TeX strings into the output format that has been loaded. See the *Converting a Math String to Other Formats* section for details.

input/tex

This is the standard TeX input component. It includes the main TeX/LaTeX input parser, along with the base definitions for the most common macros and environments. It also includes the *ams*, *newcommand*, *require*, *autoload*, *configmacros*, and *noundefined* extensions. The remaining extensions (other than *physics* and *colorv2*) are loaded automatically when needed, or you can use `\require` to load any of them explicitly. This will cause the extensions to be loaded dynamically, so if you are calling MathJax's typesetting or conversion methods yourself, you should use the promise-based versions in order to handle that properly.

See the *TeX Input Processor Options* section for information about configuring this component.

input/tex-full

This is the most complete TeX input component. It includes the main TeX/LaTeX input parser, along with all the TeX extensions, and is configured to enable all of them other than *physics* and *colorv2*. You can add these two to the `packages` array in the `tex` section of your MathJax configuration, though you should remove the *color* extension if you add the *colorv2* extension, and should remove the *braket* extension if you enable the *physics* package.

See the *TeX Input Processor Options* section for information about configuring this component.

input/tex-base

This is a minimal TeX input component. It includes the main TeX/LaTeX input parser, along with the base definitions for the most common macros and environments. No other extensions are included, so no extensions are autoloaded, and you can not use `\require`. For this component, you must explicitly load the extensions you want to use, and add them to the `packages` array.

See the *TeX Input Processor Options* section for information about configuring this component.

TeX Extension Packages

Each of the TeX extensions listed in the *The TeX/LaTeX Extension List* has its own component. The name of the component is the name of the extension preceded by `[tex]/`; so the component for the `enclose` extension is `[tex]/enclose`. You can include any of the extension components in the `load` array of the `loader` section of your MathJax configuration, and add the extension to the `packages` array in the `tex` block. For example:

```
window.MathJax = {
  loader: {load: ['[tex]/enclose']},
  tex: {
    packages: {'[+]': ['enclose']}
  }
};
```

Of course, if you are using one of the packages that includes the *autoload* extension, then you don't have to load the extensions explicitly (except for *physics* and *colorv2*), as they will be loaded automatically when first used.

In addition, there is a `[tex]/all-packages` component that includes all the packages, and configures the TeX input processors to include all of them except *physics* and *colorv2*. The *input/tex-base* and *[tex]/all-packages* components together are effectively the same as the *input/tex-full* component.

See the *TeX Extension Options* section for information about configuring the TeX extensions.

8.2.2 input/mml

The *input/mml* component contains the MathML input processor, including the function that identifies MathML within the page. See the *MathML input* section for details concerning the MathML input processor. When you include the *input/mml* component, MathJax will define a function to convert serialized MathML strings into the output format that has been loaded. See the *Converting a Math String to Other Formats* section for details.

- See the *MathML Support* section for details about MathML output.
- See the *MathML Input Processor Options* section for information about configuring this component.

8.2.3 input/asciimath

The *input/asciimath* component contains the AsciiMath input processor, including the function that identifies AsciiMath within the page. See *AsciiMath input* section or details concerning the AsciiMath input processor. When you include the *input/asciimath* component, MathJax will define a function to convert AsciiMath strings into the output format that has been loaded. See the *Converting a Math String to Other Formats* section for details.

See the *AsciiMath Input Processor Options* section for information about configuring this component.

Note: The AsciiMath input jax has not been fully ported to version 3 yet. The AsciiMath component includes legacy MathJax 2 code patched into the MathJax 3 framework. That makes the AsciiMath component larger than usual, and slower than the other input components.

8.3 Output Components

Currently there are two MathJax output formats, each packaged into its own component.

- *output/chtml*
- *output/svg*

These are described in more detail below.

Note: The *NativeMML* output jax from version 2 has not been ported to version 3, and is unlikely to be. See the *MathML Support* section for details.

8.3.1 output/chtml

The *output/chtml* component includes the CommonHTML output processor. When loaded, it causes data for handling the MathJax TeX font to be loaded as well (via a separate component). Currently, this is the only font available in version 3 (see the *MathJax Font Support* section for more information). The *output/chtml/fonts/tex* component holds the font data.

- See the *HTML Support* section for details on the CommonHTML output processor.
- See the *CommonHTML Output Processor Options* section for information about configuring this component.

8.3.2 output/svg

The *output/svg* component includes the SVG output processor. When loaded, it causes data for handling the MathJax TeX font to be loaded as well (via a separate component). Currently, this is the only font available in version 3 (see the *MathJax Font Support* section for more information). The *output/svg/fonts/tex* component holds the font data.

- See the *SVG Support* section for details on the CommonHTML output processor.
- See the *SVG Output Processor Options* section for information about configuring this component.

8.4 Accessibility Components

Currently, there are three components designed specifically to support assistive technology.

- *a11y/semantic-enrich*
- *a11y/complexity*
- *a11y/explorer*
- *a11y/assistive-mml*

To load one of these components, include the component name in the `load` array of the `loader` block of your MathJax configuration. For example:

```
<script>
MathJax = {
  loader: {
    load: ['a11y/semantic-enrich']
  }
}
</script>
```

to load the *semantic-enrich* extension.

Note: The *auto-collapse* extension has not yet been converted to version 3, but will be in a future release.

Note: The *assistive-menu* extension is now part of the standard *contextual menu extension*, so doesn't have to be loaded separately.

8.4.1 a11y/semantic-enrich

The *semantic-enrich* component connects MathJax with the Speech Rule Engine, which allows MathJax to generate speech strings for the mathematics that it processes. These can be attached to the output for use by screen readers, or for use with the *a11y/explorer* component described below.

See the *Semantic-Enrich Extension Options* section for information about configuring this component.

8.4.2 a11y/complexity

The *complexity* component computes a complexity measure for each element within an expression, and allows complex expressions to "collapse" to make them both shorter, and simpler to read. The collapsed portions can be expanded with a click of the mouse, or by keyboard actions when using the *a11y/explorer* extension described below.

See the *Complexity Extension Options* section for information about configuring this component.

8.4.3 a11y/explorer

The *explorer* component allows readers to explore a mathematical expression interactively. When an expression is focused (by tabbing to it, or by clicking on it), a reader can "enter" the expression by pressing shift-space on the keyboard. The arrow keys then move the reader through the expression (down moves to more detail by selecting the first subexpression of the selected expression, up moves to more complete expressions, while left and right move through the sub-expressions at the current level). See the *Accessibility Features* section for more details about using the expression explorer and its various features.

See the *Explorer Extension Options* section for information about configuring this component.

8.4.4 a11y/assistive-mml

The *assistive-mml* component embeds visually hidden MathML alongside MathJax's visual rendering while hiding the visual rendering from assistive technology (AT) such as screenreaders. This allows most MathML-enabled screenreaders to read out the underlying mathematics. It's important to note that Presentation MathML is usually not expressive enough to voice the mathematics properly in all circumstances, which is why screenreaders have to rely on heuristics to analyze the MathML semantically. See the *Screen Reader Support* section for more details about screen reader support via the *assistive-mml* extension.

See the *Assisitve-MML Extension Options* section for information about configuring this component.

8.5 Miscellaneous Components

There are several miscellaneous components that don't fit into other categories. These are:

- *startup*
- *ui/safe*
- *ui/menu*
- *adaptors/liteDOM*
- *core*
- *loader*

They are described in more detail below.

8.5.1 startup

The *startup* component is the one that you would use if you are not using a *combined component*, but are using the `load` array to specify the components you want to load. Like a combined component, you would load this directly via a <*script*> tag, as in

```
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/startup.js">
</script>
```

This is the component that manages the global `MathJax` object. It is responsible for creating the needed objects (like the input and output jax), and for adding the typesetting and conversion methods described in the *Typesetting and Converting Mathematics* section.

See the *Startup Options* section for information about configuring this component.

8.5.2 ui/safe

The *ui/safe* component is intended for use in situations where your readers will be allowed to enter mathematical notation into your pages themselves, such as a question-and-answer site, or a blog with user comments. It filters the mathematics on the page to make sure that certain values within the mathematics are not misused by the reader to cause problems on your page. For example, the `\href` macro normally could be used to insert `javascript:` URLs into the page; the *ui/safe* extension can be used to prevent that.

See the *Safe Extension Options* section for more information on what is filtered and how to control the level of filtering being performed. See *Typesetting User-Supplied Content* for additional details.

8.5.3 ui/menu

The *ui/menu* component implements the MathJax contextual menu, which allows you to obtain the MathML or original format of the mathematics, change parameters about the output renderer, enable accessibility features, and so on.

See the *Contextual Menu Options* section for information about configuring this component.

8.5.4 adaptors/liteDOM

The *adaptors/liteDOM* component implements an alternative to the browser DOM that can be used to parse HTML pages outside of a browser. This can be used in Node applications that don't have access to a browser DOM, or in webworkers that can't access the document DOM.

8.5.5 core

The *core* component includes the code that is required for all other components, including the base classes for input and output jax, math documents, math items within those documents, DOM adaptors, and so on. This component is loaded automatically when needed, so you don't usually have to load it yourself. But you can include it if you are creating your own combined component.

8.5.6 loader

The *loader* component contains the code needed to load other components. It is included automatically by the *startup* component, but if you don't want the features created by the *startup* module, you can use the *loader* component instead to load the MathJax component you need. You can even use it as a general loader for other javascript, if you want.

See the *Loader Options* section for information about configuring this component.

CHAPTER 9

Typesetting and Converting Mathematics

There are two main uses for MathJax:

- Typesetting all the mathematics within a web page, and
- Converting a string containing mathematics into another form.

In version 2, MathJax could perform the first function very well, but it was much harder to do the second. MathJax version 3 makes both easy to do. Both these tasks are described below.

9.1 Typesetting Math in a Web Page

MathJax makes it easy to typeset all the math in a web page, and in fact it will do this automatically when it is first loaded unless you configure it not to. So this is one of the easiest actions to perform in MathJax; if your page is static, there is nothing to do but load MathJax.

If your page is dynamic, and you may be adding math after the page is loaded, then you will need to tell MathJax to typeset the mathematics once it has been inserted into the page. There are two methods for doing that: `MathJax.typeset()` and `MathJax.typesetPromise()`.

The first of these, `MathJax.typeset()`, typesets the page, and does so immediately and synchronously, so when the call finishes, the page will have been typeset. Note, however, that if the math includes actions that require additional files to be loaded (e.g., TeX input that uses *require*, or that includes autoloaded extensions), then an error will be thrown. You can use the `try/catch` command to trap this condition.

The second, `Mathjax.typesetPromise()`, performs the typesetting asynchronously, and returns a promise that is resolved when the typesetting is complete. This properly handles loading of external files, so if you are expecting to process TeX input that can include *require* or autoloaded extensions, you should use this form of typesetting. It can be used with `await` as part of a larger `async` function.

Both functions take an optional argument, which is an array of elements whose content should be processed. An element can be either an actual DOM element, or a CSS selector string for an element or collection of elements. Supplying an array of elements will restrict the typesetting to the contents of those elements only.

9.1.1 Handling Asynchronous Typesetting

It is generally a bad idea to try to perform multiple asynchronous typesetting calls simultaneously, so if you are using `MathJax.typesetPromise()` to make several typeset calls, you should chain them using the promises they return. For example:

```
MathJax.typesetPromise().then(() => {
  // modify the DOM here
  MathJax.typesetPromise();
}).catch((err) => console.log(err.message));
```

This approach can get complicated fast, however, so you may want to maintain a promise that can be used to chain the later typesetting calls. For example,

```
let promise = Promise.resolve();  // Used to hold chain of typesetting calls

function typeset(code) {
  promise = promise.then(() => MathJax.typesetPromise(code()))
                   .catch((err) => console.log('Typeset failed: ' + err.message));
  return promise;
}
```

Then you can use `typeset()` to run code that changes the DOM and typesets the result. The `code()` that you pass it does the DOM modifications and returns the array of elements to typeset, or `null` to typeset the whole page. E.g.,

```
typeset(() => {
  const math = document.querySelector('#math');
  math.innerHTML = '$$\\frac{a}{1-a^2}$$';
  return math;
});
```

would replace the contents of the element with `id="math"` with the specified fraction and have MathJax typeset it (asynchronously). Because the `then()` call returns the result of `MathJax.typesetPromise()`, which is itself a promise, the `then()` will not resolve until that promise is resolved; i.e., not until the typesetting is complete. Finally, since the `typeset()` function returns the `promise`, you can use `await` in an `async` function to wait for the typesetting to complete:

```
await typeset(...);
```

Note that this doesn't take the initial typesetting that MathJax performs into account, so you might want to use `MathJax.startup.promise` in place of `promise` above. I.e., simply use

```
function typeset(code) {
  MathJax.startup.promise = MathJax.startup.promise
    .then(() => MathJax.typesetPromise(code()))
    .catch((err) => console.log('Typeset failed: ' + err.message));
  return MathJax.startup.promise;
}
```

This avoids the need for the global `promise` variable, and makes sure that your typesetting doesn't occur until the initial typesetting is complete.

9.1.2 Resetting Automatic Equation Numbering

The TeX input jax allows you to automatically number equations. When modifying a page, this can lead to problems as numbered equations may be removed and added; most commonly, duplicate labels lead to issues.

You can reset equation numbering using the command

`MathJax.texReset([start])`

where `start` is the number at which to start equation numbering.

9.1.3 Updating Previously Typeset Content

MathJax keeps track of all the math that it has typeset within your page. This is so that if you change the output renderer (using the MathJax contextual menu), it can be changed to use the new format, for example; or if you change the accessibility settings, say to enable the expression explorer, all the math can be updated to incldue the speech strings that it uses. If you modify the page to include new mathematics and call `MathJax.typeset()` or `MathJax.typesetPromise()`, the newly typeset mathematics will be added to the list of already typeset mathematics, as you would expect.

If you modify the page to remove content that contains typeset mathematics, you will need to tell MathJax about that so that it knows the typeset math that you are removed is no longer on the page. You do this by using the `MathJax.typesetClear()` method.

When called with no arguments, `MathJax.typesetClear()` tells MathJax to forget about all the math that has been typeset so far. Note that the math will remain in the page as typeset math, but MathJax will no longer know anything about it. For example, that means that changes to the output renderer or accessibility setting swill not affect any of the math that was typeset previously.

If you remove math from only a portion of the page, you can call `MathJax.typesetClear()` passing it an array of container elements that have been (or will be) removed, and MathJax will forget about the math that is within those containiers, while remembering the rest of the math on the page. For example, if you have an element with `id="has-math"` that you have perviously typeset, and you are planning to replace the contents of this element with new content (stored in a variable `new_html`) that needs to be typeset, you might use soemthing like:

```
const node = document.getElementById('has-math');
MathJax.typesetClear([node]);
node.innerHTML = new_html;
MathJax.typesetPromise([node]).then(() => {
  // the new content is has been typeset
});
```

The argument passed to `MathJax.typestClear()` can be an actual DOM element, as in the example above, or a CSS selector string (e.g., `'#has-math'`), or an array of these. The selector can specify more than one container element (e.g., via a class selector).

If you are using automatic equation numbers and insert new content in the middle of the page, that may require the equation numbers to be adjusted throughout the page. In that case, you can do

```
MathJax.startup.document.state(0);
MathJax.texReset();
MathJax.typeset();
```

to force MathJax to reset the page to the state it was before MathJax processed it (i.e., remove its typeset math), reset the TeX automatic line numbering and labels, and then re-typeset the contents of the page from scratch.

9.1.4 Looking up the Math on the Page

MathJax saves its information about a particular expression that it has typeset in an object called a `MathItem`; each typeset expression has an associated MathItem. You can look up the MathItems using the `MathJax.startup.document.getMathItemsWithin()` function. You pass this a container element (or a CSS selector for an

element or collection of elements, or an array of containers or selectors) and it will return an array of the MathItems that are within those containsers. E.g.,

```
MathJax.startup.document.getMathItemsWithin(document.body);
```

will return an array of all the MathItems for the typeset math on the page. See the MathItem definition for details on the contents of the MathItem structure. The MathItem is the v3 replacement for the v2 *ElementJax* object, and `getMathItemsWithin()` performs a similar function to the v2 function `MathJax.Hub.getAllJax()`.

9.1.5 Typesetting User-Supplied Content

Mathematics formats like LaTeX and MathML allow a powerful range of layout options, including access to hyperlinks, CSS styles, font selection and sizing, spacing, and so on. Such features give you a great deal of flexibility in producing the mathematics for your pages, but if your readers are allowed to enter mathematics into you pages (e.g., for a question-and-answer site, or in comments on a blog), these features can be abused to cause problems for other readers and pose a potential security risk to them. For example, the TeX `\href` command can be used to insert `javascript:` links into the page, while the `\style` macro could be used to disrupt the user interface or layout of your pages.

In order to limit the potential interference that could be caused by the mathematics entered by your readers, MathJax provides the *ui/safe* extension. This extension filters the mathematics on the page in order to try to remove problematic attributes, like javascript links, or font sizes that are too large or too small, or style settings that would be disruptive to the page layout. If your page allows your readers to post content that includes mathematics processed by MathJax, you should strongly consider using the *ui/safe* extension.

See the *Safe Extension Options* section for details of how to load and configure the *ui/safe* extension.

9.1.6 Loading MathJax Only on Pages with Math

The MathJax combined configuration files are large, and so you may wish to include MathJax in your page only if it is necessary. If you are using a content-management system that puts headers and footers into your pages automatically, you may not want to include MathJax directly, unless most of your pages include math, as that would load MathJax on *all* your pages. Once MathJax has been loaded, it should be in the browser's cache and load quickly on subsequent pages, but the first page a reader looks at will load more slowly. In order to avoid that, you can use a script like the following one that checks to see if the content of the page seems to include math, and only loads MathJax if it does. Note that this is not a very sophisticated test, and it may think there is math in some cases when there really isn't but it should reduce the number of pages on which MathJax will have to be loaded.

Create a file called `check-for-tex.js` containing the following:

```
(function () {
  var body = document.body.textContent;
  if (body.match(/(?:\$|\\\(|\\\[|\\begin\{.*?})/)) {
    if (!window.MathJax) {
      window.MathJax = {
        tex: {
          inlineMath: {'[+]': [['$', '$']]}
        }
      };
    }
    var script = document.createElement('script');
    script.src = 'https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js';
    document.head.appendChild(script);
  }
})();
```

and then use

```
<script src="check-for-tex.js" defer></script>
```

in order to load the script when the page content is ready. Note that you will want to include the path to the location where you stored `check-mathjax.js`, that you should change `tex-chtml.js` to whatever component file you want to use, and that the `window.MathJax` value should be set to whatever configuration you want to use. In this case, it just adds dollar signs to the in-line math delimiters. Finally, adjust the `body.match()` regular expression to match whatever you are using for math delimiters.

This simply checks if there is something that looks like a TeX in-line or displayed math delimiter, and loads MathJax if there is. If you are using different delimiters, you will need to change the pattern to include those (and exclude any that you don't use). If you are using AsciiMath instead of TeX, then change the pattern to look for the AsciiMath delimiters.

If you are using MathML, you may want to use

```
if (document.body.querySelector('math')) {...}
```

for the test instead (provided you aren't using namespace prefixes, like <*m:math*>).

9.2 Converting a Math String to Other Formats

An important use case for MathJax is to convert a string containing mathematics (in one of the three forms that MathJax understands) and convert it into another form (either MathML, or one of the output formats that MathJax supports). This was difficult to do in MathJax version 2, but easy to do in version 3.

When MathJax starts up, it creates methods for converting from the input format(s) to the output format(s) that you have loaded, and to MathML format. For example, if you have loaded the MathML input jax and the SVG output jax (say by using the `mml-svg` component), then MathJax will create the following conversion methods for you:

```
MathJax.mathml2svg(math[,options])
MathJax.mathml2svgPromise(math[,options])
MathJax.mathml2mml(math[,options])
MathJax.mathml2mmlPromise(math[,options])
```

If you had loaded the TeX input jax as well, you would also get four more methods, with `tex` in place of `mathml`.

As the names imply, the `Promise` functions perform the conversion asynchronously, and return promises, while the others operate synchronously and return the converted form immediately. The first two functions (and any others like them) produce DOM elements as the results of the conversion, with the promise versions passing that to their `then()` functions as their argument (see the section on *Asynchronous Conversion* below), and the non-promise versions returning them directly. You can insert these DOM elements into the document directly, or you can use their `outerHTML` property to obtain their serialized string form.

The functions that convert to MathML produce serialized MathML strings automatically, rather than DOM elements. (You can use the browser's `DOMParser` object to convert the string into a MathML DOM tree if you need one.)

9.2.1 Conversion Options

All four of these functions require an argument that is the math string to be converted (e.g., the serialized MathML string, or in the case of `tex2chtml()`, the TeX or LaTeX string). You can also pass a second argument that is an object containing options that control the conversion process. The options that can be included are:

- `display`, a boolean specifying whether the math is in display-mode or not (for TeX input). Default is `true`.
- `em`, a number giving the number of pixels in an `em` for the surrounding font. Default is `16`.
- `ex`, a number giving the number of pixels in an `ex` for the surrounding font. Default is `8`.
- `containerWidth`, a number giving the width of the container, in pixels. Default is 80 times the `ex` value.
- `lineWidth'`, a number giving the line-breaking width in `em` units. Default is a very large number (100000), so effectively no line breaking.
- `scale`, a number giving a scaling factor to apply to the resulting conversion. Default is 1.

For example,

```
let html = MathJax.tex2chtml('\\sqrt{x^2+1}', {em: 12, ex: 6, display: false});
```

would convert the TeX expression `\sqrt{x^2+1}` to HTML as an in-line expression, with `em` size being 12 pixels and `ex` size being 6 pixels. The result will be a DOM element containing the HTML for the expression. Similarly,

```
let html = MathJax.tex2chtml('\\sqrt{x^2+1}', {em: 12, ex: 6, display: false});
let text = html.outerHTML;
```

sets `text` to be the serialized HTML string for the expression.

9.2.2 Obtaining the Output Metrics

Since the `em`, `ex`, and `containerWidth` all depend on the location where the math will be placed in the document (they are values based on the surrounding text font and the container elements width), MathJax provides a method for obtaining these values from a given DOM element. The method

`MathJax.getMetricsFor(node, display)`

takes a DOM element (`node`) and a boolean (`display`), indicating if the math is in display mode or not, and returns an object containing all six of the options listed above. You can pass this object directly to the conversion methods discussed above. So you can do something like

```
let node = document.querySelector('#math');
let options = MathJax.getMetricsFor(node, true);
let html = MathJax.tex2svg('\\sqrt{x^2+1}', options);
node.appendChild(html);
```

in order to get get the correct metrics for the (eventual) location of the math that is being converted. Of course, it would be easier to simply insert the TeX code into the page and use `MathJax.typeset()` to typeset it, but this is just an example to show you how to obtain the metrics from a particular location in the page.

Note that obtaining the metrics causes a page refresh, so it is expensive to do this. If you need to get the metrics from many different locations, there are more efficient ways, but these are advanced topics to be dealt with elsewhere.

9.2.3 Obtaining the Output Stylesheet

The output from the SVG and CommonHTML output jax both depend on CSS stylesheets in order to properly format their results. You can obtain the SVG stylesheet element by calling

```
MathJax.svgStylesheet();
```

and the HTML stylesheet from

```
MathJax.chtmlStylesheet();
```

The CommonHTML output jax CSS can be quite large, so the output jax tries to minimize the stylesheet by including only the styles that are actually needed for the mathematics that has been processed by the output jax. That means you should request the stylesheet only *after* you have typeset the mathematics itself.

Moreover, if you typeset several expressions, the stylesheet will include everything needed for all the expressions you have typeset. If you want to reset the stylesheet, then use

```
MathJax.startup.output.clearCache();
```

if the output jax is the CommonHTML output jax. So if you want to produce the style sheet for a single expression, issue the `clearCache()` command just before the `tex2chtml()` call.

9.2.4 Asynchronous Conversion

If you are converting TeX or LaTeX that might use *require* to load extensions, or where extensions might be autoloaded, you will either need to use one of the "full" components that include all the extensions, or preload all the extensions you need if you plan to use the synchronous calls listed above. Otherwise, you can use the promise-based calls, which handle the loading of extensions transparently.

For example,

```
let node = document.querySelector('#math');
let options = MathJax.getMetricsFor(node, true);
MathJax.tex2chtmlPromise('\\require{bbox}\\bbox[red]{\\sqrt{x^2+1}}', options)
  .then((html) => {
    node.appendChild(html);
    let sheet = document.querySelector('#MJX-CHTML-styles');
    if (sheet) sheet.parentNode.removeChild(sheet);
    document.head.appendChild(MathJax.chtmlStylesheet());
  });
```

would get the metrics for the element with `id="math"`, convert the TeX expression using those metrics (properly handling the asynchronous load needed for the `\require` command); then when the expression is typeset, it is added to the document and the CHTML stylesheet is updated.

CHAPTER 10

Hosting Your Own Copy of MathJax

We recommend using a CDN service if you can, but you can also install MathJax on your own server, or locally on your own hard disk. You may need to do this if you are *creating a custom build* of MathJax, for example, or if you wish to use MathJax off-line.

10.1 Acquiring the MathJax Code

In order to host your own version of MathJax, you must first obtain a copy of the MathJax code. That can be done in several ways, the easiest being to use `npm` (the node package manager), or `git` to get MathJax from its GitHub development repository.

10.1.1 Getting MathJax via npm

To include MathJax in your project, use the command

```
npm install mathjax@3
```

This will install MathJax in `node_modules/mathjax` subdirectory of your current directory. It will include the pre-built components in the `node_modules/mathjax/es5` directory. (Note that it is important to use `mathjax@3`, as we are still making v2 releases, and so the latest mathjax npm package may not be the v3 one. The `latest` version on `npmjs.com` appears to be chronological rather than by version number.)

If you need access to the source code, as well. Then use

```
npm install mathjax-full@3
```

which installs MathJax in the `node_modules/mathjax-full` subdirectory, the source files for the components in `node_modules/mathjax-full/components/src`, the typescript source files for MathJax in `node_modules/mathjax-full/ts`, and the compiled javascript files from the typescript source in `node_modules/mathjax-full/js`.

10.1.2 Getting MathJax via git

To obtain a copy of MathJax from the GitHub component repository, use the command

```
git clone https://github.com/mathjax/MathJax.git mathjax
```

This will install a copy of MathJax in the `mathjax/es5` directory.

If you need access to the source code as well, then use

```
git clone https://github.com/mathjax/MathJax-src.git mathjax
```

which will install the source code for MathJax in the `mathjax` sub-directory of your current directory. You will need to compile the typescript source files and build the component files by hand, as they are not part of the repository itself. To do this, do the following:

```
cd mathjax
npm install
npm run compile
npm run make-components
cd ..
```

This will compile the typescript source files from the `mathjax/ts` directory into javascript files in the `mathjax/js` directory, and then will build the component files from `mathjax/components/src` into the `mathjax/es5` directory.

10.2 Make the Files Available

Once you have acquired the MathJax files by one of the methods described above, you need to make the proper files available on your web server. Note that most of the files in the MathJax distribution are not needed on the server. For example, the `mathjax/ts` directory is typescript source code for MathJax, and this is compiled into the javascript files found in the `mathjax/js` directory. But even these are not the files you want on your server. These javascript files are further processed into the MathJax components stored in the `mathjax/es5` files using the data in the `mathjax/components/src` directory.

It is the contents of the `mathjax/es5` directory that you want to make available on your server, as these are the files that are served from the CDNs that provide MathJax. You should move them to a convenient location on your server. This might be a top-level directory called `mathjax`, for example.

10.3 Linking to you Your Copy of MathJax

You can include MathJax in your web page by putting

```
<script src="path-to-MathJax/tex-chtml.js" id="MathJax-script" async></script>
```

in your document's `<head>` block. Here, `tex-chtml.js` is the combined component that you are loading, and this is just an example; you will need to pick the one you want to use. See the section on *Configuring and Loading MathJax* for more details.

The `path-to-MathJax` should be replaced by the URL for the main MathJax directory, so if you have put the `mathjax/es5` directory at the top level of you server's web site and named it `mathjax`, you could use

```
<script src="/mathjax/tex-chtml.js" id="MathJax-script" async></script>
```

to load MathJax in your page. For example, your page could look like

```
<html>
    <head>
        ...
        <script src="/mathjax/tex-chtml.js" id="MathJax-script" async></script>
    </head>
    <body>
        ...
    </body>
</html>
```

10.4 Fonts on Shared Servers

Typically, you want to have MathJax installed on the same server as your web pages that use MathJax. There are times, however, when that may be impractical, or when you want to use a MathJax installation at a different site. For example, a departmental server at `www.math.yourcollege.edu` might like to use a college-wide installation at `www.yourcollege.edu` rather than installing a separate copy on the departmental machine. MathJax can certainly be loaded from another server, but there is one important caveat — The same-origin security policy for cross-domain scripting.

Some browsers' (e.g., Firefox's) interpretation of the same-origin policy is more strict than most other browsers, and it affects how fonts are loaded with the `@font-face` CSS directive. MathJax's CommonHTML output modes use this directive to load web-based math fonts into a page when the user doesn't have them installed locally on their own computer. These browsers' security policies, however, only allow this when the fonts come from the same server as the web page itself, so if you load MathJax (and hence its web fonts) from a different server, they won't be able to access those web fonts. In this case, MathJax's CommonHTML output mode will not show the correct fonts.

There is a solution to this, however, if you manage the server where MathJax is installed, and if that server is running the Apache web software. In the remote server's MathJax folder, create a file called `.htaccess` that contains the following lines:

```
<FilesMatch "\.(ttf|otf|eot|woff)$">
<IfModule mod_headers.c>
Header set Access-Control-Allow-Origin "*"
</IfModule>
</FilesMatch>
```

and make sure the permissions allow the server to read this file. (The file's name starts with a period, which causes it to be an "invisible" file on unix-based operating systems. Some systems, particularly those with graphical user interfaces, may not allow you to create such files, so you might need to use the command-line interface to accomplish this.)

This file should make it possible for pages at other sites to load MathJax from this server in such a way that Firefox (and the other browsers with similar same-origin policies that apply to fonts) will be able to download the web-based fonts. If you want to restrict the sites that can access the web fonts, change the `Access-Control-Allow-Origin` line to something like:

```
Header set Access-Control-Allow-Origin "http://www.math.yourcollege.edu"
```

so that only pages at `www.math.yourcollege.edu` will be able to download the fonts from this site. See the open font library discussion of web-font linking for more details.

10.5 Firefox and Local Fonts

Firefox's same-origin security policy affects its ability to load web-based fonts, as described above. This has implications not only to cross-domain loading of MathJax, but also to using MathJax locally from your hard disk. Firefox's interpretation of the same-origin policy for local files used to be that the "same domain" for a page is the directory where that page exists, or any of its subdirectories. This allowed MathJax to be loaded from a subdirectory of the director where the web page was loaded.

This is no longer the case with Firefox starting with version 68 and going forward (see their documentation). Now there is no same origin for a `file://` URL (the origin for a page loaded from a `file://` URL is unique).

This means there are limited options for using MathJax in Firefox with a local copy of MathJax. The easiest option is to use the SVG output renderer rather than the CommonHTML output, as that does not require fonts to be loaded, so avoids the same-origin issue. Alternatively, you could install the MathJax TeX fonts as system fonts so that Firefox doesn't hav to try to load them as web fonts.

This is an unfortunate restriction for MathJax (though we understand their reasoning), but it is a limitation imposed by Firefox's security model that MathJax can not circumvent. Currently, this is not a problem for other browsers, though there is no guarantee that it won't be in the future.

CHAPTER 11

Making a Custom Build of MathJax

MathJax provides a number of combined components that load everything you need to run MathJax with a given input and output format. Still, you might find that none of the ones we provide fully suit your needs, and that you would like to include additional components in the build, or perhaps want to include customized configuration options.

You can use the MathJax component build tools to make your own custom component that has exactly the pieces and configuration that you want. You can also use them to make a custom extension, for example a TeX input extension, that takes advantage of the components already loaded, but implements additional functionality. These possibilities are described in *Building a Custom Component* below.

It is also possible to make a completely custom build of MathJax that doesn't use the MathJax components at all, but includes direct calls to the MathJax source files. This is described in *A Custom MathJax Build* below.

If you wish to include MathJax as part of a larger project, you can use either of the techniques to do that, and make a webpacked file that includes your own project code as well as MathJax.

11.1 Getting Things Ready

Your first step is to download a copy of MathJax via `npm` or `git`, as described in the section on *Acquiring the MathJax Code*.

- If you use `npm`, you will want to install the `mathjax-full` package rather than the `mathjax` package, since the former includes all the source code, in both its original and compiled forms, along with the webpacked components.
- If you use `git`, be sure to run the commands to compile and make the components, as listed in *Getting MathJax via git*.

In either case, you should have a `js`, an `es5`, and a `components` directory, either in the `node_modules/mathjax-full` directory (for `npm` installations) or in the main directory (for `git` installations).

Your second step is to obtain the tools needed to package your custom code using `webpack`. Use the commands

```
npm install webpack
npm install webpack-cli
npm install terser-webpack-plugin
npm install babel-loader
npm install @babel/core
npm install @babel/preset-env
```

to install `webpack` and its needed libraries. Once this is done, you should be able to make the components described below. The building instructions assume you used `npm` to aquire MathJax; if you used `git`, then you will need to remove `node_modules/mathjax-full` from the paths that incldue them.

11.2 Building a Custom Component

MathJax comes with a number of predefined components, and you can use their definitions as a starting point for your own custom component. There are also custom component examples (with documentation) in the MathJax web demos repository, which are similar to the ones described here.

There are two kinds of components you could build:

- A **combined component** that brings together several other components (the `tex-chtml` component is a combined component)
- A **extension component** that contains what is needed for one feature and can be loaded along with other components to add that feature to MathJax.

We describe how you can create each of these below. In both cases, you should create a directory to hold your component's support files. You will need the main control file for the component (that includes the code that defines the component), and a webpack control file that will tell MathJax's build tools how to handle your component. These will be discussed in the sections below.

11.2.1 A Custom Combined Component

After downloading a copy of MathJax as described in the section on *Getting Things Ready*, make the directory for your component and `cd` to that directory. We will assume the directory is called `custom-mathjax` for this discussion.

For this example, we will create a custom build that has the TeX input jax and the SVG output jax, and we will load the `newcommand`, `ams`, and `configmacros` extensions, but will not include `require` or `autoload`, so the user will not be able load any additional TeX extensions. This component also includes the contextual menu.

The Control File

Create a javascript file to house the component and call it `custom-mathjax.js`. The file should contain the following code (we assume here that you used `npm` to install MathJax. If not, you may need to adjust the locations in the `require()` commands).

```
//
//  Initialize the MathJax startup code
//
require('mathjax-full/components/src/startup/lib/startup.js');

//
```

(continues on next page)

(continued from previous page)

```
//  Get the loader module and indicate the modules that
//  will be loaded by hand below
//
const {Loader} = require('mathjax-full/js/components/loader.js');
Loader.preLoad(
  'loader', 'startup',
  'core',
  'input/tex-base',
  '[tex]/ams',
  '[tex]/newcommand',
  '[tex]/configmacros',
  'output/svg', 'output/svg/fonts/tex.js',
  'ui/menu'
);

//
// Load the components that we want to combine into one component
//   (the ones listed in the preLoad() call above)
//
require('mathjax-full/components/src/core/core.js');

require('mathjax-full/components/src/input/tex-base/tex-base.js');
require('mathjax-full/components/src/input/tex/extensions/ams/ams.js');
require('mathjax-full/components/src/input/tex/extensions/newcommand/newcommand.js');
require('mathjax-full/components/src/input/tex/extensions/configmacros/configmacros.js
↪');

require('mathjax-full/components/src/output/svg/svg.js');
require('mathjax-full/components/src/output/svg/fonts/tex/tex.js');

require('mathjax-full/components/src/ui/menu/menu.js');

//
// Update the configuration to include any updated values
//
const {insert} = require('mathjax-full/js/util/Options.js');
insert(MathJax.config, {
  tex: {
    packages: {'[+]': ['ams', 'newcommand', 'configmacros']}
  }
});

//
// Loading this component will cause all the normal startup
//   operations to be performed
//
require('mathjax-full/components/src/startup/startup.js');
```

This loads the various components that we want to include in the combined component, including the standard startup code so that the usual startup process is included.

The Webpack Configuration

Next, create the file `webpack.config.js` that includes the following:

```
const PACKAGE = require('mathjax-full/components/webpack.common.js');

module.exports = PACKAGE(
  'custom-mathjax',                    // the name of the package to build
  '../node_modules/mathjax-full/js',   // location of the mathjax library
  [],                                  // packages to link to
  __dirname,                           // our directory
  '.'                                  // where to put the packaged component
);
```

This file gives the name that will be used for this component (`custom-mathjax` in this case), a pointer to where the MathJax javascript code is to be found (adjust this to suit your setup), an array of components that we assume are already loaded when this one is loaded (none in this case), the directory name we are working in (always `__dirname`), and the directory where we want the final packaged component to go (the default is the `mathjax-full/es5` directory, but we set it to the directory containing the source files, and the component will end with `.min.js`).

Most of the real work is done by the `mathjax-full/components/webpack.common.js` file, which is included in the first line here.

Building the Component

Once these two files are ready, you are ready to build the component. First, make sure that you have obtained the needed tools as described in *Getting Things Ready* above. Then you should be able to use the command

```
node ../node_modules/mathjax-full/components/bin/makeAll
```

to process your custom build. You should end up with a file `custom-mathjax.min.js` in the directory with the other files. If you put this on your web server, you can load it into your web pages in place of loading MathJax from a CDN. This file will include all that you need to run MathJax on your pages. Just add

```
<script src="custom-mathjax.min.js" id="MathJax-script" async></script>
```

to your page and you should be in business (adjust the URL to point to wherever you have placed the `custom-mathjax.min.js` file).

Configuring the Component

Note that you can still include a `MathJax = {...}` definition in your web page before loading this custom MathJax build if you want to customize the configuration for a specific page. You could also include configuration within the component itself, as we did for the TeX `packages` array. This will override any page-provided configuration, however, so if you want to provide non-standard defaults that can still be overridden in the page, use

```
//
// Update the configuration to include any updated values
//
const {insert} = require('mathjax-full/js/util/Options.js');
insert(MathJax.config, {tex: {packages: {'[+]': ['ams', 'newcommand', 'configmacros']}
↪}}, false);
MathJax.config = insert({
  // your default options here
}, MathJax.config);
```

which will update the TeX packages, and then merge the user's configuration options into your defaults and set `MathJax.config` to the combined options.

Fonts for CommonHTML

If you include the CommonHTML output jax in your custom build, the actual web fonts are not included in the webpacked file, so you will probably need to include *fontURL* in the *chtml* block of your configuration and have it provide a URL where the fonts can be found. They are in the `mathjax-full/es5/output/chtml/fonts/woff-v2` directory, and you can put them on your server, or simply point *fontURL* to one of the CDN directories for the fonts.

11.2.2 A Custom Extension

Making a custom extension is very similar to making a custom combined component. The main difference is that the extension may rely on other components, so you need to tell the build system about that so that it doesn't include the code from those other components. You also don't load the extension file directly (like you do the combined component above), but instead include it in the *load* array of the *loader* configuration block, and MathJax loads it itself, as discussed below.

For this example, we make a custom TeX extension that defines new TeX commands implemented by javascript functions.

The commands implemented here provide the ability to generate MathML token elements from within TeX by hand. This allows more control over the content and attributes of the elements produced. The macros are `\mi`, `\mo`, `\mn`, `\ms`, and `\mtext`, and they each take an argument that is the text to be used as the content of the corresponding MathML element. The text is not further processed by TeX, but the extension does convert sequences of the form `\uNNNN` (where the `N` are hexadecimal digits) into the corresponding unicode character; e.g., `\mi{\u2460}` would produce U+2460, a circled digit 1, as the content of an `mi` element.

The Extension File

After downloading a copy of MathJax as described in the section on *Getting Things Ready*, create a directory for the extension named `custom-extension` and `cd` to it. Then create the file `mml.js` containing the following text:

```
import {Configuration}  from '../node_modules/mathjax-full/js/input/tex/Configuration.
↪js';
import {CommandMap} from '../node_modules/mathjax-full/js/input/tex/SymbolMap.js';
import TexError from '../node_modules/mathjax-full/js/input/tex/TexError.js';

/**
 * This function prevents multi-letter mi elements from being
 *   interpreted as TEXCLASS.OP
 */
function classORD(node) {
   this.getPrevClass(node);
   return this;
}

/**
 *  Convert \uXXXX to corresponding unicode characters within a string
 */
function convertEscapes(text) {
   return text.replace(/\\u([0-9A-F]{4})/gi, (match, hex) => String.
↪fromCharCode(parseInt(hex,16)));
}

/**
 * Allowed attributes on any token element other than the ones with default values
```

(continues on next page)

(continued from previous page)

```
 */
const ALLOWED = {
   style: true,
   href: true,
   id: true,
   class: true
};

/**
 * Parse a string as a set of attribute="value" pairs.
 */
function parseAttributes(text, type) {
   const attr = {};
   if (text) {
      let match;
      while ((match = text.match(/^\s*((?:data-)?[a-z][-a-z]*)\s*=\s*(?:"([^"]*)"|(.*?
↪))(?:\s+|,\s*|$)/i))) {
         const name = match[1], value = match[2] || match[3]
         if (type.defaults.hasOwnProperty(name) || ALLOWED.hasOwnProperty(name) ||␣
↪name.substr(0,5) === 'data-') {
            attr[name] = convertEscapes(value);
         } else {
            throw new TexError('BadAttribute', 'Unknown attribute "%1"', name);
         }
         text = text.substr(match[0].length);
      }
      if (text.length) {
         throw new TexError('BadAttributeList', 'Can\'t parse as attributes: %1',␣
↪text);
      }
   }
   return attr;
}

/**
 *  The mapping of control sequence to function calls
 */
const MmlMap = new CommandMap('mmlMap', {
   mi: ['mmlToken', 'mi'],
   mo: ['mmlToken', 'mo'],
   mn: ['mmlToken', 'mn'],
   ms: ['mmlToken', 'ms'],
   mtext: ['mmlToken', 'mtext']
}, {
   mmlToken(parser, name, type) {
      const typeClass = parser.configuration.nodeFactory.mmlFactory.
↪getNodeClass(type);
      const def = parseAttributes(parser.GetBrackets(name), typeClass);
      const text = convertEscapes(parser.GetArgument(name));
      const mml = parser.create('node', type, [parser.create('text', text)], def);
      if (type === 'mi') mml.setTeXclass = classORD;
      parser.Push(mml);
   }
});

/**
 * The configuration used to enable the MathML macros
```

(continues on next page)

(continued from previous page)

```
 */
const MmlConfiguration = Configuration.create(
   'mml', {handler: {macro: ['mmlMap']}}
);
```

The comments explain what this code is doing. The main piece needed to make it a TeX extension is the `Configuration` created in the last few lines. It creates a TeX package named `mml` that handles macros through a `CommandMap` named `mmlMap` that is defined just above it. That command map defines five macros described at the beginning of this section, each of which is tied to a method named `mmlToken` in the object that follows, passing it the name of the MathML token element to create. The `mmlToken` method is the one that is called by the TeX parser when the `\mi` and other macros are called. It gets the argument to the macro, and any optional attributes, and creates the MathML element with the attributes, using the argument as the text of the element.

The Webpack Configuration

Next, create the file `webpack.config.js` that includes the following:

```
const PACKAGE = require('mathjax-full/components/webpack.common.js');

module.exports = PACKAGE(
  'mml',                                  // the name of the package to build
  '../node_modules/mathjax-full/js',      // location of the mathjax library
  [                                       // packages to link to
     'components/src/core/lib',
     'components/src/input/tex-base/lib'
  ],
  __dirname,                              // our directory
  '.'                                     // where to put the packaged component
);
```

This file gives the name that will be used for this component (`mml` in this case), a pointer to where the MathJax javascript code is to be found (adjust this to suit your setup), an array of components that we assume are already loaded when this one is loaded (the `core` and `tex-base` components in this case), the directory name we are working in (always `__dirname`), and the directory where we want the final packaged component to go (the default is the `mathjax-full/es5` directory, but we set it to the directory containing the source files, and the component will end with `.min.js`).

Most of the real work is done by the `mathjax-full/components/webpack.common.js` file, which is included in the first line here.

Building the Extension

Once these two files are ready, you are ready to build the component. First, make sure that you have obtained the needed tools as described in *Getting Things Ready* above. Then you should be able to use the command

```
node ../node_modules/mathjax-full/components/bin/makeAll
```

to process your custom build. You should end up with a file `mml.min.js` in the directory with the other files. If you put this on your web server, you can load it as a component by putting it in the `load` array of the `loader` block of your configuration, as descrinbed below.

Loading the Extension

To load your custom extension, you will need to tell MathJax where it is located, and include it in the file to be loaded on startup. MathJax allows you to define paths to locations where your extensions are stored, and then you can refer to the extensions in that location by using a prefix that represents that location. MathJax has a pre-defined prefix, `mathjax` that is the default prefix when none is specified explicitly, and it refers to the location where the main MathJax file was loaded (e.g., the file `tex-svg.js`, or `startup.js`).

You can define your own prefix to point to the location of your extensions by using the `paths` object in the `loader` block of your configuration. In our case (see code below), we add a `custom` prefix, and have it point to the URL of our extension (in this case, the same directory as the HTML file that loads it, represented by the URL `.`). We use the `custom` prefix to specify `[custom]/mml.min.js` in the `load` array so that our extension will be loaded.

Finally, we add the `mml` extension to the `packages` array in the `tex` block of our configuration via the special notation `{'[+]': [...]}` that tells MathJax to append the given array to the existing `packages` array that is already in the configuration by default. So this uses all the packages that were already specified, plus our new `mml` package that is defined in our extension.

The configuration and loading of MathJax now looks something like this:

```
<script>
MathJax = {
   loader: {
      load: ['[custom]/mml.min.js'],
      paths: {custom: '.'}
   },
   tex: {
      packages: {'[+]': ['mml']}
   }
};
</script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js">
</script>
```

You should change the `custom: '.'` line to point to the actual URL for your server.

This example loads the `tex-chtml.js` combined component, so the TeX input is already loaded when our extension is loaded. If you are using `startup.js` instead, and including `input/tex` in the `load` array, you will need to tell MathJax that your extension depends on the `input/tex` extension so that it waits to load your extension until after the TeX input jax is loaded. To do that, add a `dependencies` block to your configuration like the following:

```
<script>
MathJax = {
   loader: {
      load: ['input/tex', 'output/chtml', '[custom]/mml.min.js'],
      paths: {custom: '.'},
      dependencies: {'[custom]/mml.min.js': ['input/tex']}
   },
   tex: {
      packages: {'[+]': ['mml']}
   }
};
</script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/startup.js">
</script>
```

This example can be seen live in the MathJax 3 demos repository.

11.3 A Custom MathJax Build

It is possible to make a completely custom build of MathJax that is not based on other MathJax components at all. The following example shows how to make a custom build that provides a function for obtaining the speech string for a given TeX math string. This example is similar to one in the MathJax3 demos repository.

After downloading a copy of MathJax as described in the section on *Getting Things Ready*, create a directory called `mathjax-speech` and `cd` into it.

11.3.1 The Custom Build File

Create the custom MathJax file named `mathjax-speech.js` containing the following:

```
//
//  Load the desired components
//
const mathjax     = require('mathjax-full/js/mathjax.js').mathjax;      // MathJax␣
↪core
const TeX         = require('mathjax-full/js/input/tex.js').TeX;        // TeX input
const MathML      = require('mathjax-full/js/input/mathml.js').MathML;  // MathML␣
↪input
const browser     = require('mathjax-full/js/adaptors/browserAdaptor.js').
↪browserAdaptor; // browser DOM
const Enrich      = require('mathjax-full/js/a11y/semantic-enrich.js').EnrichHandler;␣
↪    // semantic enrichment
const Register    = require('mathjax-full/js/handlers/html.js').RegisterHTMLHandler; ␣
↪    // the HTML handler
const AllPackages = require('mathjax-full/js/input/tex/AllPackages').AllPackages;    ␣
↪    // all TeX packages
const STATE       = require('mathjax-full/js/core/MathItem.js').STATE;

const sreReady    = require('mathjax-full/js/a11y/sre.js').sreReady();    // SRE␣
↪promise;

//
//  Register the HTML handler with the browser adaptor and add the semantic enrichment
//
Enrich(Register(browser()), new MathML());

//
//  Initialize mathjax with a blank DOM.
//
const html = MathJax.document('', {
   enrichSpeech: 'shallow',           // add speech to the enriched MathML
   InputJax: new TeX({
      packages: AllPackages.filter((name) => name !== 'bussproofs'),  // Bussproofs␣
↪needs an output jax
      macros: {require: ['', 1]}      // Make \require a no-op since all packages are␣
↪loaded
   })
});
```

(continues on next page)

(continued from previous page)

```
//
//  The user's configuration object
//
const CONFIG = window.MathJax || {};

//
//  The global MathJax object
//
window.MathJax = {
   version: mathjax.version,
   html: html,
   sreReady: sreReady,

   tex2speech(tex, display = true) {
      const math = new html.options.MathItem(tex, inputJax, display);
      math.convert(html, STATE.CONVERT);
      return math.root.attributes.get('data-semantic-speech') || 'no speech text␣
↪generated';
   }
}

//
// Perform ready function, if there is one
//
if (CONFIG.ready) {
   sreReady.then(CONFIG.ready);
}
```

Unlike the component-based example above, this custom build calls on the MathJax source files directly. The `require` commands at the beginning of the file load the needed objects, and the rest of the code instructs MathJax to create a `MathDocument` object for handling the conversions that we will be doing (using a TeX input jax), and then defines a global `MathJax` object that has the `tex2speech()` function that our custom build offers.

11.3.2 The Webpack Configuration

Next, create the file `webpack.config.js` that includes the following:

```
const PACKAGE = require('mathjax-full/components/webpack.common.js');

module.exports = PACKAGE(
  'mathjax-speech',                    // the name of the package to build
  '../node_modules/mathjax-full/js',   // location of the mathjax library
  [],                                  // packages to link to
  __dirname,                           // our directory
  '.'                                  // where to put the packaged component
);
```

This file gives the name that will be used for this component (`mathjax-speech` in this case), a pointer to where the MathJax javascript code is to be found (adjust this to suit your setup), an array of components that we assume are already loaded when this one is loaded (none, since this is a self-contained build), the directory name we are working in (always `__dirname`), and the directory where we want the final packaged component to go (the default is the `mathjax-full/es5` directory, but we set it to the directory containing the source files, and the component will end with `.min.js`).

Most of the real work is done by the `mathjax-full/components/webpack.common.js` file, which is included in the first line here.

11.3.3 Building the Custom File

Once these two files are ready, you are ready to make your custom build. First, make sure that you have obtained the needed tools as described in *Getting Things Ready* above. Then you should be able to use the command

```
node ../node_modules/mathjax-full/components/bin/makeAll
```

to process your custom build. You should end up with a file `mathjax-speech.min.js` in the directory with the other files. it will contain just the parts of MathJax that are needed to implement the `MathJax.tex2speech()` command defined in the file above. Note that this is not enough to do normal typesetting (for example, no output jax has been included), so this is a minimal file for producing the speech strings from TeX input.

11.3.4 Using the File in a Web Page

If you put the `mathjax-speech.min.js` file on your web server, you can load it into your web pages in place of loading MathJax from a CDN. This fill will include all that you need to use the `MathJax.tex2speech()` command in your pages. Just add

```
<script src="mathjax-speech.min.js" id="MathJax-script" async></script>
```

to your page (adjust the URL to point to wherever you have placed the `custom-mathjax.min.js` file). Then you can use javascript calls like

```
const speech = MathJax.tex2speech('\\sqrt{x^2+1}', true);
```

to obtain a text string that contains the speech text for the square root given in the TeX string.

Note, however, that the Speech-Rule Engine (SRE) that underlies the speech generation loads asynchronously, so you have to be sure that SRE is ready before you make such a call. The `mathjax-speech.js` file provides two ways of handling the synchronization with SRE. The first is to use the global `MathJax` variable to include a `ready()` function that is called when SRE is ready. For example,

```
window.speechReady = false;
window.MathJax = {
   ready: () => {
      window.speechReady = true;
   }
};
```

would set the global variable `speechReady` to true when SRE is ready to run (so you can check that value to see if speech can be generated yet). A more sophisticated `ready()` function could allow you to queue translations to be performed, and when SRE is ready, it performs them. Alternatively, if you have a user interface that allows users to transform TeX expressions, for example, then you could initially disable to buttons that trigger speech generation, and use the `ready()` function to enable them. That way, the user can't ask for speech translation until it can be produced.

The second method of synchronizing with SRE is through the fact that the code sets `MathJax.sreReady` to a promise that is resolves when SRE is ready, which you can use to make sure SRE is ready when you want to do speech generation. For example

```
function showSpeech(tex, display = false) {
   MathJax.sreReady = MathJax.sreReady.then(() => {
     const speech = MathJax.tex2speech(tex, display);
     const output = document.getElementById('speech');
     output.innerHTML = '';
     output.appendChild(document.createTextNode(speech));
```

(continues on next page)

(continued from previous page)

```
    });
}
```

provides a function that lets you specify a TeX string to translate, and then (asynchronously) generates the speech for it and displays it as the contents of the DOM element with `id="speech"` in the page.

CHAPTER 12

Examples in a Browser

There are a number of example files in the MathJax web demo repository (see the list of demos). These include documentation as well as live examples that you can run.

In addition, there are examples for:

- *Configuring MathJax using an external script*
- *Configuring and loading MathJax using one local file*
- *Synchronizing with MathJax using promises*
- *Reseting TeX equation numbering*
- *Updating previously typeset content*
- *Looking up the math on the page*
- *Loading MathJax only on pages with math*
- *Automatic Section Numbering*
- *A replacement for the NativeMML output jax*
- *Backward Compatibility for TeX input*
- *Locating MathJax v2 math script tags*

CHAPTER 13

Getting Started with Node

This page is still under construction.

See the MathJax node demos for examples of how to use MathJax from a *node* application. These are categorized into three groups

- Examples using MathJax components the simple way
- Examples using MathJax components via the startup module
- Examples using MathJax components loaded by hand
- Examples using MathJax modules directly.

More information will be coming to this section in the future.

CHAPTER 14

Three Ways to Use MathJax in Node

14.1 Using MathJax Components in Node

This page is still under construction.

It is possible to use MathJax in a *node* application in essentially the same was that it is used in a browser. in particular, you can load MathJax components and configure MathJax using a global `MathJax` object and loading the *startup* component or a *combined component* file via node's `require()` command.

See the MathJax node demos for examples of how to use MathJax from a *node* application. in particular, see the component-based examples for illustrations of how to use MathJax components in a *node* application.

More information will be coming to this section in the future.

14.2 Loading Components by Hand in NodeJS

This page is still under construction.

In a *node* application, you can load components individually yourself via node's `require()` command, rather than relying on MathJax *loader*, which operates asynchronously. This gives you the ability to work with MathJax synchronously (i.e., without the need to use promises). It also gives you more complete control over the loading of components, though in this case you do need to take care to load dependencies yourself, and to make sure the components are loaded in the right order.

This approach lets you take advantage of using the convenient packaging of MathJax into individual components, the configuration of MathJax through the global `MathJax` variable, and its automatic creation of objects and methods by the *startup* component, while still allowing you to work completely synchronously with the MathJax code. (Or you can still use promises as well — it's up to you!)

See the MathJax node demos for examples of how to use MathJax from a *node* application. in particular, see the preloading examples for illustrations of how to load MathJax components by hand in a *node* application.

More information will be coming to this section in the future.

14.3 Linking to MathJax Directly in Node

This page is still under construction.

Node applications can link directly to MathJax source code, rather than using *MathJax components*. This provides the lowest-level access to the MathJax code, and is more complicated than using components, but it gives you the greatest flexibility as well.

See the MathJax node demos for examples of how to use MathJax from a *node* application. in particular, see the non-component-based examples for illustrations of how to use MathJax modules directly in a *node* application, rather than using the pre-packaged components.

More information will be coming to this section in the future.

CHAPTER 15

Examples of MathJax in Node

This page is still under construction.

See the MathJax node demos for examples of how to use MathJax from a *node* application. These are categorized into three groups

- Examples using MathJax components via the startup module
- Examples using MathJax components loaded by hand
- Examples using MathJax modules directly.

More information will be coming to this section in the future.

CHAPTER 16

TeX and LaTeX Support

The support for *TeX* and *LaTeX* in MathJax involves two functions: the first looks for mathematics within your web page (indicated by math delimiters like `$$...$$`) and marks the mathematics for later processing by MathJax, and the second is what converts the TeX notation into MathJax's internal format, where one of MathJax's output processors then displays it in the web page. In MathJax version 2, these were separated into distinct components (the `tex2jax` preprocessor and the TeX input jax), but in version 3, the `tex2jax` functions have been folded into the TeX input jax.

The TeX input jax can be configured to look for whatever markers you want to use for your math delimiters. See the *TeX configuration options* section for details on how to customize the delimiters, and other options for TeX input.

The TeX input processor handles conversion of your mathematical notation into MathJax's internal format (which is essentially MathML), and so acts as a TeX to MathML converter. The TeX input processor can also be customized through the use of extensions that define additional functionality (see the *TeX and LaTeX extensions* section).

Note: if you are not familiar with TeX/LaTeX, a good starting point is the LaTeX Wiki book.

16.1 Differences from Actual TeX

Since MathJax renders for the web and TeX is a print layout engine, there are natural limitations to which parts of TeX can be supported in a reasonable way. Accordingly, there are several differences between "real" TeX/LaTeX systems and MathJax's TeX Input.

First and foremost, the TeX input processor implements **only** the math-mode macros of TeX and LaTeX, not the text-mode macros. MathJax expects that you will use standard HTML tags to handle formatting the text of your page; MathJax only handles the mathematics. So, for example, MathJax does not implement `\emph` or `\begin{enumerate}...\end{enumerate}` or other text-mode macros or environments. You must use HTML to handle such formatting tasks. If you need a LaTeX-to-HTML converter, you should consider other options.

There are two exception to this rule. First, MathJax supports the `\ref` macro outside of math-mode. Second, MathJax supports some macros that add text within math-mode (such as `\text{}`) as well as `$...$` and `\(...\)` to switch back into math-mode, along with `\$` to escape a dollar sign. MathJax does not perform other macros inside these text blocks, however, in general. So, for example, `\text{some \textbf{bold} text}` will produce the output "some \textbf{bold} text", not "some **bold** text".

There is an extension (new in version 3.1) that implements a numnber of text-mode macros within the `\text{}` macro and other ones that produce text-mode material. See the *textmacros* documentation for details.

Second, some features in MathJax might be necessarily limited. For example, MathJax only implements a limited subset of the `array` environment's preamble; i.e., only the `l`, `r`, `c`, and `|` characters alongside `:` for dashed lines — everything else is ignored.

16.2 TeX and LaTeX math delimiters

By default, the TeX processor uses the LaTeX math delimiters, which are `\(...\)` for in-line math, and `\[...\]` for displayed equations. It also recognizes the TeX delimiters `$$...$$` for displayed equations, but it does **not** define `$...$` as in-line math delimiters. That is because dollar signs appear too often in non-mathematical settings, which could cause some text to be treated as mathematics unexpectedly. For example, with single-dollar delimiters, "... the cost is $2.50 for the first one, and $2.00 for each additional one ..." would cause the phrase "2.50 for the first one, and" to be treated as mathematics since it falls between dollar signs. For this reason, if you want to use single dollar signs for in-line math mode, you must enable that explicitly in your configuration:

```
window.MathJax = {
  tex: {
    inlineMath: [['$', '$'], ['\\(', '\\)']]
  }
};
```

You can use `\$` to prevent a dollar sign from being treated as a math delimiter within the text of your web page, e.g., use "... the cost is $2.50 for the first one, and $2.00 for each additional one ..." to prevent these dollar signs from being used as math delimiters in a web page where dollar signs have been configured to be in-line delimiters.

Note that, as opposed to true LaTeX, MathJax processes all environments when wrapped inside math delimiters, even those like `\begin{equation}...\end{equation}` that are supposed to be used to initiate math mode. By default, MathJax will also render all environments outside of delimiters, e.g., `\begin{matrix}...\end{matrix}` would be processed even if it is not in math mode delimiters, though you are encouraged to use proper delimiters for these cases to make your files more compatible with actual LaTeX. This functionality can be controlled via the `processEnvironments` option in the *tex configuration options*.

See the *tex configuration options* page, for additional configuration parameters that you can specify for the TeX input processor.

16.3 TeX and LaTeX in HTML documents

16.3.1 HTML Special Characters

Keep in mind that your mathematics is part of an HTML document, so you need to be aware of the special characters used by HTML as part of its markup. There cannot be HTML tags within the math delimiters (other than `<br>`, `<wbr>`, and HTML comments) as TeX-formatted math does not include HTML tags. Also, since the mathematics is initially given as text in the page, you need to be careful that your mathematics doesn't look like HTML tags to the browser, which parses the page before MathJax gets to see it. In particular, that means that you have to be careful about things like less-than and greater-than signs (< and >), and ampersands (`&`), which have special meaning to web browsers. For example,

```
... when $x<y$ we have ...
```

will cause a problem, because the browser will think `<y` is the beginning of a tag named `y` (even though there is no such tag in HTML). When this happens, the browser will think the tag continues up to the next > in the document

(typically the end of the next actual tag in the HTML file), and you may notice that you are missing part of the text of the document. In the example above, the "<y" and "we have ..." will not be displayed because the browser thinks it is part of the tag starting at <y. This is one indication you can use to spot this problem; it is a common error and should be avoided.

Usually, it is sufficient simply to put spaces around these symbols to cause the browser to avoid them, so

```
... when $x < y$ we have ...
```

should work. Alternatively, you can use the HTML entities <, > and & to encode these characters so that the browser will not interpret them, but MathJax will. E.g.,

```
... when $x &lt; y$ we have ...
```

Finally, there are \lt and \gt macros defined to make it easier to enter < and > using TeX-like syntax:

```
... when $x \lt y$ we have ...
```

Again, keep in mind that the browser interprets your text before MathJax does.

16.3.2 Interactions with Content-Management Systems

Another source of difficulty is when MathJax is used in content-management systems that have their own document processing commands that are interpreted before the HTML page is created. For example, many blogs and wikis use formats like Markdown to allow you to create the content of your pages. In Markdown, the underscore is used to indicate italics, and this usage will conflict with MathJax's use of the underscore to indicate a subscript. Since Markdown is applied to the page first, it may convert your subscript markers into italics (inserting <i> or tags into your mathematics, which will cause MathJax to ignore the math).

Such systems need to be told not to modify the mathematics that appears between math delimiters. That usually involves modifying the content-management system itself, which is beyond the means of most page authors. If you are lucky, someone else will already have done this for you, and you may be able to find a MathJax plugin for your system using a web search.

If there is no plugin for your system, or if the plugin doesn't handle the subtleties of isolating the mathematics from the other markup that it supports, then you may have to "trick" the content-management system into leaving your mathematics untouched. Most content-management systems provide some means of indicating text that should not be modified ("verbatim" text), often for giving code snippets for computer languages. You may be able use that to enclose your mathematics so that the system leaves it unchanged and MathJax can process it. For example, in Markdown, the back-tick (`) is used to mark verbatim text, so

```
... we have `\(x_1 = 132\)` and `\(x_2 = 370\)` and so ...
```

may be able to protect the underscores from being processed by Markdown.

Alternatively, some content-management systems use the backslash (\) as a special character for "escaping" other characters, and you may be able to use that to prevent it from converting underscores to italics. That is, you might be able to use

```
... we have $x\_1 = 132$ and $x\_2 = 370$ and so ...
```

to avoid the underscores from making 1 = 132$ and $x into italics.

If your system uses backslashes in this way, that can help with italics, but it also causes difficulties in other ways. Because TeX uses this character to indicate a macro name, you need to be able to pass a backslash along to the page so that MathJax will be able to identify macro names; but if the content-management system is using them as escapes, it will remove the backslashes as part of its processing, and they won't make it into the final web page. In such systems,

you may have to double the backslashes in order to obtain a single backslash in your HTML page. For example, you may have to do

```
\\begin{array}{cc}
  a & b \\\\
  c & c
\\end{array}
```

to get an array with the four entries *a*, *b*, *c*, and *d* in two rows. Note in particular that if you want `\\` you will have to double *both* backslashes, giving `\\\\`.

That may also affect how you enter the math delimiters. Since the defaults are `\(...\)` and `\[...\]`, if your system uses `\` as an escape of its own, you may need to use `\\(...\\)` and `\\[...\\]` instead in order to get `\(...\)` and `\[...\]` into the page where MathJax can process it.

Finally, if you have enabled single dollar signs as math delimiters and you want to include a literal dollar sign in your web page (one that doesn't represent a math delimiter), you will need to prevent MathJax from using it as a math delimiter. If you also enable the `processEscapes` configuration parameter (it is enabled by default), then you can use `\$` in the text of your page to get a dollar sign (without the backslash) in the end. Alternatively, you can use something like `<span>$</span>` to isolate the dollar sign so that MathJax will not use it as a delimiter.

16.4 Defining TeX macros

You can use the `\def`, `\newcommand`, `\renewcommand`, `\newenvironment`, `\renewenvironment`, and `\let` commands to create your own macros and environments. Unlike actual TeX, however, in order for MathJax to process such definitions, they must be enclosed in math delimiters (since MathJax only processes macros in math-mode). For example

```
\(
   \def\RR{{\bf R}}
   \def\bold#1{{\bf #1}}
\)
```

would define `\RR` to produce a bold-faced "R", and `\bold{...}` to put its argument into bold face. Both definitions would be available throughout the rest of the page.

You can include macro definitions in the `macros` section of the `tex` blocks of your configuration, but they must be represented as javascript objects. For example, the two macros above can be pre-defined in the configuration by

```
window.MathJax = {
  tex: {
    macros: {
      RR: "{\\bf R}",
      bold: ["{\\bf #1}", 1]
    }
  }
};
```

Here you give the macro as a `name: value` pair, where the `name` is the name of the control sequence (without the backslash) that you are defining, and `value` is either the replacement string for the macro (when there are no arguments) or an array consisting of the replacement string followed by the number of arguments for the macro and, optionally, default values for optional arguments.

Note that the replacement string is given as a javascript string literal, and the backslash has special meaning in javascript strings. So to get an actual backslash in the string you must double it, as in the examples above.

Similarly, you can create new environments with the `environments` section of the `tex` block of your configuration.

See *configmacros Options* for more details on the `macros` and `environments` configuration blocks.

16.5 Automatic Equation Numbering

The TeX input processing in MathJax can be configured to add equation numbers to displayed equations automatically. This functionality is turned off by default, but it is easy to configure MathJax to produce automatic equation numbers by adding:

```
window.MathJax = {
  tex: {
    tags: 'ams'
  }
};
```

to tell the TeX input processor to use the AMS numbering rules (where only certain environments produce numbered equations, as they would be in LaTeX). It is also possible to set the tagging to `'all'`, so that every displayed equation will get a number, regardless of the environment used.

You can use `\notag` or `\nonumber` to prevent individual equations from being numbered, and `\tag{}` can be used to override the usual equation number with your own symbol instead (or to add an equation tag even when automatic numbering is off).

Note that the AMS environments come in two forms: starred and unstarred. The unstarred versions produce equation numbers (when `tags` is set to `'ams'`) and the starred ones don't. For example

```
\begin{equation}
   E = mc^2
\end{equation}
```

will be numbered, while

```
\begin{equation*}
   e^{\pi i} + 1 = 0
\end{equation*}
```

will not be numbered (when `tags` is `'ams'`).

You can use `\label` to give an equation an identifier that you can use to refer to it later, and then use `\ref` or `\eqref` within your document to insert the actual equation number at that location, as a reference. For example,

```
In equation \eqref{eq:sample}, we find the value of an
interesting integral:

\begin{equation}
  \int_0^\infty \frac{x^3}{e^x-1}\,dx = \frac{\pi^4}{15}
  \label{eq:sample}
\end{equation}
```

includes a labeled equation and a reference to that equation. Note that references can come before the corresponding formula as well as after them.

You can configure the way that numbers are displayed and how the references to them by including the `tagformat` extension, and setting options within the `tagformat` block of your `tex` configuration. See the *tagformat* extension for more details.

If you are using automatic equation numbering and modifying the page dynamically, you can run into problems due to duplicate labels. See *Resetting Automatic Equation Numbering* for how to address this.

16.6 TeX and LaTeX extensions

While MathJax includes nearly all of the Plain TeX math macros, and many of the LaTeX macros and environments, not everything is implemented in the core TeX input processor. Some less-used commands are defined in extensions to the TeX processor. MathJax will load some extensions automatically when you first use the commands they implement (for example, the `\color` macro is implemented in the `color` extension, but MathJax loads this extension itself when you use that macro). While most extensions are set up to load automatically, there are a few that you would need to load explicitly yourself. See the *autoload* extension below for how to configure which extensions to autoload.

16.6.1 Loading TeX Extensions

To enable one of the TeX extensions you need to do two things: load the extension, and configure TeX to include it in its package setup. For the first, to load an extension as a component, add its name to the `load` array in the `loader` block of your MathJax configuration. For example, to load the `color` extension, add `'[tex]/color'` to the load array, as in the example below. To do the second, add the extension name to `packages` array in the `tex` block of your configuration. You can use the special `'[+]'` notation to append it to the default packages (so you don't need to know what they are). For example:

```
window.MathJax = {
  loader: {load: ['[tex]/color']},
  tex: {packages: {'[+]': ['color']}}
};
```

will load the `color` extension and configure the TeX input jax to enable it.

A number of extensions are already loaded and configured in the components that contain the TeX extension. The `input/tex`, and the combined components containing `tex` and not ending in `-full` include the `ams`, `newcommand`, `noundefined`, `require`, `autoload`, and `configmacros` extensions, with the other extensions being autoloaded as needed. The `input/tex-base` component has no extensions loaded, while the `input/tex-full` and the combined extensions ending in `-full` load all the extensions.

If you load a component that has an extension you don't want to use, you can disable it by removing it from the `package` array in the `tex` block of your MathJax configuration. For example, to disable `\require` and autoloading of extensions, use

```
window.MathJax = {
  tex: {packages: {'[-]': ['require', 'autoload']}}
};
```

if you are using, for example, the `tex-chtml.js` combined component file.

16.6.2 Loading Extensions at Run Time

You can also load these extensions from within a math expression using the non-standard `\require{extension}` macro. For example

```
\(\require{color}\)
```

would load the *color* extension into the page. This way you you can load extensions into pages that didn't load them in their configurations (and prevents you from having to load all the extensions into all pages even if they aren't used).

16.6.3 Configuring TeX Extensions

Some extensions have options that control their behavior. For example, the *color* extension allows you to set the padding and border-width used for the `\colorbox` and `\fcolorbox` macros. Such extensions are configured using a block within the `tex` configuration of your MathJax configuration object. The block has the same name as the extension, and contains the options you want to set for that extension. For example,

```
window.MathJax = {
  loader: {load: ['[tex]/color']},
  tex: {
    packages: {'[+]': ['color']},
    color: {
      padding: '5px'
    }
  }
};
```

would set the padding for `\colorbox` to be 5 pixels.

See the *Configuring MathJax* section for details about how to configure MathJax in general, and *TeX Extension Options* for the options for individual extensions.

For extensions that are not loaded explicitly but may be loaded via the *autoload* package or the `\require` macro, you can't include the configuration within the `tex` block, because MathJax will not know the options that are available (since the extension hasn't been loaded yet). In that case, move the configuration block to the top level of the MathJax configuration object and prefix it with `[tex]/`, as in:

```
window.MathJax = {
  '[tex]/color': {
    padding: '5px'
  }
};
```

16.7 The TeX/LaTeX Extension List

The main extensions are described below:

16.7.1 action

The *action* extension gives you access to the MathML `<maction>` element. It defines three new non-standard macros:

`\mathtip{math}{tip}`
: Use `tip` (in math mode) as tooltip for `math`.

`\texttip{math}{tip}`
: Use `tip` (plain text) as tooltip for `math`.

`\toggle{math1}{math2}...\endtoggle`
: Show `math1`, and when clicked, show `math2`, and so on. When the last one is clicked, go back to `math1`.

This extension is loaded automatically when the *autoload* extension is used. To load the *action* extension explicitly, add `'[tex]/action'` to the `load` array of the `loader` block of your MathJax configuration, and add `'action'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/action']},
  tex: {packages: {'[+]': ['action']}}
};
```

Alternatively, use `\require{action}` in a TeX expression to load it dynamically from within the math on the page, if the *require* package is loaded.

16.7.2 ams

The *ams* extension implements AMS math environments and macros, and macros for accessing the characters in the AMS symbol fonts. This extension is already loaded in all the components that include the TeX input jax, other than `input/tex-base`. See the *list of control sequences* for details about what commands are implemented in this extension.

To load the *ams* extension explicitly (when using `input/tex-base` for example), add `'[tex]/ams'` to the `load` array of the `loader` block of your MathJax configuration, and add `'ams'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/ams']},
  tex: {packages: {'[+]': ['ams']}}
};
```

Alternatively, use `\require{ams}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

Since the *ams* extension is included in the combined components that contain the TeX input jax, it will already be in the package list. In that case, if you want to disable it, you can remove it:

```
window.MathJax = {
  tex: {packages: {'[-]': ['ams']}}
};
```

16.7.3 amscd

The *amscd* extensions implements the *CD* environment for commutative diagrams. See the AMScd guide for more information on how to use the *CD* environment.

This extension is loaded automatically when the *autoload* extension is used. To load the *amscd* extension explicitly, add `'[tex]/amscd'` to the `load` array of the `loader` block of your MathJax configuration, and add `'amscd'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/amscd']},
  tex: {packages: {'[+]': ['amscd']}}
};
```

Alternatively, use `\require{amscd}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

amscd Options

Adding the *amscd* extension to the `packages` array defines an `amscd` sub-block of the `tex` configuration block with the following values:

```
MathJax = {
  tex: {
    amscd: {
      colspace: '5pt',
      rowspace: '5pt',
      harrowsize: '2.75em',
      varrowsize: '1.75em',
      hideHorizontalLabels: false
    }
  }
};
```

`colspace: '5pt'`
: This gives the amount of space to use between columns in the commutative diagram.

`rowspace: '5pt'`
: This gives the amount of space to use between rows in the commutative diagram.

`harrowsize: '2.75em'`
: This gives the minimum size for horizontal arrows in the commutative diagram.

`varrowsize: '1.75em'`
: This gives the minimum size for vertical arrows in the commutative diagram.

`hideHorizontalLabels: false`
: This determines whether horizontal arrows with labels above or below will use `\smash` in order to hide the height of the labels. (Labels above or below horizontal arrows can cause excess space between rows, so setting this to `true` can improve the look of the diagram.)

16.7.4 autoload

The *autoload* extension predefines all the macros from the extensions that haven't been loaded already so that they automatically load the needed extension when they are first used, with the exception of the *physics* package, since it redefines standard macros, and the *ams* package, due to the large number of macros it contains.

The *autoload* extension is loaded in all the components that include the TeX input jax, other than `input/tex-base`. That means that the TeX input jax essentially has access to all the extensions, even if they aren't loaded initially, and you should never have to use `\require` or load other extensions (except *physics*) explicitly unless you want to.

You can control which extensions *autoload* will load using the `autoload` object in the `tex` block of your MathJax configuration. This object contains *key: value* pairs where the *key* is the name of an extension, and *value* is an array listing the macro names that cause that extension to be loaded. If environments can also cause the extension to be loaded, *value* is an array consisting of two sub-arrays, the first being the names of the macros that cause the extension to autoload, and the second being the names of the environments that cause the extension to be loaded.

For example,

```
window.MathJax = {
  tex: {
    autoload: {
      verb: ['verb']
    }
```

(continues on next page)

(continued from previous page)

```
  }
};
```

says that the `\verb` command should load the *verb* extension when it is first used.

If the array is empty, then that extension will not be loaded, so to prevent *autoload* from loading an extension, assign it the empty array. E.g.,

```
window.MathJax = {
  tex: {
    autoload: {
      verb: []
    }
  }
};
```

says that the *verb* extension will not be autoloaded.

Note: The *autoload* extension defines `\color` to be the one from the *color* extension (the LaTeX-compatible one rather than the non-standard MathJax version). If you wish to use the non-standard version-2 `\color` macro from the *colorv2* extension instead, use the following:

```
window.MathJax = {
  tex: {
    autoload: {
      color: [],
      colorv2: ['color']
    }
  }
};
```

This extension is already loaded in all the components that include the TeX input jax, other than `input/tex-base`. To load the *autoload* extension explicitly (when using `input/tex-base` for example), add `'[tex]/autoload'` to the `load` array of the `loader` block of your MathJax configuration, and add `'autoload'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/autoload']},
  tex: {packages: {'[+]': ['autoload']}}
};
```

Since the *autoload* extension is included in the combined components that contain the TeX input jax, it will already be in the package list. In that case, if you want to disable it, you can remove it:

```
window.MathJax = {
  tex: {packages: {'[-]': ['autload']}}
};
```

---

autoload Options

Adding the *autoload* extension to the `packages` array defines an `autoload` sub-block to the `tex` configuration block. This block contains *key: value* pairs where the *key* is a TeX package name, and the *value* is an array of macros

that cause that package to be loaded, or an array consisting of two arrays, the first giving names of macros and the second names of environments; the first time any of them are used, the extension will be loaded automatically.

The default autoload definitions are the following:

```
MathJax = {
  tex: {
    autoload: expandable({
      action: ['toggle', 'mathtip', 'texttip'],
      amscd: [[], ['CD']],
      bbox: ['bbox'],
      boldsymbol: ['boldsymbol'],
      braket: ['bra', 'ket', 'braket', 'set', 'Bra', 'Ket', 'Braket', 'Set', 'ketbra',
↪ 'Ketbra'],
      cancel: ['cancel', 'bcancel', 'xcancel', 'cancelto'],
      color: ['color', 'definecolor', 'textcolor', 'colorbox', 'fcolorbox'],
      enclose: ['enclose'],
      extpfeil: ['xtwoheadrightarrow', 'xtwoheadleftarrow', 'xmapsto',
                 'xlongequal', 'xtofrom', 'Newextarrow'],
      html: ['href', 'class', 'style', 'cssId'],
      mhchem: ['ce', 'pu'],
      newcommand: ['newcommand', 'renewcommand', 'newenvironment', 'renewenvironment',
↪ 'def', 'let'],
      unicode: ['unicode'],
      verb: ['verb']
    }
  }
};
```

To prevent an extension from autoloading, set its value to an empty array. E.g., to not autoload the *color* extension, use

```
MathJax = {
  tex: {
    autoload: expandable({
      color: []
    }
  }
};
```

If you define your own extensions, and they have a prefix other than `[tex]`, then include that in the extension name. For instance,

```
MathJax = {
  tex: {
    autoload: expandable({
      '[extensions]/myExtension' : ['myMacro', 'myOtherMacro']
    }
  }
};
```

See the *Loader Options* section for details about how to define your own prefixes, like the `[extensions]` prefix used here.

16.7.5 bbox

The *bbox* extension defines a new macro for adding background colors, borders, and padding to your math expressions.

\bbox[options]{math}
puts a bounding box around `math` using the provided `options`. The options can be one of the following:

1. A color name used for the background color.
2. A dimension (e.g., `2px`) to be used as a padding around the mathematics (on all sides).
3. Style attributes to be applied to the mathematics (e.g., `border: 1px solid red`).
4. A combination of these separated by commas.

Here are some examples:

```
\bbox[red]{x+y}       % a red box behind x+y
\bbox[2pt]{x+1}       % an invisible box around x+y with 2pt of extra space
\bbox[red,2pt]{x+1}   % a red box around x+y with 2pt of extra space
\bbox[5px, border: 2px solid red]
                      % a 2px red border around the math 5px away
```

This extension is loaded automatically when the *autoload* extension is used. To load the *bbox* extension explicitly, add `'[tex]/bbox'` to the `load` array of the `loader` block of your MathJax configuration, and add `'bbox'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/bbox']},
  tex: {packages: {'[+]': ['bbox']}}
};
```

Alternatively, use `\require{bbox}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

16.7.6 boldsymbol

The *boldsymbol* extension defines the `\boldsymbol` LaTeX macro that produces a bold version of its argument, provided bold versions of the required characters are available.

This extension is loaded automatically when the *autoload* extension is used. To load the *boldsymbol* extension explicitly (when using `input/tex-base` for example), add `'[tex]/boldsymbol'` to the `load` array of the `loader` block of your MathJax configuration, and add `'boldsymbol'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/boldsymbol']},
  tex: {packages: {'[+]': ['boldsymbol']}}
};
```

Alternatively, use `\require{boldsymbol}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

16.7.7 braket

The *braket* extension defines the following macros for producing the bra-ket notation and set notation used in quantum mechanics

\bra{math}
\ket{math}
\braket{math}
\set{math}

\Bra{math}
\Ket{math}
\Braket{math}
\Set{math}

and the non-standard macros

\ketbra{math}
\Ketbra{math}

See the documentation for the LaTeX braket package for details of how these are used.

This extension is loaded automatically when the *autoload* extension is used. To load the *braket* extension explicitly (when using `input/tex-base` for example), add `'[tex]/braket'` to the `load` array of the `loader` block of your MathJax configuration, and add `'braket'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/braket']},
  tex: {packages: {'[+]': ['braket']}}
};
```

Alternatively, use `\require{braket}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

16.7.8 bussproofs

The *bussproofs* extension implements the `bussproofs` style package from LaTeX. See the CTAN page for more information and documentation of *bussproofs*.

Note that there are a couple of important differences between the use of the package in MathJax compared to actual LaTeX. First, proofs always have to be in a *prooftree* environment, i.e., inference macros are only recognised if they are enclosed in `\begin{prooftree}` and `\end{prooftree}`. Consequently the `\DisplayProof` command is not necessary.

Second, unlike in the LaTeX package, options for abbreviated inference rule macros do not have to be manually set. All abbreviated macros are directly available. Thus commands like `\BinaryInfC` and `\BIC` can be used immediately and interchangeably.

For example:

```
\begin{prooftree}
\AxiomC{}
\RightLabel{Hyp$^{1}$}
\UnaryInfC{$P$}
\AXC{$P\to Q$}
\RL{$\to_E$}
\BIC{$Q^2$}
\AXC{$Q\to R$}
\RL{$\to_E$}
\BIC{$R$}
\AXC{$Q$}
\RL{Rit$^2$}
\UIC{$Q$}
\RL{$\wedge_I$}
\BIC{$Q\wedge R$}
\RL{$\to_I$$^1$}
\UIC{$P\to Q\wedge R$}
\end{prooftree}
```

Also note that the *bussproofs* commands for sequent calculus derivations are not yet fully implemented.

This extension is loaded automatically when the *autoload* extension is used. To load the *bussproofs* extension explicitly, add `'[tex]/bussproofs'` to the `load` array of the `loader` block of your MathJax configuration, and add `'bussproofs'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/bussproofs']},
  tex: {packages: {'[+]': ['bussproofs']}}
};
```

Alternatively, use `\require{bussproofs}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

16.7.9 cancel

The *cancel* extension defines the following macros:

`\cancel{math}`
: Strikeout `math` from lower left to upper right.

`\bcancel{math}`
: Strikeout `math` from upper left to lower right.

`\xcancel{math}`
: Strikeout `math` with an "X".

`\cancelto{value}{math}`
: Strikeout `math` with an arrow going to `value`.

This extension is loaded automatically when the *autoload* extension is used. To load the *cancel* extension explicitly, add `'[tex]/cancel'` to the `load` array of the `loader` block of your MathJax configuration, and add `'cancel'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/cancel']},
  tex: {packages: {'[+]': ['cancel']}}
};
```

Alternatively, use `\require{cancel}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

16.7.10 color

The *color* extension defines the `\color` macro as in the LaTeX *color* package, along with `\colorbox`, `\fcolorbox`, and `\definecolor`. It declares the standard set of colors (*Apricot*, *Aquamarine*, *Bittersweet*, and so on), and provides the RGB, rgb, and grey-scale color spaces in addition to named colors.

This extension is loaded automatically when the *autoload* extension is used. To load the *color* extension explicitly, add `'[tex]/color'` to the `load` array of the `loader` block of your MathJax configuration, and add `'color'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/color']},
  tex: {packages: {'[+]': ['color']}}
};
```

Alternatively, use `\require{color}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

Note: In version 2, a non-standard `\color` macro was the default implementation, but in version 3, the standard LaTeX one is now the default. The difference between the two is that the standard `\color` macro is a switch (everything that follows it is in the new color), whereas the non-standard version 2 `\color` macro takes an argument that is the mathematics to be colored. That is, in version 2, you would do

```
\color{red}{x} + \color{blue}{y}
```

to get a red *x* added to a blue *y*. But in version 3 (and in LaTeX itself), you would do

```
{\color{red} x} + {\color{blue} y}
```

If you want the old version 2 behavior, use the *colorv2* extension instead.

---

color Options

Adding the *color* extension to the `packages` array defines a `color` sub-block of the `tex` configuration block with the following values:

```
MathJax = {
  tex: {
    color: {
      padding: '5px',
      borderWidth: '2px'
    }
  }
};
```

`padding: '5px'`
: This gives the padding to use for color boxes with background colors.

`borderWidth: '2px'`
: This gives the border width to use with framed color boxes produced by `\fcolorbox`.

16.7.11 colorv2

The *colorv2* extension defines the `\color` macro to be the non-standard macro that is the default in MathJax version 2, namely, it takes two arguments, one the name of the color (or an HTML color of the form `#RGB` or `#RRGGBB`), and the second the math to be colored. This is in contrast to the standard LaTeX `\color` command, which is a switch that changes the color of everything that follows it.

This extension is **not** loaded automatically when the *autoload* extension is used. To load the *color* extension explicitly, add `'[tex]/color'` to the `load` array of the `loader` block of your MathJax configuration, and add `'color'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/colorv2']},
  tex: {packages: {'[+]': ['color']}}
};
```

or, use `\require{colorv2}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

Alternatively, you can configure the *autoload* package to load *colorv2* when `\color` is used rather than the (LaTeX-compatible) *color* extension:

```
window.MathJax = {
  tex: {
    autoload: {
      color: [],            // don't autoload the color extension
      colorv2: ['color']    // autoload colorv2 on the first use of \color
    }
  }
};
```

16.7.12 configmacros

The *configmacros* extension provides the `macros` and `environments` configuration options for the `tex` block of your MathJax configuration. This allows you to predefine custom macros end environments for your page using javascript. For example,

```
window.MathJax = {
  tex: {
    macros: {
      RR: "{\\bf R}",
      bold: ["{\\bf #1}", 1]
    },
    environments: {
      braced: ["\\left\\{", "\\right\\}"]
    }
  }
};
```

defines a macro `\RR` that produces a bold "R", while `\bold{math}` typesets the `math` using the bold font (see *Defining TeX macros* for more information). It also creates the `braced` environment that puts `\left\{` and `\right\}` around its contents.

This extension is already loaded in all the components that include the TeX input jax, other than `input/tex-base`. To load the *configmacros* extension explicitly (when using `input/tex-base` for example), add `'[tex]/configmacros'` to the `load` array of the `loader` block of your MathJax configuration, and add `'configmacros'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/configmacros']},
  tex: {packages: {'[+]': ['configmacros']}}
};
```

Since the *configmacros* extension is included in the combined components that contain the TeX input jax, it may already be in the package list. In that case, if you want to disable it, you can remove it:

```
window.MathJax = {
  tex: {packages: {'[-]': ['configmacros']}}
};
```

configmacros Options

The *configmacros* extension adds a `macros` option to the `tex` block that lets you pre-define macros, and the `environments` option that lets you pre-define your own environments.

`macros: {}`
: This lists macros to define before the TeX input processor begins. These are *name: value* pairs where the *name* gives the name of the TeX macro to be defined, and *value* gives the replacement text for the macro. The *value* can be a simple replacement string, or an array of the form *[value, n]*, where *value* is the replacement text and *n* is the number of parameters for the macro. The array can have a third entry: either a string that is the default value to give for an optional (bracketed) parameter when the macro is used, or an array consisting of template strings that are used to separate the various parameters. The first template must precede the first parameter, the second must precede the second, and so on until the final which must end the last parameter to the macro. See the examples below.

  Note that since the *value* is a javascript string, backslashes in the replacement text must be doubled to prevent them from acting as javascript escape characters.

  For example,

```
macros: {
  RR: '{\\bf R}',                    // a simple string replacement
  bold: ['\\boldsymbol{#1}',1] ,     // this macro has one parameter
  ddx: ['\\frac{d#2}{d#1}', 2, 'x'], // this macro has an optional parameter that␣
↪defaults to 'x'
  abc: ['(#1)', 1, [null, '\\cba']]  // equivalent to \def\abc#1\cba{(#1)}
}
```

  would ask the TeX processor to define four new macros: `\RR`, which produces a bold-face "R", and `\bold{...}`, which takes one parameter and sets it in the bold-face font, `\ddx`, which has an optional (bracketed) parameter that defaults to `x`, so that `\ddx{y}` produces `\frac{dy}{dx}` while `\ddx[t]{y}` produces `\frac{dy}{dt}`, and `\abc` that is equivalent to `\def\abc#1\cba{(#1)}`.

`environments: {}`
: This lists environments to define before the TeX input processor begins. These are *name: value* pairs where the *name* gives the name of the environment to be defined, and *value* gives an array that defines the material to go before and after the content of the environment. The array is of the form *[before, after, n, opt]* where *before* is the material that replaces the `\begin{name}`, *after* is the material that replaces `\end{name}`, *n* is the number of parameters that follow the `\begin{name}`, and *opt* is the default value used for an optional parameter that would follow `\begin{name}` in brackets. The parameters can be inserted into the *before* string using `#1`, `#2`, etc., where `#1` is the optional parameter, if there is one.

  Note that since the *before* and *after* values are javascript strings, backslashes in the replacement text must be doubled to prevent them from acting as javascript escape characters.

  For example,

```
environments: {
  braced: ['\\left\\{', '\\right\\}'],
  ABC: ['(#1)(#2)(', ')', 2, 'X']
}
```

  would define two environments, `braced` and `ABC`, where

```
\begin{braced} \frac{x}{y} \end{braced}
```

  would produce the fraction *x/y* in braces that stretch to the height of the fraction, while

```
\begin{ABC}{Z} xyz \end{ABC}
```

would produce `(X)(Z)(xyz)`, and

```
\begin{ABC}[Y]{Z} xyz \end{ABC}
```

would produce `(Y)(Z)(xyz)`.

16.7.13 enclose

The *enclose* extension gives you access to the MathML `<menclose>` element for adding boxes, ovals, strikethroughs, and other marks over your mathematics. It defines the following non-standard macro:

`\enclose{notation}[attributes]{math}`
: Where `notation` is a comma-separated list of MathML `<menclose>` notations (e.g., `circle`, `left`, `updiagonalstrike`, `longdiv`, etc.), `attributes` are MathML attribute values allowed on the `<menclose>` element (e.g., `mathcolor="red"`, `mathbackground="yellow"`), and `math` is the mathematics to be enclosed. See the MathML 3 specification for more details on `<menclose>`.

For example

```
\enclose{circle}[mathcolor="red"]{x}
\enclose{circle}[mathcolor="red"]{\color{black}{x}}
\enclose{circle,box}{x}
\enclose{circle}{\enclose{box}{x}}
```

This extension is loaded automatically when the *autoload* extension is used. To load the *enclose* extension explicitly, add `'[tex]/enclose'` to the `load` array of the `loader` block of your MathJax configuration, and add `'enclose'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/enclose']},
  tex: {packages: {'[+]': ['enclose']}}
};
```

Alternatively, use `\require{enclose}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

16.7.14 extpfeil

The *extpfeil* extension adds more macros for producing extensible arrows, including `\xtwoheadrightarrow`, `\xtwoheadleftarrow`, `\xmapsto`, `\xlongequal`, `\xtofrom`, and a non-standard `\Newextarrow` for creating your own extensible arrows. The latter has the form

`\Newextarrow{\cs}{lspace,rspace}{unicode-char}`
: where `\cs` is the new control sequence name to be defined, `lspace` and `rspace` are integers representing the amount of space (in suitably small units) to use at the left and right of text that is placed above or below the arrow, and `unicode-char` is a number representing a unicode character position in either decimal or hexadecimal notation.

For example

```
\Newextarrow{\xrightharpoonup}{5,10}{0x21C0}
```

defines an extensible right harpoon with barb up. Note that MathJax knows how to stretch only a limited number of characters, so you may not actually get a stretchy character this way. The characters that can be stretched may also depend on the font you have selected.

This extension is loaded automatically when the *autoload* extension is used. To load the *extpfeil* extension explicitly, add `'[tex]/extpfeil'` to the `load` array of the `loader` block of your MathJax configuration, and add `'extpfeil'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/extpfeil']},
  tex: {packages: {'[+]': ['extpfeil']}}
};
```

Alternatively, use `\require{extpfeil}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

16.7.15 html

The *html* extension gives you access to some HTML features like styles, classes, element ID's, and clickable links. It defines the following non-standard macros:

`\href{url}{math}`
: Makes `math` be a link to the page given by `url`. Note that the url is not processed by TeX, but is given as the literal url. In actual TeX or LaTeX, special characters must be escaped; so, for example, a url containing a `#` would need to use `\#` in the url in actual TeX. That is not necessary in MathJax, and if you do use `\#`, it will produce `/#` in the url (since the `\` will be inserted into the url verbatim, and browsers will convert that to `/` (thinking it is a DOS directory separator).

`\class{name}{math}`
: Attaches the CSS class `name` to the output associated with `math` when it is included in the HTML page. This allows your CSS to style the element.

`\cssId{id}{math}`
: Attaches an id attribute with value `id` to the output associated with `math` when it is included in the HTML page. This allows your CSS to style the element, or your javascript to locate it on the page.

`\style{css}{math}`
: Adds the give `css` declarations to the element associated with `math`.

For example:

```
x \href{why-equal.html}{=} y^2 + 1

(x+1)^2 = \class{hidden}{(x+1)(x+1)}

(x+1)^2 = \cssId{step1}{\style{visibility:hidden}{(x+1)(x+1)}}
```

Note: For the `\href` macro, the *url* parameter is not processed futher, as it is in actual TeX, so you do not need to quote special characters. For example, `\href{#section1}{x}` is fine, but `\href{\#section}{x}` will not work as expected.

This extension is loaded automatically when the *autoload* extension is used. To load the *html* extension explicitly, add `'[tex]/html'` to the `load` array of the `loader` block of your MathJax configuration, and add `'html'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/html']},
  tex: {packages: {'[+]': ['html']}}
};
```

Alternatively, use `\require{html}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

16.7.16 mhchem

The *mhchem* extensions implements the `\ce` and `\pu` chemical equation macros of the LaTeX *mhchem* package. See the mhchem home page for more information and documentation for *mhchem*.

For example

```
\ce{C6H5-CHO}
\ce{$A$ ->[\ce{+H2O}] $B$}
\ce{SO4^2- + Ba^2+ -> BaSO4 v}
```

This extension is loaded automatically when the *autoload* extension is used. To load the *mhchem* extension explicitly, add `'[tex]/mhchem'` to the `load` array of the `loader` block of your MathJax configuration, and add `'mhchem'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/mhchem']},
  tex: {packages: {'[+]': ['mhchem']}}
};
```

Alternatively, use `\require{mhchem}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

Note: The implementation of the *mhchem* extension was completely rewritten for MathJax by the author of the original LaTeX package. The older version was still available MathJax version 2.7, but it is no longer part of MathJax version 3. Only the newer version of *mhchem* is available.

16.7.17 newcommand

The *newcommand* extension provides the `\def`, `\newcommand`, `\renewcommand`, `\let`, `\newenvironment`, and `\renewenvironment` macros for creating new macros and environments in TeX. For example,

```
\(
   \def\RR{{\bf R}}
   \def\bold#1{{\bf #1}}
\)
```

defines a macro `\RR` that produces a bold "R", while `\bold{math}` typesets its argument using a bold font. See *Defining TeX macros* for more information.

This extension is already loaded in all the components that include the TeX input jax, other than `input/tex-base`. To load the *newcommand* extension explicitly (when using `input/tex-base` for example), add `'[tex]/newcommand'` to the `load` array of the `loader` block of your MathJax configuration, and add `'newcommand'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/newcommand']},
  tex: {packages: {'[+]': ['newcommand']}}
};
```

Alternatively, use `\require{newcommand}` in a TeX expression to load it dynamically from within the math on the page, if the *require* package is loaded.

Since the *nnewcommand* extension is included in the combined components that contain the TeX input jax, it may already be in the package list. In that case, if you want to disable it, you can remove it:

```
window.MathJax = {
  tex: {packages: {'[-]': ['newcommand']}}
};
```

16.7.18 noerrors

The *noerrors* extension prevents TeX error messages from being displayed and shows the original TeX code instead.

Note: In version 2 of MathJax, you could configure the CSS that applied to the display of the original TeX. In version 3, the original TeX is shown via an *merror* MathML element instead.

Note: In version 2, this extension was included in all the combined configuration files that contain the TeX input jax, but in MathJax version 3, you must load it explicitly if you want to use it.

To load the *noerrors* extension, add `'[tex]/noerrors'` to the `load` array of the `loader` block of your MathJax configuration, and add `'noerrors'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/noerrors']},
  tex: {packages: {'[+]': ['noerrors']}}
};
```

16.7.19 noundefined

The *noundefined* extension causes undefined control sequences to be shown as their macro names rather than generating error messages. So `$X_{\xyz}$` would display as an "X" with a subscript consisting of the text `\xyz` in red.

Note: In version 2, the styling for the undefined macro could be configured. In version 3, this is not yet implemented.

This extension is already loaded in all the components that include the TeX input jax, other than `input/tex-base`. To load the *ams* extension explicitly (when using `input/tex-base` for example), add `'[tex]/noundefined'` to the `load` array of the `loader` block of your MathJax configuration, and add `'noundefined'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/noundefined']},
```

(continues on next page)

(continued from previous page)

```
  tex: {packages: {'[+]': ['noundefined']}}
};
```

Since the *noundefined* extension is included in the combined components that contain the TeX input jax, it may already be in the package list. In that case, if you want to disable it, you can remove it:

```
window.MathJax = {
  tex: {packages: {'[-]': ['noundefined']}}
};
```

---

noundefined Options

Adding `'[tex]/noundefined'` to the `packages` array defines a `noundefined` sub-block of the `tex` configuration block with the following values:

```
MathJax = {
  tex: {
    noundefined: {
      color: 'red',
      background: '',
      size: ''
    }
  }
};
```

`color: 'red'`
: This gives the color to use for the text of the undefined macro name, or an empty string to make the color the same as the surrounding mathematics.

`background: ''`
: This gives the color to use for the background for the undefined macro name, or an empty srting to have no brackground color.

`size: ''`
: This gives the size to use for the undefined macro name (e.g., `90%` or `12px`), or an emtpy string to keep the size the same as the surrounding mathematics.

16.7.20 physics

The *physics* extension implements much of the LaTeX physics package, which defines simple, yet flexible macros for typesetting equations via:

- Automatic bracing
- Vector notation
- Derivatives
- Dirac bra-ket notation
- Matrix macros
- Additional trig functions and other convenient operators
- Flat fractions and other useful miscellaneous math macros

See the documentation for the LaTeX package for more information.

This package is not autoloaded, due to the fact that it redefines many standard macros, so you must request it explicitly if you want to use it. To load the *physics* extension, add `'[tex]/physics'` to the `load` array of the `loader` block of your MathJax configuration, and add `'physics'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/physics']},
  tex: {packages: {'[+]': ['physics']}}
};
```

Alternatively, use `\require{physics}` in a TeX expression to load it dynamically from within the math on the page, if the *require* package is loaded.

16.7.21 require

The *require* extension defines the non-standard `\require` macro that allows you to load extensions from within a math expression in a web page. For example:

```
\(\require{enclose} \enclose{circle}{x}\)
```

would load the *enclose* extension, making the following `\enclose` command available for use.

An extension only needs to be loaded once, and then it is available for all subsequent typeset expressions.

This extension is already loaded in all the components that include the TeX input jax, other than `input/tex-base`. To load the *require* extension explicitly (when using `input/tex-base` for example), add `'[tex]/require'` to the `load` array of the `loader` block of your MathJax configuration, and add `'require'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/require']},
  tex: {packages: {'[+]': ['require']}}
};
```

Since the *require* extension is included in the combined components that contain the TeX input jax, it may already be in the package list. In that case, if you want to disable it, you can remove it:

```
window.MathJax = {
  tex: {packages: {'[-]': ['require']}}
};
```

---

require Options

Adding the *require* extension to the `packages` array defines a `require` sub-block of the `tex` configuration block with the following values:

```
MathJax = {
  tex: {
    require: {
      allow: {
        base: false,
        'all-packages': false
```

(continues on next page)

(continued from previous page)

```
      },
      defaultAllow: true
    }
  };
```

allow: {...}
: This sub-object indicates which extensions can be loaded by `\require`. The keys are the package names, and the value is `true` to allow the extension to be loaded, and `false` to disallow it. If an extension is not in the list, the default value is given by `defaultAllow`, described below.

defaultAllow: true
: This is the value used for any extensions that are requested, but are not in the `allow` object described above. If set to `true`, any extension not listed in `allow` will be allowed; if `false`, only the ones listed in `allow` (with value `true`) will be allowed.

16.7.22 tagformat

The *tagformat* extension provides the ability to customize the format of the equation tags and automatic equation numbers. You do this by providing functions in the `tagformat` object of the `tex` block of your MathJax configuration. The functions you can provide are listed in the *tagformat Options* section.

To load the *tagformat* extension, add `'[tex]/tagformat'` to the `load` array of the `loader` block of your MathJax configuration, and add `'tagformat'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/tagformat']},
  tex: {packages: {'[+]': ['tagformat']}}
};
```

---

tagformat Options

Adding the *tagformat* extension to the `packages` array adds a `tagformat` sub-object to the `tex` configuration block with the following values:

```
tagformat: {
   number:  (n) => n.toString(),
   tag:     (tag) => '(' + tag + ')',
   id:      (id) => 'mjx-eqn:' + id.replace(/\s/g, '_'),
   url:     (id, base) => base + '#' + encodeURIComponent(id),
 }
```

number: function (n) {return n.toString()}
: A function that tells MathJax what tag to use for equation number `n`. This could be used to have the equations labeled by a sequence of symbols rather than numbers, or to use section and subsection numbers instead.

tag: function (n) {return '(' + n + ')'}
: A function that tells MathJax how to format an equation number for displaying as a tag for an equation. This is what appears in the margin of a tagged or numbered equation.

id: function (n) {return 'mjx-eqn:' + n.replace(/\s/g, '_')}
: A function that tells MathJax what ID to use as an anchor for the equation (so that it can be used in URL references).

url: function (id, base) {return base + '#' + encodeURIComponent(id)}
A function that takes an equation ID and base URL and returns the URL to link to it. The `base` value is taken from the *baseURL* value, so that links can be make within a page even if it has a `<base>` element that sets the base URL for the page to a different location.

Example: Section Numbering

This example shows one way to provide section numbers for the automatic equation numbers generated when the `tags` option in the `tex` configuration block is set to `'ams'` or `'all'`.

```
MathJax = {
  section: 1,
  tex: {
    tagformat: {
      number: (n) => MathJax.config.section + '.' + n,
      id: (tag) => 'eqn-id:' + tag
    }
  },
  startup: {
    ready() {
      MathJax.startup.defaultReady();
      MathJax.startup.input[0].preFilters.add(({math}) => {
        if (math.inputData.recompile) {
          MathJax.config.section = math.inputData.recompile.section;
        }
      });
      MathJax.startup.input[0].postFilters.add(({math}) => {
        if (math.inputData.recompile) {
          math.inputData.recompile.section = MathJax.config.section;
        }
      });
    }
  }
};
```

This arranges for automatic equation numbers to be of the form `1.n`, and uses ids of the form `eqn-id:1.n` as the `id` attribute of the tags within the web page. It also sets up pre- and post-filters for the TeX input jax that arrange for the section number to be properly handled for automatically numbered equations that contain forward references to later expressions. This example uses the modern function notation (using =>), but you could also use `function (n) {return ...}`.

You can adjust the section number using JavaScript by setting the `MathJax.config.section` variable. It is also possible to create TeX macros for controlling the section number. Here is one possibility:

```
MathJax = {
  startup: {
    ready() {
      const Configuration = MathJax._.input.tex.Configuration.Configuration;
      const CommandMap = MathJax._.input.tex.SymbolMap.CommandMap;
      new CommandMap('sections', {
        nextSection: 'NextSection',
        setSection: 'SetSection',
      }, {
        NextSection(parser, name) {
```

(continues on next page)

(continued from previous page)

```
          MathJax.config.section++;
          parser.tags.counter = parser.tags.allCounter = 0;
        },
        SetSection(parser, name) {
          const n = parser.GetArgument(name);
          MathJax.config.section = parseInt(n);
        }
      });
      Configuration.create(
        'sections', {handler: {macro: ['sections']}}
      );
      MathJax.startup.defaultReady();
    }
  }
};
```

Of course, you will want to merge this configuraiton in with the rest of your configuration options.

This makes two new macros available: `\nextSection`, which increments the section counter, and `\setSection{n}`, which sets the section number to `n`. Note that these must be issued within math delimiters in order for MathJax to process them. In order to prevent them from producing any output in your page, you could enclose them within a hidden element. For example,

```
<span style="display: hidden">\(\nextSection\)</span>
```

or something similar.

16.7.23 textmacros

The *textmacros* extension adds the ability to process some text-mode macros within `\text{}` and other macros that produce text-mode material. See the *Differences from Actual TeX* section for how text-mode is handled without this extension.

This extension is not loaded automatically, and can't be loaded via the *autoload* extension. To load the *textmacros* extension, add `'[tex]/textmacros'` to the `load` array of the `loader` block of your MathJax configuration, and add `'textmacros'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/textmacros']},
  tex: {packages: {'[+]': ['textmacros']}}
};
```

Alternatively, use `\require{textmacros}` in a TeX expression to load it dynamically from within the math on the page, if the *require* package is loaded.

Available Macros:

The macros available in text mode with this extension are listed below. In addition, any macro that is defined via `\def` or `\newcommand` or in the `macros` section of the `tex` configuration block will also be processed if they only contain macros from the list below.

Additional Special Characters

~	non-breaking space
`	open quote (use two for double quote)
'	close quote (use two for double quote)

Math Mode Delimiters

$	start/end math mode
\(	start math mode
\)	end math mode

Quoted Special Characters

\$	literal dollar sign
_	literal underscore
\%	literal percent
\{	literal open brace
\}	literal close brace
\ (backslash-space)	literal space
\&	literal ampersand
\#	literal hash mark
\\	literal backslash

Text Accents

\'	acute accent
\`	grave accent
\^	circumflex accent
\"	umlaut accent
\~	tilde accent
\=	macron accent
\.	over dot accent
\u	breve accent
\v	caron accent

Font Control

`\emph`	emphasized text
`\rm`	roman text
`\oldstyle`	oldstyle numerals
`\cal`	calligraphic text
`\it`	italic text
`\bf`	bold text
`\scr`	script text
`\frak`	Fraktur text
`\sf`	sans-serif text
`\tt`	typewriter text
`\Bbb`	blackboard-bold text
`\textrm`	roman text
`\textit`	italic text
`\textbf`	bold text
`\textsf`	sans-serif text
`\texttt`	typewriter text

Size Control

`\tiny`	very tiny size
`\Tiny`	tiny size
`\scriptsize`	size of super- and subscripts
`\small`	small size
`\normalsize`	standard size
`\large`	large size
`\Large`	larger sizse
`\LARGE`	very large size
`\huge`	even larger size
`\Huge`	largest size

Special Characters

`\dagger`	†
`\ddagger`	‡
`\S`	§

Spacing Commands

`\,`	thin space
`\:`	medium space
`\>`	medium space
`\;`	thick space
`\!`	negative thin space
`\enspace`	en-space
`\quad`	quad space
`\qquad`	double quad space
`\thinspace`	thin space
`\negthinspace`	negative thin space
`\hskip`	horizontal skip (by following amount)
`\hspace`	horizontal space (of a given size)
`\kern`	kern (by a given size)
`\rule`	line of a given width and height
`\Rule`	box with given dimensions
`\Space`	space with given dimensions

Color Commands

`\color`	set text color
`\textcolor`	set text color
`\colorbox`	make colored box
`\fcolorbox`	make framed colored box

HTML Commands

`\href`	make hyperlink
`\style`	specify CSS styles
`\class`	specify CSS class
`\cssId`	specify CSS id
`\unicode`	character from unicode value

Equation Numbers

`\ref`	cite a labeled equation
`\eqref`	cite a labeled equation with parentheses

Additional Packages

You can configure the *textmacros* extension to use additional packages, just as you can specify additional math TeX packages. Normally, these should be pckages designed for text mode, but it is possible to load some of the regular TeX packages as text macros. For example

```
MathJax = {
  loader: {load: ['[tex]/textmacros', '[tex]/bbox']},
  tex: {
    packages: {'[+]': {'textmacros'}},
    textmacros: {
      packages: {'[+]': ['bbox']}
    }
  }
}
```

would make the *bbox* extension available in text mode, so you could use `\bbox` inside `\text{}`, for example. Not all math-mode extensions are approrpriate for textmode, but some can be usefully employed in text mode.

16.7.24 unicode

The *unicode* extension implements a (non-standard) `\unicode{}` macro that allows arbitrary unicode code points to be entered in your mathematics. You can specify the height and depth of the character (the width is determined by the browser), and the default font from which to take the character.

Examples:

```
\unicode{65}                          % the character 'A'
\unicode{x41}                         % the character 'A'
\unicode[.55,0.05]{x22D6}             % less-than with dot, with height .55em and depth␣
↪0.05em
\unicode[.55,0.05][Geramond]{x22D6}   % same taken from Geramond font
\unicode[Garamond]{x22D6}             % same, but with default height, depth of .8em,.
↪2em
```

Once a size and font are provided for a given unicode point, they need not be specified again in subsequent `\unicode{}` calls for that character.

The result of `\unicode{...}` will have TeX class *ORD* (i.e., it will act like a variable). Use `\mathbin{...}`, `\mathrel{...}`, etc., to specify a different class.

Note that a font list can be given in the `\unicode{}` macro. If not is provided, MathJax will use its own fonts, if possible, and then the default font list for unknown characters if not.

Note: In version 2, you could configure the default font to be used for `\unicode` characters if one wasn't given explicitly. This has not been implemented in version 3.

This extension is loaded automatically when the *autoload* extension is used. To load the *unicode* extension explicitly, add `'[tex]/unicode'` to the `load` array of the `loader` block of your MathJax configuration, and add `'unicode'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/unicode']},
  tex: {packages: {'[+]': ['unicode']}}
};
```

Alternatively, use `\require{unicode}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

16.7.25 verb

The *verb* extension defines the `\verb` LaTeX macro that typesets its argument "verbatim" (without further processing) in a monospaced (typewriter) font. The first character after the `\verb` command is used as a delimiter for the argument, which is everything up to the next copy of the delimiter character). E.g.

```
\verb|\sqrt{x}|
```

will typeset `\sqrt{x}` as a literal string.

Note that, due to how MathJax locates math strings within the document, the argument to `\verb` must have balanced braces, so `\verb|{|` is not valid in a web page (use `\mathtt{\{}` instead). If you are passing TeX strings to `MathJax.tex2svg()` or `MathJax.tex2chtml()`, however, braces don't have to be balanced. So

```
const html = MathJax.tex2chtml('\\verb|{|');
```

is valid.

This extension is loaded automatically when the *autoload* extension is used. To load the *verb* extension explicitly (when using `input/tex-base` for example), add `'[tex]/verb'` to the `load` array of the `loader` block of your MathJax configuration, and add `'verb'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/verb']},
  tex: {packages: {'[+]': ['verb']}}
};
```

Alternatively, use `\require{verb}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

These extensions have not yet been ported to version 3:

16.7.26 autobold

The *autobold* extension is no longer available in MathJax version 3.

16.7.27 autoload-all

The *autoload-all* extension has been replaced by the *autoload* extension, which is more easily configurable.

16.7.28 begingroup

The *begingroup* extension has not yet been translated to version 3, so currently it is not available. It should be included in a future release of MathJax.

16.7.29 mediawiki-texvc

The *mediawiki-texvc* extension predefines macros that match the behavior of the MediaWiki Math Extension.

This extension has not yet been translated to version 3, so currently it is not available. It should be included in a future release of MathJax.

See the *A Custom Extension* section for how to create your own TeX extension.

16.8 Supported TeX/LaTeX commands

This is a long list of the TeX macros supported by MathJax. If the macro is defined in an extension, the name of the extension follows the macro name. If the extension is in brackets, the extension will be loaded automatically when the macro or environment is first used.

More complete details about how to use these macros, with examples and explanations, is available at Carol Fisher's TeX Commands Available in MathJax page. (These were written for MathJax v2, but most of the information is still correct for v3.)

In the following tables, the first column lists the macro (or character, or environment), and the second column indicates which package(s) defines the macro. If none is listed, then it is in the base package. If the package name is in bold, then it is preloaded by the components that include the TeX input jax (except for `input/tex-base`, which only includes the base package). If the package name is in italics, then the package is *not* autoloaded by the *autoload* extension.

Note that most macros are not processed inside text-mode material (such as that within `\text{}` and other similar macros). The *textmacros* extension makes additional macros available in text mode, as listed in the documentation for that extention.

16.8.1 Symbols

&	
#	
%	
^	
~	
_	
'	
’	
{	
}	
\ (backslash-space)	
_	
\,	
\;	
\:	
\!	
\{	
\}	
\\	
\&	
\#	
\%	
\>	
|	
\$	

16.8.2 A

\above	
\abovewithdelims	
\abs	*physics*
\absolutevalue	*physics*
\acomm	*physics*
\acos	*physics*
\acosecant	*physics*
\acosine	*physics*
\acot	*physics*
\acotangent	*physics*
\acsc	*physics*
\acute	
\admat	*physics*
\aleph	
\alpha	
\amalg	
\And	
\angle	
\anticommutator	*physics*
\antidiagonalmatrix	*physics*
\approx	
\approxeq	**ams**
\arccos	**base**, *physics*
\arccosecant	*physics*
\arccosine	*physics*
\arccot	*physics*
\arccotangent	*physics*
\arccsc	*physics*
\arcsec	*physics*
\arcsecant	*physics*
\arcsin	**base**, *physics*
\arcsine	*physics*
\arctan	**base**, *physics*
\arctangent	*physics*
\arg	
\array	
\Arrowvert	
\arrowvert	
\asec	*physics*
\asecant	*physics*
\asin	*physics*
\asine	*physics*
\ast	
\asymp	
\atan	*physics*
\atangent	*physics*
\atop	
\atopwithdelims	

16.8.3 B

\backepsilon	**ams**
\backprime	**ams**
\backsim	**ams**
\backsimeq	**ams**
\backslash	
\bar	
\barwedge	**ams**
\Bbb	
\Bbbk	**ams**
\bbFont	
\bbox	bbox
\bcancel	cancel
\because	**ams**
\begin	
\beta	
\beth	**ams**
\between	**ams**
\bf	
\Big	
\big	
\bigcap	
\bigcirc	
\bigcup	
\Bigg	
\bigg	
\Biggl	
\biggl	
\Biggm	
\biggm	
\Biggr	
\biggr	
\Bigl	
\bigl	
\Bigm	
\bigm	
\bigodot	
\bigoplus	
\bigotimes	
\Bigr	
\bigr	
\bigsqcup	
\bigstar	**ams**
\bigtriangledown	
\bigtriangleup	
\biguplus	
\bigvee	
\bigwedge	
\binom	**ams**

Continued on next page

Table 2 – continued from previous page

\blacklozenge	**ams**
\blacksquare	**ams**
\blacktriangle	**ams**
\blacktriangledown	**ams**
\blacktriangleleft	**ams**
\blacktriangleright	**ams**
\bmod	
\bmqty	*physics*
\boldsymbol	boldsymbol
\bot	
\bowtie	
\Box	**ams**
\boxdot	**ams**
\boxed	**ams**
\boxminus	**ams**
\boxplus	**ams**
\boxtimes	**ams**
\Bqty	*physics*
\bqty	*physics*
\Bra	braket
\bra	braket, *physics*
\brace	
\bracevert	
\brack	
\Braket	braket
\braket	braket, *physics*
\breve	
\buildrel	
\bullet	
\Bumpeq	**ams**
\bumpeq	**ams**

16.8.4 C

\cal	
\cancel	cancel
\cancelto	cancel
\Cap	**ams**
\cap	
\cases	
\cdot	
\cdotp	
\cdots	
\ce	mhchem
\centerdot	**ams**
\cfrac	**ams**
\check	
\checkmark	**ams**
\chi	

Continued on next page

Table 3 – continued from previous page

\choose	
\circ	
\circeq	**ams**
\circlearrowleft	**ams**
\circlearrowright	**ams**
\circledast	**ams**
\circledcirc	**ams**
\circleddash	**ams**
\circledR	**ams**
\circledS	**ams**
\class	html
\clubsuit	
\colon	
\color	color, *colorv2*
\colorbox	color
\comm	*physics*
\commutator	*physics*
\complement	**ams**
\cong	
\coprod	
\cos	**base**, *physics*
\cosecant	*physics*
\cosh	**base**, *physics*
\cosine	*physics*
\cot	**base**, *physics*
\cotangent	*physics*
\coth	**base**, *physics*
\cp	*physics*
\cr	
\cross	*physics*
\crossproduct	*physics*
\csc	**base**, *physics*
\csch	*physics*
\cssId	html
\Cup	**ams**
\cup	
\curl	*physics*
\curlyeqprec	**ams**
\curlyeqsucc	**ams**
\curlyvee	**ams**
\curlywedge	**ams**
\curvearrowleft	**ams**
\curvearrowright	**ams**

16.8.5 D

\dagger	
\daleth	**ams**
\dashleftarrow	**ams**

Continued on next page

Table 4 – continued from previous page

\dashrightarrow	**ams**
\dashv	
\dbinom	**ams**
\dd	*physics*
\ddagger	
\ddddot	**ams**
\dddot	**ams**
\ddot	
\ddots	
\DeclareMathOperator	**ams**
\def	**newcommand**
\definecolor	color
\deg	
\Delta	
\delta	
\derivative	*physics*
\det	**base**, *physics*
\determinant	*physics*
\dfrac	**ams**
\diagdown	**ams**
\diagonalmatrix	*physics*
\diagup	**ams**
\Diamond	**ams**
\diamond	
\diamondsuit	
\differential	*physics*
\digamma	**ams**
\dim	
\displaylines	
\displaystyle	
\div	**base**, *physics*
\divergence	*physics*
\divideontimes	**ams**
\dmat	*physics*
\dot	
\Doteq	**ams**
\doteq	
\doteqdot	**ams**
\dotplus	**ams**
\dotproduct	*physics*
\dots	
\dotsb	
\dotsc	
\dotsi	
\dotsm	
\dotso	
\doublebarwedge	**ams**
\doublecap	**ams**
\doublecup	**ams**
\Downarrow	

Continued on next page

Table 4 – continued from previous page

\downarrow	
\downdownarrows	**ams**
\downharpoonleft	**ams**
\downharpoonright	**ams**
\dv	*physics*
\dyad	*physics*

16.8.6 E

\ell	
\emptyset	
\enclose	enclose
\end	
\enspace	
\epsilon	
\eqalign	
\eqalignno	
\eqcirc	**ams**
\eqref	**ams**
\eqsim	**ams**
\eqslantgtr	**ams**
\eqslantless	**ams**
\equiv	
\erf	*physics*
\eta	
\eth	**ams**
\ev	*physics*
\eval	*physics*
\evaluated	*physics*
\exists	
\exp	**base**, *physics*
\expectationvalue	*physics*
\exponential	*physics*
\expval	*physics*

16.8.7 F

\fallingdotseq	**ams**
\fbox	
\fcolorbox	color
\fderivative	*physics*
\fdv	*physics*
\Finv	**ams**
\flat	
\flatfrac	*physics*
\forall	
\frac	**ams**, **base**
\frak	
\frown	
\functionalderivative	*physics*

16.8.8 G

\Game	**ams**
\Gamma	
\gamma	
\gcd	
\ge	
\genfrac	**ams**
\geq	
\geqq	**ams**
\geqslant	**ams**
\gets	
\gg	
\ggg	**ams**
\gggtr	**ams**
\gimel	**ams**
\gnapprox	**ams**
\gneq	**ams**
\gneqq	**ams**
\gnsim	**ams**
\grad	*physics*
\gradient	*physics*
\gradientnabla	*physics*
\grave	
\gt	
\gtrapprox	**ams**
\gtrdot	**ams**
\gtreqless	**ams**
\gtreqqless	**ams**
\gtrless	**ams**
\gtrsim	**ams**
\gvertneqq	**ams**

16.8.9 H

\hat	
\hbar	
\hbox	
\hdashline	
\heartsuit	
\hfil	
\hfill	
\hfilll	
\hline	
\hom	
\hookleftarrow	
\hookrightarrow	
\hphantom	
\href	html
\hskip	
\hslash	**ams**
\hspace	
\Huge	
\huge	
\hypcosecant	*physics*
\hypcosine	*physics*
\hypcotangent	*physics*
\hypsecant	*physics*
\hypsine	*physics*
\hyptangent	*physics*

16.8.10 I

\identitymatrix	*physics*
\idotsint	**ams**
\iff	
\iiiint	**ams**
\iiint	
\iint	
\Im	**base**, *physics*
\imaginary	*physics*
\imat	*physics*
\imath	
\impliedby	**ams**
\implies	**ams**
\in	
\inf	
\infty	
\injlim	**ams**
\innerproduct	*physics*
\int	
\intercal	**ams**
\intop	
\iota	
\it	

16.8.11 J

\jmath	
\Join	**ams**

16.8.12 K

\kappa	
\ker	
\kern	
\Ket	braket
\ket	braket, *physics*
\Ketbra	braket
\ketbra	braket, *physics*

16.8.13 L

\label	
\Lambda	
\lambda	

Continued on next page

Table 5 – continued from previous page

\land	
\langle	
\laplacian	*physics*
\LARGE	
\Large	
\large	
\LaTeX	
\lbrace	
\lbrack	
\lceil	
\ldotp	
\ldots	
\le	
\leadsto	**ams**
\left	
\Leftarrow	
\leftarrow	
\leftarrowtail	**ams**
\leftharpoondown	
\leftharpoonup	
\leftleftarrows	**ams**
\Leftrightarrow	
\leftrightarrow	
\leftrightarrows	**ams**
\leftrightharpoons	**ams**
\leftrightsquigarrow	**ams**
\leftroot	
\leftthreetimes	**ams**
\leq	
\leqalignno	
\leqq	**ams**
\leqslant	**ams**
\lessapprox	**ams**
\lessdot	**ams**
\lesseqgtr	**ams**
\lesseqqgtr	**ams**
\lessgtr	**ams**
\lesssim	**ams**
\let	**newcommand**
\lfloor	
\lg	
\lgroup	
\lhd	**ams**
\lim	
\liminf	
\limits	
\limsup	
\ll	
\llap	
\llcorner	**ams**

Continued on next page

Table 5 – continued from previous page

\Lleftarrow	**ams**
\lll	**ams**
\llless	**ams**
\lmoustache	
\ln	**base**, *physics*
\lnapprox	**ams**
\lneq	**ams**
\lneqq	**ams**
\lnot	
\lnsim	**ams**
\log	**base**, *physics*
\logarithm	*physics*
\Longleftarrow	
\longleftarrow	
\Longleftrightarrow	
\longleftrightarrow	
\longleftrightarrows	*mhchem*
\longLeftrightharpoons	*mhchem*
\longmapsto	
\Longrightarrow	
\longrightarrow	
\longRightleftharpoons	*mhchem*
\longrightleftharpoons	*mhchem*
\looparrowleft	**ams**
\looparrowright	**ams**
\lor	
\lower	
\lozenge	**ams**
\lrcorner	**ams**
\Lsh	**ams**
\lt	
\ltimes	**ams**
\lVert	**ams**
\lvert	**ams**
\lvertneqq	**ams**

16.8.14 M

\maltese	**ams**
\mapsto	
\mathbb	
\mathbf	
\mathbfcal	
\mathbffrak	
\mathbfit	
\mathbfscr	
\mathbfsf	
\mathbfsfit	
\mathbfsfup	

Continued on next page

Table 6 – continued from previous page

\mathbfup	
\mathbin	
\mathcal	
\mathchoice	
\mathclose	
\mathfrak	
\mathinner	
\mathit	
\mathnormal	
\mathop	
\mathopen	
\mathord	
\mathpunct	
\mathrel	
\mathring	**ams**
\mathrm	
\mathscr	
\mathsf	
\mathsfit	
\mathsfup	
\mathstrut	
\mathtip	action
\mathtt	
\matrix	
\matrixdeterminant	*physics*
\matrixel	*physics*
\matrixelement	*physics*
\matrixquantity	*physics*
\max	
\mbox	
\mdet	*physics*
\measuredangle	**ams**
\mel	*physics*
\mho	**ams**
\mid	
\middle	
\min	
\minCDarrowheight	amscd
\minCDarrowwidth	amscd
\mit	
\mkern	
\mmlToken	
\mod	
\models	
\moveleft	
\moveright	
\mp	
\mqty	*physics*
\mskip	
\mspace	

Continued on next page

Table 6 – continued from previous page

\mu	
\multimap	**ams**

16.8.15 N

\nabla	
\natural	
\naturallogarithm	*physics*
\ncong	**ams**
\ne	
\nearrow	
\neg	
\negmedspace	**ams**
\negthickspace	**ams**
\negthinspace	
\neq	
\newcommand	**newcommand**
\newenvironment	**newcommand**
\Newextarrow	extpfeil
\newline	
\nexists	**ams**
\ngeq	**ams**
\ngeqq	**ams**
\ngeqslant	**ams**
\ngtr	**ams**
\ni	
\nLeftarrow	**ams**
\nleftarrow	**ams**
\nLeftrightarrow	**ams**
\nleftrightarrow	**ams**
\nleq	**ams**
\nleqq	**ams**
\nleqslant	**ams**
\nless	**ams**
\nmid	**ams**
\nobreakspace	**ams**
\nolimits	
\nonumber	
\norm	*physics*
\normalsize	
\not	
\notag	**ams**
\notChar	
\notin	
\nparallel	**ams**
\nprec	**ams**
\npreceq	**ams**
\nRightarrow	**ams**
\nrightarrow	**ams**

Continued on next page

Table 7 – continued from previous page

\nshortmid	**ams**
\nshortparallel	**ams**
\nsim	**ams**
\nsubseteq	**ams**
\nsubseteqq	**ams**
\nsucc	**ams**
\nsucceq	**ams**
\nsupseteq	**ams**
\nsupseteqq	**ams**
\ntriangleleft	**ams**
\ntrianglelefteq	**ams**
\ntriangleright	**ams**
\ntrianglerighteq	**ams**
\nu	
\nVDash	**ams**
\nVdash	**ams**
\nvDash	**ams**
\nvdash	**ams**
\nwarrow	

16.8.16 O

\odot	
\oint	
\oldstyle	
\Omega	
\omega	
\omicron	
\ominus	
\op	*physics*
\operatorname	**ams**
\oplus	
\order	*physics*
\oslash	
\otimes	
\outerproduct	*physics*
\over	
\overbrace	
\overleftarrow	
\overleftrightarrow	
\overline	
\overparen	
\overrightarrow	
\overset	
\overwithdelims	
\owns	

16.8.17 P

\parallel	
\partial	
\partialderivative	*physics*
\paulimatrix	*physics*
\pb	*physics*
\pderivative	*physics*
\pdv	*physics*
\perp	
\phantom	
\Phi	
\phi	
\Pi	
\pi	
\pitchfork	**ams**
\pm	
\pmat	*physics*
\pmatrix	
\pmb	
\pmod	
\Pmqty	*physics*
\pmqty	*physics*
\pod	
\poissonbracket	*physics*
\pqty	*physics*
\Pr	**base**, *physics*
\prec	
\precapprox	**ams**
\preccurlyeq	**ams**
\preceq	
\precnapprox	**ams**
\precneqq	**ams**
\precnsim	**ams**
\precsim	**ams**
\prime	
\principalvalue	*physics*
\Probability	*physics*
\prod	
\projlim	**ams**
\propto	
\Psi	
\psi	
\pu	mhchem
\PV	*physics*
\pv	*physics*

16.8.18 Q

\qall	*physics*
\qand	*physics*
\qas	*physics*
\qassume	*physics*
\qc	*physics*
\qcc	*physics*
\qcomma	*physics*
\qelse	*physics*
\qeven	*physics*
\qfor	*physics*
\qgiven	*physics*
\qif	*physics*
\qin	*physics*
\qinteger	*physics*
\qlet	*physics*
\qodd	*physics*
\qor	*physics*
\qotherwise	*physics*
\qq	*physics*
\qqtext	*physics*
\qquad	
\qsince,	*physics*
\qthen	*physics*
\qty	*physics*
\quad	
\quantity	*physics*
\qunless	*physics*
\qusing	*physics*

16.8.19 R

\raise	
\rangle	
\rank	*physics*
\rbrace	
\rbrack	
\rceil	
\Re	**base**, *physics*
\real	*physics*
\ref	
\renewcommand	**newcommand**
\renewenvironment	**newcommand**
\require	**require**
\Res	*physics*
\restriction	**ams**
\rfloor	
\rgroup	

Continued on next page

Table 9 – continued from previous page

\rhd	**ams**
\rho	
\right	
\Rightarrow	
\rightarrow	
\rightarrowtail	**ams**
\rightharpoondown	
\rightharpoonup	
\rightleftarrows	**ams**
\rightleftharpoons	**ams**, **base**
\rightrightarrows	**ams**
\rightsquigarrow	**ams**
\rightthreetimes	**ams**
\risingdotseq	**ams**
\rlap	
\rm	
\rmoustache	
\root	
\Rrightarrow	**ams**
\Rsh	**ams**
\rtimes	**ams**
\Rule	
\rule	
\rVert	**ams**
\rvert	**ams**

16.8.20 S

\S	
\sbmqty	*physics*
\scr	
\scriptscriptstyle	
\scriptsize	
\scriptstyle	
\searrow	
\sec	**base**, *physics*
\secant	*physics*
\sech	*physics*
\Set	braket
\set	braket
\setminus	
\sf	
\sharp	
\shortmid	**ams**
\shortparallel	**ams**
\shoveleft	**ams**
\shoveright	**ams**
\sideset	**ams**
\Sigma	

Continued on next page

Table 10 – continued from previous page

\sigma	
\sim	
\simeq	
\sin	**base**, *physics*
\sine	*physics*
\sinh	**base**, *physics*
\skew	
\SkipLimits	**ams**
\small	
\smallfrown	**ams**
\smallint	
\smallmatrixquantity	*physics*
\smallsetminus	**ams**
\smallsmile	**ams**
\smash	
\smdet	*physics*
\smile	
\smqty	*physics*
\Space	
\space	
\spadesuit	
\sphericalangle	**ams**
\sPmqty	*physics*
\spmqty	*physics*
\sqcap	
\sqcup	
\sqrt	
\sqsubset	**ams**
\sqsubseteq	
\sqsupset	**ams**
\sqsupseteq	
\square	**ams**
\stackrel	
\star	
\strut	
\style	html
\Subset	**ams**
\subset	
\subseteq	
\subseteqq	**ams**
\subsetneq	**ams**
\subsetneqq	**ams**
\substack	**ams**
\succ	
\succapprox	**ams**
\succcurlyeq	**ams**
\succeq	
\succnapprox	**ams**
\succneqq	**ams**
\succnsim	**ams**

Continued on next page

Table 10 – continued from previous page

\succsim	**ams**
\sum	
\sup	
\Supset	**ams**
\supset	
\supseteq	
\supseteqq	**ams**
\supsetneq	**ams**
\supsetneqq	**ams**
\surd	
\svmqty	*physics*
\swarrow	
\symbb	
\symbf	
\symbfcal	
\symbffrak	
\symbfit	
\symbfscr	
\symbfsf	
\symbfsfit	
\symbfsfup	
\symbfup	
\symcal	
\symfrak	
\symit	
\symnormal	
\symrm	
\symscr	
\symsf	
\symsfit	
\symsfup	
\symtt	
\symup	

16.8.21 T

\tag	**ams**
\tan	**base**, *physics*
\tangent	*physics*
\tanh	**base**, *physics*
\tau	
\tbinom	**ams**
\TeX	
\text	
\textbf	
\textcolor	color
\textit	
\textnormal	
\textrm	

Continued on next page

Table 11 – continued from previous page

\textsf	
\textstyle	
\texttip	action
\texttt	
\textup	
\tfrac	**ams**
\therefore	**ams**
\Theta	
\theta	
\thickapprox	**ams**
\thicksim	**ams**
\thinspace	
\tilde	
\times	
\Tiny	
\tiny	
\to	
\toggle	action
\top	
\Tr	*physics*
\tr	*physics*
\Trace	*physics*
\trace	*physics*
\triangle	
\triangledown	**ams**
\triangleleft	
\trianglelefteq	**ams**
\triangleq	**ams**
\triangleright	
\trianglerighteq	**ams**
\tripledash	*mhchem*
\tt	
\twoheadleftarrow	**ams**
\twoheadrightarrow	**ams**

16.8.22 U

\ulcorner	**ams**
\underbrace	
\underleftarrow	
\underleftrightarrow	
\underline	
\underparen	
\underrightarrow	
\underset	
\unicode	unicode
\unlhd	**ams**
\unrhd	**ams**
\Uparrow	
\uparrow	
\Updownarrow	
\updownarrow	
\upharpoonleft	**ams**
\upharpoonright	**ams**
\uplus	
\uproot	
\Upsilon	
\upsilon	
\upuparrows	**ams**
\urcorner	**ams**

16.8.23 V

\va	*physics*
\var	*physics*
\varDelta	**ams**
\varepsilon	
\varGamma	**ams**
\variation	*physics*
\varinjlim	**ams**
\varkappa	**ams**
\varLambda	**ams**
\varliminf	**ams**
\varlimsup	**ams**
\varnothing	**ams**
\varOmega	**ams**
\varPhi	**ams**
\varphi	
\varPi	**ams**
\varpi	
\varprojlim	**ams**
\varpropto	**ams**
\varPsi	**ams**
\varrho	

Continued on next page

Table 12 – continued from previous page

\varSigma	**ams**
\varsigma	
\varsubsetneq	**ams**
\varsubsetneqq	**ams**
\varsupsetneq	**ams**
\varsupsetneqq	**ams**
\varTheta	**ams**
\vartheta	
\vartriangle	**ams**
\vartriangleleft	**ams**
\vartriangleright	**ams**
\varUpsilon	**ams**
\varXi	**ams**
\vb	*physics*
\vcenter	
\Vdash	**ams**
\vDash	**ams**
\vdash	
\vdot	*physics*
\vdots	
\vec	
\vectorarrow	*physics*
\vectorbold	*physics*
\vectorunit	*physics*
\vee	
\veebar	**ams**
\verb	verb
\Vert	
\vert	
\vmqty	*physics*
\vphantom	
\vqty	*physics*
\vu	*physics*
\Vvdash	**ams**

16.8.24 W

\wedge	
\widehat	
\widetilde	
\wp	
\wr	

16.8.25 X

\xcancel	cancel
\Xi	
\xi	
\xleftarrow	**ams**, *mhchem*
\xleftrightarrow	*mhchem*
\xLeftrightharpoons	*mhchem*
\xlongequal	extpfeil
\xmapsto	extpfeil
\xmat	*physics*
\xmatrix	*physics*
\xrightarrow	**ams**, *mhchem*
\xRightleftharpoons	*mhchem*
\xrightleftharpoons	*mhchem*
\xtofrom	extpfeil
\xtwoheadleftarrow	extpfeil
\xtwoheadrightarrow	extpfeil

16.8.26 Y

\yen	**ams**

16.8.27 Z

\zeromatrix	*physics*
\zeta	
\zmat	*physics*

16.8.28 Environments

LaTeX environments of the form \begin{NAME} ... \end{NAME} are provided where NAME is one of the following:

align	**ams**
align*	**ams**
alignat	**ams**
alignat*	**ams**
aligned	**ams**
alignedat	**ams**
array	
Bmatrix	**ams**
bmatrix	**ams**
cases	**ams**
CD	amscd
eqnarray	
eqnarray*	**ams**
equation	
equation*	
gather	**ams**
gather*	**ams**
gathered	**ams**
matrix	**ams**
multline	**ams**
multline*	**ams**
pmatrix	**ams**
smallmatrix	**ams**, *physics*
split	**ams**
subarray	**ams**
Vmatrix	**ams**
vmatrix	**ams**

CHAPTER 17

MathML Support

The support for MathML in MathJax involves two functions: the first looks for `<math>` tags within your document and marks them for later processing by MathJax, and the second converts the MathML to the internal format used by MathJax, where one of MathJax's output processors then displays it in the web page.

In addition, MathJax's internal format is essentially MathML (with a few additions), implemented as javascript objects rather than DOM elements. MathJax's various input processors all convert their original format into this internal MathML format, and its output processors take this MathML and produce the proper output from it. Because the internal format is MathML-based, MathJax provides the ability to convert to and from MathML notation.

Although some browsers have native support for rendering MathML, not all do, and so MathJax makes it possible to view MathML notation in *all* browsers. Even for those that do support MathML, it may be valuable to use MathJax, since that will produce consistent output across all browsers, and MathJax implements features and functionality that is not available in some native MathML implementations.

17.1 MathML in HTML pages

For MathML that is handled via the preprocessor, you should not use named MathML entities, but rather use numeric entities like `&#x221A;` or unicode characters embedded in the page itself. The reason is that entities are replaced by the browser before MathJax runs, and some browsers report errors for unknown entities. For browsers that are not MathML-aware, that will cause errors to be displayed for the MathML entities. While that might not occur in the browser you are using to compose your pages, it can happen with other browsers, so you should avoid the named entities whenever possible. If you must use named entities, you may need to declare them in the *DOCTYPE* declaration by hand.

When you use MathML in an HTML document rather than an XHTML one (MathJax will work with both), you should not use the "self-closing" form for MathML tags with no content, but should use separate open and close tags. That is, use

```
<mspace width="thinmathspace"></mspace>
```

rather than `<mspace width="thinmathspace" />`. This is because HTML does not have self-closing tags, and some browsers will get the nesting of tags wrong if you attempt to use them. For example, with `<mspace`

`width="1em" />`, since there is no closing tag, the rest of the mathematics will become the content of the `<mspace>` tag; but since `<mspace>` should have no content, the rest of the mathematics will not be displayed. This is a common error that should be avoided. Modern browsers that support HTML5 should be able to handle self-closing tags, but older browsers have problems with them, so if you want your mathematics to be visible to the widest audience, do not use the self-closing form in HTML documents.

17.2 Supported MathML tags

MathJax supports the MathML3.0 mathematics tags, with some limitations. The MathML support is still under active development, so some tags are not yet implemented, and some features are not fully developed, but are coming.

The deficiencies include:

- No support for alignment groups in tables.
- Not all attributes are supported for tables. E.g., `columnspan` and `rowspan` are not implemented yet.
- Experimental support for the elementary math tags: `mstack`, `mlongdiv`, `msgroup`, `msrow`, `mscarries`, and `mscarry` (via the `mml3` extension, see below).
- Experimental support for bidirectional mathematics (via the `mml3` extension, see below).

See the results of the MathML3.0 test suite for details.

17.3 Content MathML

The version 2 `content-mathml` extension is not yet available in version 3.

17.4 Experimental mml3 extension

The version 2 `mml3` extension is not yet available in version 3.

17.5 Semantics and Annotations

Some popular annotation formats like TeX, Maple, or Content MathML are often included in the MathML source via the `semantics` element. This is particularly true of MathML that is generated by other software, such as editors or computational tools.

MathJax provides access to these annotations through the `"Show Math As"` menu, via the `Annotations` sub-menu. See the MathML Annotation Framework and the *Contextual Menu Options* documentation for details.

CHAPTER 18

AsciiMath Support

The support for AsciiMath in MathJax involves two functions: the first looks for mathematics within your web page (indicated by delimiters like `` `...` ``) and marks the mathematics for later processing by MathJax, and the second is what converts the AsciiMath notation into MathJax's internal format, where one of MathJax's output processors then displays it in the web page. In MathJax version 2, these were separated into distinct components (the `asciimath2jax` preprocessor and the AsciiMath input jax), but in version 3, the `asciimath2jax` functions have been folded into the AsciiMath input jax.

The AsciiMath input jax actually includes a copy of `ASCIIMathML.js` itself (see the AsciiMath home page for details). This means that the results of MathJax's AsciiMath processing should be the same as using the actual `ASCIIMathML.js` package (at least as far as the MathML that it generates is concerned). Thanks go to David Lippman for writing the initial version of the AsciiMath preprocessor and input jax and for the ongoing improvements from the AsciiMath community.

The AsciiMath input jax handles only the original ASCIIMathML notation (from ASCIIMathML v1.4.7), not the extended LaTeXMathML notation added in version 2.0 of ASCIIMathML, though the AsciiMath input jax does expose the tables that define the symbols that AsciiMath processes, and so it would be possible to extend them to include additional symbols. In general, it is probably better to use MathJax's *TeX input jax* to handle LaTeX notation.

AsciiMath can be configured to look for whatever markers you want to use for your math delimiters. See the *AsciiMath configuration options* section for details on how to customize the action of the AsciiMath input jax.

18.1 Loading the AsciiMath Component

The AsciiMath input jax has not yet been fully ported to version 3. Instead, the AsciiMath component uses the version 2 AsciiMath input jax together with some of the legacy version 2 code patched into the version 3 framework. This is less efficient, and somewhat larger, than a pure version-3 solution would be, and it can complicate the configuration process. A full version-3 port of AsciiMath is planned for a future release.

Because AsciiMath hasn't been fully ported to version 3, none of the combined components include it. So in order to use AsciiMath notation, you will need to configure MathJax to load it yourself by adding `input/asciimath` to the `load` array in the `loader` block of your MathJax configuration. For example,

```
<script>
MathJax = {
  loader: {load: ['input/asciimath', 'output/chtml', 'ui/menu']},
};
</script>
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/startup.js">
</script>
```

would load the AsciiMath input jax, the CommonHTML output jax, and the contextual menu component.

18.2 AsciiMath delimiters

By default, the AsciiMath processor defines the back-tick (`) as the delimiters for mathematics in AsciiMath format. It does **not** define `$...$` as math delimiters. That is because dollar signs appear too often in non-mathematical settings, which could cause some text to be treated as mathematics unexpectedly. For example, with single-dollar delimiters, "... the cost is $2.50 for the first one, and $2.00 for each additional one ..." would cause the phrase "2.50 for the first one, and" to be treated as mathematics since it falls between dollar signs. For this reason, if you want to use single-dollars for AsciiMath notation, you must enable that explicitly in your configuration:

```
window.MathJax = {
  loader: {
    load: ['input/asciimath']
  },
  asciimath: {
    delimiters: [['$','$'], ['`','`']]
  }
});
```

Note that the dollar signs are frequently used as a delimiter for mathematics in the TeX format, and you can not enable the dollar-sign delimiter for both. It is probably best to leave dollar signs for TeX notation.

See the *AsciiMath Input Processor Options* page, for additional configuration parameters that you can specify for the AsciiMath input processor.

18.3 AsciiMath in HTML documents

The AsciiMath syntax is described on the official AsciiMath homepage.

Keep in mind that your mathematics is part of an HTML document, so you need to be aware of the special characters used by HTML as part of its markup. There cannot be HTML tags within the math delimiters (other than `<br>`, `<wbr>`, and HTML comments) as AsciiMath-formatted math does not include HTML tags. Also, since the mathematics is initially given as text in the page, you need to be careful that your mathematics doesn't look like HTML tags to the browser, which parses the page before MathJax gets to see it. In particular, that means that you have to be careful about things like less-than and greater-than signs (< and >), and ampersands (`&`), which have special meaning to web browsers. For example,

```
... when `x<y` we have ...
```

will cause a problem, because the browser will think `<y` is the beginning of a tag named `y` (even though there is no such tag in HTML). When this happens, the browser will think the tag continues up to the next > in the document (typically the end of the next actual tag in the HTML file), and you may notice that you are missing part of the text of the document. In the example above, the "`<y`" and "`we have ...`" will not be displayed because the browser

thinks it is part of the tag starting at `<y`. This is one indication you can use to spot this problem; it is a common error and should be avoided.

Usually, it is sufficient simply to put spaces around these symbols to cause the browser to avoid them, so

```
... when `x < y` we have ...
```

should work. Alternatively, you can use the HTML entities `&lt;`, `&gt;` and `&amp;` to encode these characters so that the browser will not interpret them, but MathJax will. E.g.,

```
... when `x &lt; y` we have ...
```

Keep in mind that the browser interprets your text before MathJax does.

CHAPTER 19

MathJax Output Formats

Currently, MathJax can render math in three ways:

- Using HTML and CSS to lay out the mathematics,
- Using Scalable Vector Graphics (SVG) to lay out the mathematics, or
- As a serialized MathML string.

The first two are implemented by the `CommonHTML` and `SVG` output processors. The third is a consequence of the fact that MathJax uses MathML as its internal format. While MathJax version 2 included a `NativeMML` output processor that produced MathML notation for those browsers that support it, this has been dropped from version 3. See the *MathML Support* section for more information on how to get MathML output.

If you are using one of the combined component files, then this will select one of these output processors for you. If the component file ends in `-chtml`, then it is the CommonHTML output processor, while if it ends in `-svg` then the SVG output processor will be used.

If you are performing your own in-line or file-based configuration, you select which one you want to use by including either `'output/chtml'` or `'output/svg'` in the `load` array of the `loader` section of your MathJax configuration. For example

```
window.MathJax = {
  loader: {load: ["input/tex", "output/chtml"]}
};
```

would specify TeX input and CommonHTML output for the mathematics in your document.

Warning: The `PreviewHTML`, `PlainSource`, and `NativeMML` output formats from version 2 are not available in version 3. These may be available in future releases if there is demand for them.

19.1 HTML Support

The *CommonHTML* output processor renders your mathematics using HTML with CSS styling. It produces high-quality output in all modern browsers, with results that are consistent across browsers and operating systems. This is MathJax's primary output mode since MathJax version 2.6. Its major advantage is its quality, consistency, and the fact that its output is independent of the browser, operating system, and user environment. This means you can pre-process mathematics on a server, without needing to know the browser, what fonts are available, and so on. (In version 2, both the `HTML-CSS` and `NativeMML` processors produced different output for different browsers and user environments.)

The CommonHTML output uses web-based fonts so that users don't have to have math fonts installed on their computers, but will use locally installed ones if they are available. It currently only supports MathJax's default TeX fonts (see the *MathJax Font Support* section for more information).

See *CommonHTML Output Processor Options* for information about the options that control the CommonHTML output.

19.2 SVG Support

The `SVG` output processor uses *Scalable Vector Graphics* to render the mathematics on the page. SVG is supported in all the major browsers and most mobile devices; note, however, that Internet Explorer prior to IE9 does not support SVG (MathJax version 3 doesn't support these in any case), and IE9 only does in "IE9 standards mode", not its emulation modes for earlier versions. The SVG output mode is high quality, and displays and prints well in all browsers. Since it uses SVG data instead of font files, it is not affected by user-based web-font blocking, or other character placement issues that sometimes occur with the HTML-based output.

One advantage to the SVG output is that it is relatively self-contained (it does not rely heavily on CSS, though it does use some in certain circumstances), so it can be saved and used as an independent image. One disadvantage of this mode is that its variable-width tables become fixed size once they are typeset, and don't rescale if the window size changes (for example).

In version 2, equation tags and numbers where produced using a fixed width as well, so the equation number would not change with changes in window size. In version 3, however, equation numbers now are based on the container size, and move with changes in its size, just as they do with CommonHTML output.

Finally, because mathematical characters in SVG output are produced by SVG paths, not characters in a font, they can't be copy and pasted, as the output of the CommonHTML processor can.

See *SVG Output Processor Options* for information about the options that control the SVG output.

19.3 MathML Support

MathJax uses MathML as the basis for its internal format for mathematical expressions, so MathML support is built into MathJax at a fundamental level. There is a *MathML input jax* for converting from MathML elements into the internal format (javascript objects representing the MathML elements), and there is a mechanism that can convert the internal format into a serialized MathML string provided by `MathJax.startup.toMML()` (if you are using MathJax components).

While MathJax version 2 included a *NativeMML* output jax for producing MathML output in the web page, because MathML is not available in the Chrome, Edge, and IE browsers, because the MathML support in Safari and Firefox don't include all the features needed by MathJax (e.g., the *<mlabeledtr>* element needed for labeled equations), and because the quality of the results in Safari and Firefox are not always comparable to the output from MathJax, the *NativeMML* output jax is no longer provided in MathJax version 3.

You can, however, use MathJax's MathML serialization features to implement your own native MathML output if you wish. Here is one example that does so for TeX input to MathML output.

```
<style>
mjx-container[display="block"] {
  display: block;
  margin: 1em 0;
}
</style>
<script>
MathJax = {
  //
  //  Load only TeX input and the contextual menu
  //
  loader: {load: ['input/tex', 'ui/menu']},
  //
  //  When page is ready, render the math in the document
  //
  //
  //  When page is ready:
  //    disable the assistive-mathml menu item
  //    render the document, handling require and autoload calls
  //
  startup: {
    pageReady() {
      MathJax.startup.document.menu.menu.findID('Accessibility', 'AssistiveMml').
↪disable();
      MathJax._.mathjax.mathjax.handleRetriesFor(() => MathJax.startup.document.
↪render());
    }
  },
  //
  //  Override the usual typeset render action with one that generates MathML output
  //
  options: {
    renderActions: {
      assistiveMml: [],  // disable assistive mathml
      typeset: [150,
        (doc) => {for (math of doc.math) {MathJax.config.renderMathML(math, doc)}},
        (math, doc) => MathJax.config.renderMathML(math, doc)
      ]
    },
    menuOptions: {
      settings: {
        assistiveMml: false
      }
    }
  },
  //
  // The action to use for rendering MathML
  //
  renderMathML(math, doc) {
    math.typesetRoot = document.createElement('mjx-container');
    math.typesetRoot.innerHTML = MathJax.startup.toMML(math.root);
    math.display && math.typesetRoot.setAttribute('display', 'block');
  }
};
</script>
```

(continues on next page)

(continued from previous page)

```
<script type="text/javascript" id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/startup.js">
</script>
```

This example uses the *startup* component to load just the *input/tex* and *contextual menu* components, and defines a new render action that replaces the standard `typeset` action with one that creates a MathJax container element and stores it in `math.typesetRoot`, then converts the internal format to a MathML string (via `MathJax.startup.toMML()`) and has the browser parse that into DOM element (via `innerHTML`). A later render action will move the container and its MathML contents into the DOM at the proper location. For math that is in display style, the container is marked with an attribute so that CSS can be used to make the container be a block-level element with some top and bottom margin.

The example also takes several steps to disable the Assistive MathML extension that inserts hidden MathML for the usual output renders. This is unneeded since we are generating MathML ourselves as the primary output. Setting the `menuOptions.settings.assistiveMml` option to `false` turns off the assistive MathML in the contextual menu. The `pageReady()` function also includes a line that disables the assistive-MathML item in the menu, so user's can't accidentaly turn it on again. Finally, the *assistiveMml* render action is disabled, since it will never be activated (overkill perhaps, but no need to run the usual code for nothing).

Note: MathJax's version 2 NativeMML output processor worked around various limitations of Firefox/Gecko and Safari/WebKit (e.g., to provide support for equation labels), but this approach does not, as it just uses the generic MathML.

CHAPTER 20

Automatic Line Breaking

Automatic line breaking has not yet been implemented in MathJax version 3, but is high on our list for inclusion in a future release.

CHAPTER 21

MathJax Font Support

MathJax version 3 currently supports only one font, the MathJax TeX font. Version 2 provides the following fonts:

- MathJax TeX (default)
- STIX General
- Asana Math
- Neo Euler
- Gyre Pagella
- Gyre Termes
- Latin Modern

MathJax contains customized webfont versions of these fonts. In particular, these customized versions are split over several files to minimize the page load.

MathJax 3 will support these fonts in a future version.

21.1 Use of Other Fonts

In version 2 of MathJax, it was difficult to adjust the fonts in use (once loaded), or to replace individual or collections of characters being used. For example, switching the variables and function names to use a sans-serif font rather than the standard serifed font is quite difficult in version 2. The structure of the font data in version 3 has been completely redesigned to help make such changes easier to make.

Since browsers do not provide APIs to access font metrics, MathJax has to ship with the necessary font data; this font data is generated during development and cannot be determined easily on the fly. The tools for creating the data needed by MathJax have not yet been created for version 3 (the data for the MathJax TeX font was converted from the version 2 format by hand). These tools are high on the list for inclusion in the next version of MathJax, which should provide the additional fonts missing from the initial release of version 3. At that point, the details of how to mix-and-match font characters, and how to create the data files for your own fonts for use in MathJax, will be provided.

21.2 Character fallbacks

No font contains a suitable glyph for every character specified in the Unicode standard. When MathJax encounters a character that isn't in the font that it is using, it will fall back to other fonts in a variety of ways.

First, MathJax enhances Unicode coverage of its default TeX fonts, e.g., combining two double integrals `U+222C` when a quadruple integral `U+2A0C` is used. However, this cannot create every character specified in Unicode. Next, MathJax will run through a fallback chain within the configured fonts (e.g., upright Greek will be substituted with italic Greek).

Finally, when all else fails, MathJax will ask the browser to provide the glyph from a system font. Since in that final case, MathJax will not have the necessary data on the glyph's bounding box, MathJax will guess these metrics. When run in a browser, MathJax will be able to determine the character's width, but not its height and depth, so it will use default values these metrics. Measuring the width can negatively affect the rendering speed, and guessing the height and depth can reduce the quality of the resulting output. When used on a server or in a command-line application, MathJax won't even be able to determine the width, and that has an even more serous consequences for the layout, in general. Thus it is best to use only the characters that are in the MathJax fonts when using server-side rendering.

CHAPTER 22

Browser Compatibility

Extensive browser support is an important goal for MathJax; at the same time, MathJax does require a certain minimum level of browser functionality. While MathJax version 2 went to great lengths to remain compatible with early versions of most browsers (even back to IE6), MathJax version 3 relies on more modern browser features, and so older browsers are no longer supported.

The CommonHTML and SVG output supports all modern browsers (Chrome, Safari, Firefox, Edge), and most mobile browsers. Include the polyfill library in order to support earlier browser versions (see their browser support page for details). In particular, to allow MathJax version 3 to work with IE11, include the line

```
<script src="https://polyfill.io/v3/polyfill.min.js?features=es6"></script>
```

before the script that loads MathJax.

Please file issues on GitHub if you notice inaccuracies or problems. It may help to add a screenshot; we suggest services such as browsershots.org, saucelabs.com, or browserstack.com for obtaining them.

22.1 Viewport meta tag

The viewport meta tag provides the browser with instructions regarding viewports and zooming. This way, web developers can control how a webpage is displayed on a mobile device.

Incorrect or missing viewport information can confuse MathJax's layout process, leading to very small font sizes. We recommend that you use standard values such as the following:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

22.2 Internet Explorer Emulation modes

Internet Explorer provides so-called emulation modes for backward compatibility to its legacy versions. These emulation modes have been deprecated since Internet Explorer 11, cf. Microsoft documentation.

MathJax is fastest when in the standards mode of each IE version, so it is best to force the highest mode possible. That can be accomplished by adding

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

at the top of the <head> section of your HTML documents.

Note: This line must come at the beginning of the <head>, before any stylesheets, scripts, or other content are loaded.

Note that versions of IE prior to 11 are no longer supported in MathJax version 3.

CHAPTER 23

Configuring MathJax

The various components of MathJax, including its input and output processors, its extensions, and the MathJax core, all can be configured though a `MathJax` global object that specifies the configuration you want to use. The `MathJax` object consists of sub-objects that configure the individual components of MathJax. For example, the *input/tex* component is configured through a `tex` block within the `MathJax` object, while the *startup* component is configured through the `startup` block.

These blocks are javascript objects that includes `name: value` pairs giving the names of parameters and their values, with pairs separated by commas. Be careful not to include a comma after the last value, however, as some browsers will fail to process the configuration if you do.

Some blocks may contain further sub-blocks. For example, the `tex` block can have a `macros` sub-block that predefines macros, and a `tagformat` block (when the *tagformat* component is used) to define how equation tags are displayed and handled.

For example,

```
window.MathJax = {
  loader: {
    load: ['[tex]/tagformat']
  },
  startup: {
    pageReady: () => {
      alert('Running MathJax');
      return MathJax.startup.defaultPageReady();
    }
  },
  tex: {
    packages: {'[+]': ['tagformat']},
    tagSide: 'left',
    macros: {
      RR: '{\\bf R}',
      bold: ['{\\bf #1}',1]
    },
    tagformat: {
       tag: (n) => '[' + n + ']'
```

(continues on next page)

(continued from previous page)

```
    }
  }
};
```

is a configuration that asks for the *tagformat* extension to be loaded, sets up the *startup* component to have a function that it runs when the page (and MathJax) are ready (the function issues an alert and then does the usual `pageReady()` function, which typesets the page), configures the *TeX input* component to use the *tagformat* extension, asks for displayed equations to be typeset to the left (rather than centered), defines two macros, and finally set the tagging so that it uses square brackets rather than parentheses for equation numbers and tags.

Note the special notation used with the `packages` option above. The `packages` property is an array of extension names, but the configuration uses a special object to add to that array rather than replace it. If the option you are setting is an array, and you provide an object that has a single properly whose name is `'[+]'` and whose value is an array, then that array will be appended to the default value for the option you are setting. So in the example above, the `'tagformat'` string is added to the default `packages` array (without your needing to know what that default value is).

Similarly, if you use an object with a single property whose name is `'[-]'` and whose value is an array, the elements in that array are *removed* from the default value of the option you are setting. For example,

```
packages: {'[-]': ['autoload', 'require']}
```

would **remove** the *autoload* and *require* packages from the default `packages` array.

Finally, you can combine `'[+]'` and `'[-]'` in one object to do both actions. E.g.,

```
packages: {'[+]': ['enclose'], '[-]': ['autoload', 'require']}
```

would remove the *autoload* and *require* packages from the default `packages` array, and add *enclose* to the result.

In the links below, the various options are first listed with their default values as a complete configuration block, and then each option is explained further below that.

23.1 Input Processor Options

23.1.1 TeX Input Processor Options

The options below control the operation of the *TeX input processor* that is run when you include `'input/tex'`, `'input/tex-full'`, or `'input/tex-base'` in the `load` array of the `loader` block of your MathJax configuration, or if you load a combined component that includes the TeX input jax. They are listed with their default values. To set any of these options, include a `tex` section in your `MathJax` global object.

The Configuration Block

```
MathJax = {
  tex: {
    packages: ['base'],         // extensions to use
    inlineMath: [               // start/end delimiter pairs for in-line math
      ['\\(', '\\)']
```

(continues on next page)

(continued from previous page)

```
    ],
    displayMath: [              // start/end delimiter pairs for display math
      ['$$', '$$'],
      ['\\[', '\\]']
    ],
    processEscapes: true,       // use \$ to produce a literal dollar sign
    processEnvironments: true, // process \begin{xxx}...\end{xxx} outside math mode
    processRefs: true,          // process \ref{...} outside of math mode
    digits: /^(?:[0-9]+(?:\{,\}[0-9]{3})*(?:\.[0-9]*)?|\.[0-9]+)/,
                                // pattern for recognizing numbers
    tags: 'none',               // or 'ams' or 'all'
    tagSide: 'right',           // side for \tag macros
    tagIndent: '0.8em',         // amount to indent tags
    useLabelIds: true,          // use label name rather than tag for ids
    multlineWidth: '85%',       // width of multline environment
    maxMacros: 1000,            // maximum number of macro substitutions per expression
    maxBuffer: 5 * 1024,        // maximum size for the internal TeX string (5K)
    baseURL:                    // URL for use with links to tags (when there is a
↪<base> tag in effect)
       (document.getElementsByTagName('base').length === 0) ?
        '' : String(document.location).replace(/#.*$/, '')),
    formatError:                // function called when TeX syntax errors occur
        (jax, err) => jax.formatError(err)
  }
};
```

Note that some extensions make additional options available. See the *TeX Extension Options* section below for details.

Note: The default for `processEscapes` has changed from `false` in version 2 to `true` in version 3.

Option Descriptions

`packages: ['base']`
: This array lists the names of the packages that should be initialized by the TeX input processor. The *input/tex* and *input/tex-full* components automatically add to this list the packages that they load. If you explicitly load addition tex extensions, you should add them to this list. For example:

```
MathJax = {
  loader: {load: ['[tex]/enclose']},
  tex: {
    packages: {'[+]': ['enclose']}
  }
};
```

This loads the *enclose* extension and acticates it by including it in the package list.

You can remove packages from the default list using `'[-]'` rather than `[+]`, as in the followiong example:

```
MathJax = {
  tex: {
    packages: {'[-]': ['noundefined']}
  }
};
```

This would disable the *noundefined* extension, so that unknown macro names would cause error messages rather than be displayed in red.

If you need to both remove some default packages and add new ones, you can do so by including both within the braces:

```
MathJax = {
  loader: {load: ['[tex]/enclose']},
  tex: {
    packages: {'[-]': ['noundefined', 'autoload'], '[+]': ['enclose']}
  }
};
```

This disables the *noundefined* and *autoload* extensions, and adds in the *enclose* extension.

inlineMath: [['\(','\)']]
This is an array of pairs of strings that are to be used as in-line math delimiters. The first in each pair is the initial delimiter and the second is the terminal delimiter. You can have as many pairs as you want. For example,

```
inlineMath: [ ['$','$'], ['\\(','\\)'] ]
```

would cause MathJax to look for `$...$` and `\(...\)` as delimiters for in-line mathematics. (Note that the single dollar signs are not enabled by default because they are used too frequently in normal text, so if you want to use them for math delimiters, you must specify them explicitly.)

Note that the delimiters can't look like HTML tags (i.e., can't include the less-than sign), as these would be turned into tags by the browser before MathJax has the chance to run. You can only include text, not tags, as your math delimiters.

displayMath: [['$$','$$'], ['\[','\]']]
This is an array of pairs of strings that are to be used as delimiters for displayed equations. The first in each pair is the initial delimiter and the second is the terminal delimiter. You can have as many pairs as you want.

Note that the delimiters can't look like HTML tags (i.e., can't include the less-than sign), as these would be turned into tags by the browser before MathJax has the chance to run. You can only include text, not tags, as your math delimiters.

processEscapes: false
When set to `true`, you may use `\$` to represent a literal dollar sign, rather than using it as a math delimiter, and `\\` to represent a literal backslash (so that you can use `\\\$` to get a literal `\$` or `\\$...$` to get a backslash jsut before in-line math). When `false`, `\$` will not be altered, and its dollar sign may be considered part of a math delimiter. Typically this is set to `true` if you enable the `$ ... $` in-line delimiters, so you can type `\$` and MathJax will convert it to a regular dollar sign in the rendered document.

processRefs: true
When set to `true`, MathJax will process `\ref{...}` outside of math mode.

processEnvironments: true
When `true`, *tex2jax* looks not only for the in-line and display math delimiters, but also for LaTeX environments (`\begin{something}...\end{something}`) and marks them for processing by MathJax. When `false`, LaTeX environments will not be processed outside of math mode.

digits: /^(?:[0-9]+(?:{,}[0-9]{3})*(?:.[0-9]*)?|.[0-9]+)/
This gives a regular expression that is used to identify numbers during the parsing of your TeX expressions. By default, the decimal point is `.` and you can use `{,}` between every three digits before that. If you want to use `{,}` as the decimal indicator, use

```
MathJax = {
  tex: {
```

(continues on next page)

(continued from previous page)

```
    digits: /^(?:[0-9]+(?:\{,\}[0-9]*)?|\{,\}[0-9]+)/
  }
};
```

tags: 'none'
: This controls whether equations are numbered and how. By default it is set to `'none'` to be compatible with earlier versions of MathJax where auto-numbering was not performed (so pages will not change their appearance). You can change this to `'ams'` for equations numbered as the *AMSmath* package would do, or `'all'` to get an equation number for every displayed equation.

tagSide: 'right'
: This specifies the side on which `\tag{}` macros will place the tags, and on which automatic equation numbers will appear. Set it to `'left'` to place the tags on the left-hand side.

tagIndent: "0.8em"
: This is the amount of indentation (from the right or left) for the tags produced by the `\tag{}` macro or by automatic equation numbers.

useLabelIds: true
: This controls whether element IDs for tags use the `\label` name or the equation number. When `true`, use the label, when `false`, use the equation number.

multlineWidth: "85%"
: The width to use for the *multline* environment that is part of the *ams* extension. This width gives room for tags at either side of the equation, but if you are displaying mathematics in a small area or a thin column of text, you might need to change the value to leave sufficient margin for tags.

maxMacros: 10000
: Because a definition of the form `\def\x{\x} \x` would cause MathJax to loop infinitely, the `maxMacros` constant will limit the number of macro substitutions allowed in any expression processed by MathJax.

maxBuffer: 5 * 1024
: Because a definition of the form `\def\x{\x aaa} \x` would loop infinitely, and at the same time stack up lots of a's in MathJax's equation buffer, the `maxBuffer` constant is used to limit the size of the string being processed by MathJax. It is set to 5KB, which should be sufficient for any reasonable equation.

baseURL: (document.getElementsByTagName('base').length === 0) ? '' : String(document.location).replace(/#.*$/, '')
: This is the base URL to use when creating links to tagged equations (via `\ref{}` or `\eqref{}`) when there is a `<base>` element in the document that would affect those links. You can set this value by hand if MathJax doesn't produce the correct link.

formatError: (jax, err) => jax.formatError(err)
: This is a function that is called when the TeX input jax reports a syntax or other error in the TeX that it is processing. The default is to generate an `<merror>` MathML element with the message indicating the error that occurred. You can override the function to perform other tasks, like recording the message, replacing the message with an alternative message, or throwing the error so that MathJax will stop at that point (you can catch the error using promises or a `try/carch` block).

The remaining options are described in the *Options Common to All Input Processors* section.

Developer Options

In addition to the options listed above, low-level options intended for developers include the following:

FindTeX: null
: The FindTeX object instance that will override the default one. This allows you to create a subclass of FindTeX and pass that to the TeX input jax. A null value means use the default FindTeX class and make a new instance of that.

TeX Extension Options

Several of the TeX extensions make additional options available in the tex block of your MathJax configuration. These are described below. Note that the *input/tex* component, and the combined components that load the TeX input jax, include a number of these extensions automatically, so some these options will be available by default.

For example, the *configmacros* package adds a macros block to the tex configuration block that allows you to pre-define macros for use in TeX espressions:

```
MathJax = {
  tex: {
    macros: {
      R: '\\mathbf{R}'
    }
  }
}
```

The options for the various TeX packages (that have options) are described in the links below:

- *amscd Options*
- *autoload Options*
- *color Options*
- *configmacros Options*
- *noundefined Options*
- *require Options*
- *tagformat Options*

23.1.2 MathML Input Processor Options

The options below control the operation of the *MathML input processor* that is run when you include 'input/mml' in the load array of the loader block of your MathJax configuration, or if you load a combined component that includes the MathML input jax. They are listed with their default values. To set any of these options, include an mml section in your MathJax global object.

The Configuration Block

```
MathJax = {
  mml: {
    parseAs: 'html',                    // or 'xml'
    forceReparse: false,                // true to serialize and re-parse all MathML
    parseError: function (node) {       // function to process parsing errors
```

(continues on next page)

(continued from previous page)

```
      this.error(this.adaptor.textContent(node).replace(/\n.*/g, ''));
    },
    verify: {                           // parameters controling verification of
↪MathML
      checkArity: true,                 //   check if number of children is correct
      checkAttributes: false,           //   check if attribute names are valid
      fullErrors: false,                //   display full error messages or just
↪error node
      fixMmultiscripts: true,           //   fix unbalanced mmultiscripts
      fixMtables: true                  //   fix incorrect nesting in mtables
    }
  }
};
```

Option Descriptions

parseAs: 'html'
: Specifies how MathML strings should be parsed: as XML or as HTML. When set to `'xml'`, the browser's XML parser is used, which is more strict about format (e.g., matching end tags) than the HTML parser, which is the default. In node application (where the `liteDOM` is used), these both use the same parser, which is not very strict.

forceReparse: false
: Specifies whether MathJax will serialize and re-parse MathML found in the document. This can be useful if you want to do XML parsing of the MathML from an HTML document.

parseError: (node) => {...}
: Specifies a function to be called when there is a parsing error in the MathML (usually only happens with XML parsing). The `node` is a DOM node containing the error text. Your function can process that in any way it sees fit. The default is to call the MathML input processor's error function with the text of the error (which will create an `merror` node with the error message). Note that this function runs with `this` being the MathML input processor object.

verify: {...}
: This object controls what verification/modifications are to be performed on the MathML that is being processed by MathJax. The values that can be included in the `verify` object are the following:

  checkArity: true
  : This specifies whether the number of children is verified or not. The default is to check for the correct number of children. If the number is wrong, the node is replaced by an <merror> node containing either a message indicating the wrong number of children, or the name of the node itself, depending on the setting of `fullErrors` below.

  checkAttributes: false
  : This specifies whether the names of all attributes are checked to see if they are valid on the given node (i.e., they have a default value, or are one of the standard attributes such as `style`, `class`, `id`, `href`, or a `data-` attribute. If an attribute is in error, the node is either placed inside an <merror> node (so that it is marked in the output as containing an error), or is replaced by an <merror> containing a full message indicating the bad attribute, depending on the setting of `fullErrors` below.

    Currently only names are checked, not values. Value verification may be added in a future release.

  fullErrors: false
  : This specifies whether a full error message is displayed when a node produces an error, or whether just the node name is displayed (or the node itself in the case of attribute errors).

fixMmultiscripts: true
: This specifies whether extra <none/> entries are added to <mmultiscripts> elements to balance the super- ans subscripts, as required by the specification, or whether to generate an error instead.

fixMtables: true
: This specifies whether missing <mtable>, <mtr> and <mtd> elements are placed around cells or not. When true, MathJax will attempt to correct the table structure if these elements are missing from the tree. For example, an <mtr> element that is not within an <mtable> will have an <mtable> placed around it automatically, and an <mtable> containing an <mi> as a direct child node will have an <mtr> and <mtd> inserted around the <mi>.

Developer Options

In addition to the options listed above, low-level options intended for developers include the following:

FindMathML: null
: The FindMathML object instance that will override the default one. This allows you to create a subclass of FindMathML and pass that to the MathML input jax. A null value means use the default FindMathML class and make a new instance of that.

MathMLCompile: null
: The MathMLCompile object instance that will override the default one. This allows you to create a subclass of MathMLCompile and pass that to the MathML input jax. A null value means use the default MathMLCompile class and make a new instance of that.

23.1.3 AsciiMath Input Processor Options

The options below control the operation of the *AsciiMath input processor* that is run when you include 'input/asciimath' in the in the load array of the loader block of your MathJax configuration, or if you load a combined component that includes the AsciiMath input jax (none currently do, since the AsciiMath input has not been fully ported to version 3). They are listed with their default values. To set any of these options, include an asciimath section in your MathJax global object.

The Configuration Block

```
MathJax = {
  asciimath: {
    fixphi: true,               // true for TeX mapping, false for unicode mapping
    displaystyle: true,         // true for displaystyle typesetting, false for in-line
    decimalsign: '.'            // character to use for decimal separator
  }
};
```

Option Descriptions

`fixphi: true`
: Determines whether MathJax will switch the Unicode values for `phi` and `varphi`. If set to `true` MathJax will use the TeX mapping, otherwise the Unicode mapping.

`displaystyle: true`
: Determines whether operators like summation symbols will have their limits above and below the operators (true) or to their right (false). The former is how they would appear in displayed equations that are shown on their own lines, while the latter is better suited to in-line equations so that they don't interfere with the line spacing so much.

`decimalsign: "."`
: This is the character to be used for decimal points in numbers. If you change this to `','`, then you need to be careful about entering points or intervals. E.g., use `(1, 2)` rather than `(1,2)` in that case.

The remaining options are described in the *Options Common to All Input Processors* section.

Developer Options

In addition to the options listed above, low-level options intended for developers include the following:

`FindAsciiMath: null`
: The `FindAsciiMath` object instance that will override the default one. This allows you to create a subclass of `FindAsciiMath` and pass that to the AsciiMath input jax. A `null` value means use the default `FindAsciiMath` class and make a new instance of that.

23.1.4 Options Common to All Input Processors

There are no options that are common to all input jax, but a number of the *Document Options* affect what portions of the document will be processed by the input jax that scan the page for delimiters (i.e., TeX and AsciiMath). In particular, the options that correspond to the version-2 options `skipTags`, `includeTags`, and similar options for the various v2 pre-processors are now document-level options.

23.2 Output Processor Options

There are a number of configuration options that are common to all the output processors. These are described following the links below, which give the options that are specific to the particular output jax.

23.2.1 CommonHTML Output Processor Options

The options below control the operation of the *CommonHTML output processor* that is run when you include `'output/chtml'` in the `load` array of the `loader` block of your MathJax configuration, or if you load a combined component that includes the CommonHTML output jax. They are listed with their default values. To set any of these options, include a `chtml` section in your `MathJax` global object.

The Configuration Block

```
MathJax = {
  chtml: {
    scale: 1,                     // global scaling factor for all expressions
    minScale: .5,                 // smallest scaling factor to use
    matchFontHeight: true,        // true to match ex-height of surrounding font
    mtextInheritFont: false,      // true to make mtext elements use surrounding font
    merrorInheritFont: true,      // true to make merror text use surrounding font
    mathmlSpacing: false,         // true for MathML spacing rules, false for TeX␣
↪rules
    skipAttributes: {},           // RFDa and other attributes NOT to copy to the␣
↪output
    exFactor: .5,                 // default size of ex in em units
    displayAlign: 'center',       // default for indentalign when set to 'auto'
    displayIndent: '0',           // default for indentshift when set to 'auto'
    fontURL: '[mathjax]/components/output/chtml/fonts/woff-v2',   // The URL where␣
↪the fonts are found
    adaptiveCSS: true             // true means only produce CSS that is used in the␣
↪processed equations
  }
};
```

Option Descriptions

`fontURL: '[mathjax]/components/output/chtml/fonts/woff-v2'`
: This is the URL to the location where the MathJax fonts are stored. In the default, `[mathjax]` is replaced by the location from which you have loaded MathJax. You should include a complete URL to the location of the fonts you want to use.

`adaptiveCSS: true`
: This setting controls how the CommonHTML output jax handles the CSS styles that it generates. When true, this means that only the CSS needed for the math that has been processed on the page so far is generated. When false, the CSS needed for all elements and all characters in the MathJax font are generated. This is an extremely large amount of CSS, and that can have an effect on the performance of your page, so it is best to leave this as `true`. You can reset the information about what CSS is needed by using the command

  ```
  MathJax.startup.document.output.clearCache();
  ```

  to clear the font cache.

The remaining options are described in the *Options Common to All Output Processors* section.

23.2.2 SVG Output Processor Options

The options below control the operation of the *SVG output processor* that is run when you include `'output/svg'` in the `load` array of the `loader` block of your MathJax configuration, or if you load a combined component that includes the CommonHTML output jax. They are listed with their default values. To set any of these options, include an `svg` section in your `MathJax` global object.

The Configuration Block

```
MathJax = {
  svg: {
    scale: 1,                      // global scaling factor for all expressions
    minScale: .5,                  // smallest scaling factor to use
    mtextInheritFont: false,       // true to make mtext elements use surrounding font
    merrorInheritFont: true,       // true to make merror text use surrounding font
    mathmlSpacing: false,          // true for MathML spacing rules, false for TeX␣
↪rules
    skipAttributes: {},            // RFDa and other attributes NOT to copy to the␣
↪output
    exFactor: .5,                  // default size of ex in em units
    displayAlign: 'center',        // default for indentalign when set to 'auto'
    displayIndent: '0',            // default for indentshift when set to 'auto'
    fontCache: 'local',            // or 'global' or 'none'
    localID: null,                 // ID to use for local font cache (for single␣
↪equation processing)
    internalSpeechTitles: true,    // insert <title> tags with speech content
    titleID: 0                     // initial id number to use for aria-labeledby␣
↪titles
  }
};
```

Option Descriptions

`fontCache: 'local'`
: This setting determines how the SVG output jax manages characters that appear multiple times in an equation or on a page. The SVG processor uses SVG paths to display the characters in your math expressions, and when a character is used more than once, it is possible to reuse the same path description; this can save space in the SVG image, as the paths can be quite complex. When set to `'local'`, MathJax will cache font paths on an express-by-expression (each expression has its own cache within the SVG image itself), which makes the SVG self-contained, but still allows for some savings if characters are repeated. When set to `'global'`, a single cache is used for all paths on the page; this gives the most savings, but makes the images dependent on other elements of the page. When set to `'none'`, no caching is done and explicit paths are used for every character in the expression.

`internalSpeechTitles: true`
: This tells the SVG output jax whether to put speech text into `<title>` elements within the SVG (when set to `'true'`), or to use an `aria-label` attribute instead. Neither of these control whether speech strings are generated (that is handled by the *Semantic-Enrich Extension Options* settings); this setting only tells what to do with a speech string when it has been generated or included as an attribute on the root MathML element.

The remaining options are described in the *Options Common to All Output Processors* section.

Developer Options

In addition to the options listed above, low-level options intended for developers include the following:

`localID: null`
: This gives the ID prefix to use for the paths stored in a local font cache when `fontCache` is set to `'local'`.

This is useful if you need to process multiple equations by hand and want to generate unique ids for each equation, even if MathJax is restarted between equations. If set to `null`, no prefix is used.

`titleID: 0`
This gives the initial number used to make unique `<title>` ids when `internalSpeechTitles` is `true`. This is useful if you need to process multiple equations by hand and want to generate unique ids for each equation, even if MathJax is restarted between equations.

23.2.3 Options Common to All Output Processors

The following options are common to all the output processors listed above. They are given here with their default values, using the `chtml` block as an example.

```
MathJax = {
  chtml: {
    scale: 1,                     // global scaling factor for all expressions
    minScale: .5,                 // smallest scaling factor to use
    matchFontHeight: true,        // true to match ex-height of surrounding font
    mtextInheritFont: false,      // true to make mtext elements use surrounding font
    merrorInheritFont: false,     // true to make merror text use surrounding font
    mtextFont: '',                // font to use for mtext, if not inheriting (empty␣
↪means use MathJax fonts)
    merrorFont: 'serif',          // font to use for merror, if not inheriting␣
↪(empty means use MathJax fonts)
    unknownFamily: 'serif',       // font to use for character that aren't in MathJax
↪'s fonts
    mathmlSpacing: false,         // true for MathML spacing rules, false for TeX␣
↪rules
    skipAttributes: {},           // RFDa and other attributes NOT to copy to the␣
↪output
    exFactor: .5,                 // default size of ex in em units
    displayAlign: 'center',       // default for indentalign when set to 'auto'
    displayIndent: '0'            // default for indentshift when set to 'auto'
  }
};
```

23.2.4 Option Descriptions

`scale: 1`
The scaling factor for math compaired to the surrounding text. The *CommonHTML* output processor tries to match the ex-size of the mathematics with that of the text where it is placed, but you may want to adjust the results using this scaling factor. The user can also adjust this value using the contextual menu item associated with the typeset mathematics.

`minScale: .5`
This gives a minimum scale factor for the scaling used by MathJax to match the equation to the surrounding text. This will prevent MathJax from making the mathematics too small.

`matchFontHeight: true`
This setting controls whether MathJax will scale the mathematics so that the ex-height of the math fonts matches the ex-height of the surrounding fonts. This makes the math match the surroundings better, but if the surrounding font doesn't have its ex-height set properly (and not all fonts do), it can cause the math to *not* match the

surrounding text. While this will make the lower-case letters match the surrounding fonts, the upper case letters may not match (that would require the font height and ex-height to have the same ratio in the surrounding text as in the math fonts, which is unlikely).

Note that, although this option is available on all the output renderers, it has no effect on SVG output, since that is scaled to match the surrounding height automatically.

mtextInheritFont: false
This setting controls whether <mtext> elements will be typeset using the math fonts or the font of the surrounding text. When false, the *mtextFont* will be used, unless it is blank, in which case math fonts will be used, as they are for other token elements; when true, the font will be inherited from the surrounding text, when possible, depending on the mathvariant for the element (some math variants, such as fraktur can't be inherited from the surroundings).

merrorInheritFont: false
This setting controls whether the text for <merror> elements will be typeset using the math fonts or the font of the surrounding text. When false, the *merrorFont* will be used; when true, the font will be inherited from the surrounding text, when possible, depending on the mathvariant for the element (some math variants, such as fraktur can't be inherited from the surroundings).

mtextFont: ''
This specifies the font family to use for <mtext> elements when *mtextInheritFont* is false (and is ignored if it is true). It can be a comma-separated list of font-family names. If it is empty, then the math fonts are used, as they are with other token elements.

merrorFont: 'serif'
This specifies the font family to use for <merror> elements when *merrorInheritFont* is false (and is ignored if it is true). It can be a comma-separated list of font-family names. If it is empty, then the math fonts are used, as they are with other token elements.

unknownFamily: 'serif'
This specifies the font family to use for characters that are not found in the MathJax math fonts. For exmaple, if you enter unicode characters directly, these may not be in MathJax's font, and so they will be taken from the font specified here.

mathmlSpacing: false
This specifies whether to use TeX spacing or MathML spacing when typesetting the math. When true, MathML spacing rules are used; when false, the TeX rules are used.

skipAttributes: {}
This object gives a list of non-standard attributes (e.g., RFDa attributes) that will **not** be transferred from MathML element to their corresponding DOM elements in the typeset output. For example, with

```
skipAttributes: {
    data-my-attr: true
}
```

a MathML element like <mi data-my-attr="some data">x</mi> will not have the data-my-attr attribute on the <mjx-mi> element created by the CommonHTML output processor to represent the <mi> element (normally, any non-standard attributes are retained in the output).

exFactor: .5
This is the size of an ex in comparison to 1 em that is to be used when the ex-size can't be determined (e.g., when running in a Node application, where the size of DOM elements can't be determined).

displayAlign: 'center'
This determines how displayed equations will be aligned (left, center, or right). The default is 'center'.

displayIndent: 0
This gives the amount of indentation that should be used for displayed equations. The default is 0. A value of

'1em', for example, would introduce an extra 1 em of space from whichever margin the equation is aligned to, or an offset from the center position if the expression is centered. Note that negative values are allowed.

23.2.5 Developer Options

In addition to the options listed above, low-level options intended for developers include the following:

wrapperFactory: null
: The WrapperFactory object instance to use for creating wrappers for the internal MathML objects. This allows you to create a subclass of WrapperFactory and pass that to the output jax. A null value means use the default WrapperFactory class and make a new instance of that.

font: null
: The FontData object instance to use for creating wrappers for the internal MathML objects. This allows you to create a subclass of FontData and pass that to the output jax. A null value means use the default FontData class and make a new instance of that.

cssStyles: null
: The CssStyles object instance to use for creating wrappers for the internal MathML objects. This allows you to create a subclass of CssStyles and pass that to the output jax. A null value means use the default CssStyles class and make a new instance of that.

23.3 Document Options

The options below control the operation of the MathDocument object created by MathJax to process the mathematics in your web page. They are listed with their default values. To set any of these options, include an options section in your MathJax global object.

23.3.1 The Configuration Block

```
MathJax = {
  options: {
    skipHtmlTags: [            //  HTML tags that won't be searched for math
        'script', 'noscript', 'style', 'textarea', 'pre',
        'code', 'annotation', 'annotation-xml'
    ],
    includeHtmlTags: {         //  HTML tags that can appear within math
        br: '\n', wbr: '', '#comment': ''
    },
    ignoreHtmlClass: 'tex2jax_ignore',    //  class that marks tags not to search
    processHtmlClass: 'tex2jax_process',  //  class that marks tags that should be␣
↪searched
    compileError: function (doc, math, err) {doc.compileError(math, err)},
    typesetError: function (doc, math, err) {doc.typesetError(math, err)},
    renderActions: {...}
  }
};
```

23.3.2 Option Descriptions

skipHtmlTags: ['script', 'noscript', 'style', 'textarea', 'pre', 'code', 'annotation', 'annotation-xml']
This array lists the names of the tags whose contents should not be processed by MathJaX (other than to look for ignore/process classes as listed below). You can add to (or remove from) this list to prevent MathJax from processing mathematics in specific contexts. E.g.,

```
skipHtmlTags: {'[-]': ['code', 'pre'], '[+]': ['li']}
```

would remove `'code'` and `'pre'` tags from the list, while adding `'li'` tags to the list.

includeHtmlTags: {br: 'n', wbr: '', '#comment': ''}
This object specifies what tags can appear within a math expression, and what text to replace them by within the math. The default is to allow `<br>`, which becomes a newline, and `<wbr>` and HTML comments, which are removed entirely.

ignoreHtmlClass: 'mathjax_ignore'
This is the class name used to mark elements whose contents should not be processed by MathJax (other than to look for the `processHtmlClass` pattern below). Note that this is a regular expression, and so you need to be sure to quote any *regexp* special characters. The pattern is inserted into one that requires your pattern to match a complete word, so setting `ignoreHtmlClass: 'class2'` would cause it to match an element with `class='class1 class2 class3'` but not `class='myclass2'`. Note that you can assign several classes by separating them by the vertical line character (`|`). For instance, with `ignoreHtmlClass: 'class1|class2'` any element assigned a class of either `class1` or `class2` will be skipped. This could also be specified by `ignoreHtmlClass: 'class[12]'`, which matches `class` followed by either a `1` or a `2`.

processHtmlClass: 'mathjax_process'
This is the class name used to mark elements whose contents *should* be processed by MathJax. This is used to restart processing within tags that have been marked as ignored via the `ignoreHtmlClass` or to cause a tag that appears in the `skipHtmlTags` list to be processed rather than skipped. Note that this is a regular expression, and so you need to be sure to quote any *regexp* special characters. The pattern is inserted into one that requires your pattern to match a complete word, so setting `processHtmlClass: 'class2'` would cause it to match an element with `class='class1 class2 class3'` but not `class='myclass2'`. Note that you can assign several classes by separating them by the vertical line character (`|`). For instance, with `processHtmlClass: 'class1|class2'` any element assigned a class of either `class1` or `class2` will have its contents processed. This could also be specified by `processHtmlClass: 'class[12]'`, which matches `class` followed by either a `1` or a `2`.

compileError: function (doc, math, err) {doc.compileError(math, err)}
This is the function called whenever there is an uncaught error while an input jax is running (i.e., during the document's `compile()` call). The arguments are the `MathDocument` in which the error occurred, the `MathItem` for the expression where it occurred, and the `Error` object for the uncaught error. The default action is to call the document's default `compileError()` function, which sets `math.root` to a math element containing an error message (i.e., `<math><merror><mtext>Math input error<mtext></merror></math>`). You can replace this with your own function for trapping run-time errors in the input processors.

typesetError: function (doc, math, err) {doc.typesetError(math, err)}
This is the function called whenever there is an uncaught error while an output jax is running (i.e., during the document's `typeset()` call). The arguments are the `MathDocument` in which the error occurred, the `MathItem` for the expression where it occurred, and the `Error` object for the uncaught error. The default action is to call the document's default `typesetError()` function, which sets `math.typesetRoot` to a `<span>` element containing the text `Math output error`. You can replace this with your own function for trapping run-time errors in the output processors.

renderActions: {...}

This is an object that specifies the actions to take during the `MathJax.typeset()` (and its underlying `MathJax.startup.document.render()` call), and the various conversion functions, such as `MathJax.tex2svg()` (and their underlying `MathJax.startup.document.convert()` call). The structure of the object is `name: value` pairs separated by commas, where the `name` gives an identifier for each action, and the `value` is an array consisting of a number and zero, one, or two functions, followed optionally by a boolean value.

The number gives the priority of the action (lower numbers are executed first when the actions are performed). The first function gives the action to perform when a document is rendered as a whole, and the second a function to perform when an individual expression is converted or re-rendered. These can be given either as an explicit function, or as a string giving the name of a method to call (the first should be a method of a `MathDocument`, and the second of a `MathItem`). If either is an empty string, that action is not performed. If the function is missing, the method name is taken from the `name` of the action. The boolean value tells whether the second function should be performed during a `convert()` call (when true) or only during a `rerender()` call (when false).

For example,

```
MathJax = {
  options: {
    renderActions: {
      compile: [MathItem.STATE.COMPILED],
      metrics: [MathItem.STATE.METRICS, 'getMetrics', '', false]
    }
  }
};
```

specifies two actions, the first called `compile` that uses the `compile()` method of the `MathDocument` and `MathItem`, and the second called `metrics` that uses the `getMetric()` call for the `MathDocument` when the document is rendered, but does nothing during a `rerender()` or `convert()` call or an individual `MathItem`.

If the first function is given explicitly, it should take one argument, the `MathDocument` on which it is running. If the second function is given explicitly, it should take two arguments, the `MathItem` that is being processed, and the `MathDocument` in which it exists.

The default value includes actions for the main calls needed to perform rendering of math: `find`, `compile`, `metrics`, `typeset`, `update`, and `reset`. These find the math in the document, call the input jax on the math that was located, obtain the metric information for the location of the math, call the output jax to convert the internal format to the output format, insert the output into the document, and finally reset the internal flags so that a subsequent typesetting action will process properly.

You can add your own actions by adding new named actions to the `renderActions` object, or override existing ones by re-using an existing name from above. See the *MathML Support* section for an example of doing this. The priority number tells where in the list your actions will be performed.

Loading extensions may cause additional actions to be inserted into the list. For example, the *ui/menu* component inserts an action to add the menu event handlers to the math after it is inserted into the page.

23.3.3 Developer Options

`OutputJax: null`
: The `OutputJax` object instance to use for this `MathDocument`. If you are using MathJax components, the *startup* component will create this automatically. If you are writing a Node application accessing MathJax code directly, you will need to create the output jax yourself and pass it to the document through this option.

InputJax: null
: The `InputJax` object instance to use for this `MathDocument`. If you are using MathJax components, the *startup* component will create this automatically. If you are writing a Node application accessing MathJax code directly, you will need to create the input jax yourself and pass it to the document through this option.

MmlFactory: null
: The `MmlFactory` object instance to use for creating the internal MathML objects. This allows you to create a subclass of `MmlFactory` and pass that to the document. A `null` value means use the default `MmlFactory` class and make a new instance of that.

MathList: DefaultMathList
: The `MathList` object class to use for managing the list of `MathItem` objects associated with the `MathDocument`. This allows you to create a subclass of `MathList` and pass that to the document.

MathItem: DefaultMathItem
: The `MathItem` object class to use for maintaining the information about a single expression in a `MathDocument`. This allows you to create a subclass of `MathItem` and pass that to the document. The document `Handler` object may define its own subclass of `MathItem` and use that as the default instead. For example, the HTML handler uses `HTMLMathItem` objects for this option.

23.4 Accessibility Extensions Options

MathJax contains several extensions meant to support those who need assistive technology, such as screen readers. See the *Accessibility Components* page for more details. The options that control these extensions are listed below.

- *Semantic-Enrich Extension Options*
- *Complexity Extension Options*
- *Explorer Extension Options*
- *Assisitve-MML Extension Options*

Because the accessibility extensions are controlled by the settings of the MathJax contextual menu, you may use the *Contextual Menu Options* to control whether they are enabled or not. There are settings below that can be used to *disable* the extensions, in case they are loaded automatically, but these are not the settings that control whether the extensions themselves are loaded. That is controlled by the menu settings:

```
MathJax = {
  options: {
    menuOptions: {
      settings: {
        assistiveMml: true;    // true to enable assitive MathML
        collapsible: false;    // true to enable collapsible math
        explorer: false;       // true to enable the expression explorer
      }
    }
  }
};
```

Note that there is no control for the semantic enrichment *per se*, but it is enabled automatically by enabling the collapsible math or the expression explorer.

Although you can load the extensions explicitly using the *Loader Options*, it is probably better to use the men u options above, so that if a user turns the extensions off, they will not incur the network and startup costs of laoding the extensions they will not be using.

23.4.1 Semantic-Enrich Extension Options

This extension coordinates the creation and embedding of semantic information generated by the enrichment process within the MathJax output for use by the other extensions.

The *semantic-enrich* extension adds two actions to the document's default *renderActions* object: an `enrich` action to perform the semantic enrichment, and an `attachSpeech` action to attach speech (if it is being generated) to the output.

The Configuration Block

```
MathJax = {
  options: {
    enableEnrichment: true,  // false to disable enrichment
    sre: {
      speech: 'none',          // or 'shallow', or 'deep'
      domain: 'mathspeak',     // speech rules domain
      style: 'default',        // speech rules style
      locale: 'en'             // the language to use (en, fr, es, de, it)
    },
    enrichError: (doc, math, err) => doc.enrichError(doc, math, err),  // function to␣
↪call if enrichment fails
  }
};
```

Option Descriptions

`enableEnrichment: true`
: This setting controls whether semantic enrichment is applied to the internal MathML representation of the mathematics in the page. This is controlled automatically by the settings of the context menu, so you should not need to adjust it yourself. You can, however, use it to disable semantic enrichment if the *semantic-enrich* component has been loaded automatically and you don't need that.

`sre: {...}`
: This block sets configuration values for the Speech-Rule Engine (SRE) that underlies MathJax's semantic enrichment features. See the SRE documentation for more details.

`enrichError: (doc, math, err) => doc.enrichError(doc, math, err)`
: This setting provides a function that gets called when the semantic enrichment process fails for some reason. The default is to call the MathDocument's `enrichError()` method, which simply prints a warning message in the browser console window. The original (unenriched) MathML will be used for the output of the expression. You can override the defaul tbehavior by providing a function that does whatever you want, such as recording the error, or replacing the original MathML with alterntiave MathML contianing an error message.

Note: As of version 3.1.3, the `enrichSpeech` option has been renamed as `speech` in the `sre` block of the configuration.

23.4.2 Complexity Extension Options

This extension generates a complexity metric and inserts elements that allow the expressions to be collapsed by the user by clicking on the expression based on that metric. Use the `'a11y/complexity'` block of your MathJax configuration to configure the extension.

The *complexity* extension adds a `complexity` action to the document's default *renderActions* object.

The Configuration Block

```
MathJax = {
  options: {
    enableComplexity: true,       // set to false to disable complexity computations
    makeCollapsible: true         // insert mactions to allow collapsing
  }
};
```

Option Descriptions

`enableComplexity: true`
: This setting controls whether the *complexity* extension is to run or not. The value is controlled automatically by the settings of the context menu, so you should not need to adjust it yourself. You can, however, use it to disable it if the *complexity* component has been loaded automatically and you don't need it.

`makeCollapsible: true`
: This setting determines whether the extension will insert `<maction>` elements to allow complex expressions to be "collapsed" so that they take up less space, and produce condensed speech strings that are simpler to listen to. When false, the expression is not altered, but elements are marked (internally) if they would be collapsible.

Developer Options

`identifyCollapsible: true`
: This setting determines whether the complexity numbers computed for each element in the expression should take collapsing into account. If true, parents of collapsible elements will get complexities that reflect the collapsible elements being collapsed. When false, the complexities assume no collapsing will take place.

`Collapse: Collapse`
: The `Collapse` object class to use for creating the `<maction>` elements needed for collapsing complex expressions. This allows you to create a subclass of `Collapse` and pass that to the document.

`ComplexityVisitor: ComplexityVisitor`
: The `ComplexityVisitor` object class to use for managing the computations of complexity values. This allows you to create a subclass of `ComplexityVisitor` and pass that to the document.

23.4.3 Explorer Extension Options

This extension provides support for interactive exploration of expressions within the page. See the *Accessibility Features* page for details about how this works.

The *explorer* extension adds an `explorable` action to the document's default *renderActions* object.

The Configuration Block

```
MathJax = {
  options: {
    enableExplorer: true,                // set to false to disable the explorer
    a11y: {
      speech: true,                      // switch on speech output
      braille: true,                     // switch on Braille output
      subtitles: true,                   // show speech as a subtitle
      viewBraille: false,                // display Braille output as subtitles

      backgroundColor: 'Blue',           // color for background of selected sub-
↪expression
      backgroundOpacity: .2,             // opacity for background of selected sub-
↪expression
      foregroundColor: 'Black',          // color to use for text of selected sub-
↪expression
      foregroundOpacity: 1,              // opacity for text of selected sub-
↪expression

      highlight: 'None',                 // type of highlighting for collapsible sub-
↪expressions
      flame: false,                      // color collapsible sub-expressions
      hover: false,                      // show collapsible sub-expression on mouse␣
↪hovering

      treeColoring: false,               // tree color expression

      magnification: 'None',             // type of magnification
      magnify: '400%',                   // percentage of magnification of zoomed␣
↪expressions
      keyMagnifier: false,               // switch on magnification via key␣
↪exploration
      mouseMagnifier: false,             // switch on magnification via mouse hovering
      align: 'top',                      // placement of magnified expression

      infoType: false                    // show semantic type on mouse hovering
      infoRole: false,                   // show semantic role on mouse hovering
      infoPrefix: false,                 // show speech prefixes on mouse hovering
    }
  }
};
```

Option Descriptions

enableExplorer: true
: This setting controls whether the *explorer* extension is to run or not. The value is controlled automatically by the settings of the context menu, so you should not need to adjust it yourself. You can, however, use it to disable it if the *explorer* component has been loaded automatically and you don't need it.

The a11y options belong roughly to one of the following four categories:

Speech Options

speech: true
Sets if speech output is produced. By default speech is computed for every expression on the page and output once the explorer is started.

braille: true
Sets whether or not Braille is produced and output for an expression.

subtitles: true
This option indicates whether the speech string for the selected sub-expression will be shown as a subtitle under the expression as it is explored.

viewBraille: false
This option indicates whether Braille output will be displayed under the expression as it is explored.

Note: As of version 3.1.3, the `speechRules` option has been broken into two separate options, `domain` and `style`, in the `sre` block of the configuration. See the *Semantic-Enrich Extension Options* above for more.

Highlighting Options

foregroundColor: 'Black'
This specifies the color to use for the text of the selected sub-expression during expression exploration. The color should be chosen from among the following: `'Blue'`, `'Red'`, `'Green'`, `'Yellow'`, `'Cyan'`, `'Magenta'`, `'White'`, and `'Black'`.

foregroundOpacity: 1
This indicates the opacity to use for the text of the selected sub-expression.

backgroundColor: 'Blue'
This specifies the background color to use for the selected sub-expression during expression exploration. The color should be chosen from among the following: `'Blue'`, `'Red'`, `'Green'`, `'Yellow'`, `'Cyan'`, `'Magenta'`, `'White'`, and `'Black'`.

backgroundOpacity: .2
This indicates the opacity to use for the background color of the selected sub-expression.

highlight: 'None'
Chooses a particular highlighter for showing collapsible sub-expressions. Choices are `'None'`, `'Flame'`, and `'Hover'`.

flame: false
This flag switches on the Flame highlighter, which permanently highlights collapsible sub-expressions, with successively darkening background for nested collapsible expressions.

hover: false
This switches on the Hover highlighter that highlights collapsible sub-expression when hovering over them with a the mouse pointer.

Note, that having both `'hover'` and `'flame'` set to true can lead to unexpected side-effects.

treeColoring: false
This setting enables tree coloring, by which expressions are visually distinguished by giving neighbouring symbols different, ideally contrasting foreground colors.

Magnification Options

magnification: 'None'
: This option specifies a particular magnifier for enlarging sub-expressions. Choices are `'None'`, `'Keyboard'`, and `'Mouse'`.

magnify: '400%'
: This gives the magnification factor (as a percent) to use for the zoomed sub-expression when zoomed sub-expressions are being displayed during expression exploration. The default is 400%.

keyMagnifier: false
: Switches on zooming of sub-expressions during keyboard exploration of an expression.

mouseMagnifier: false
: Switches on zooming of sub-expressions by hovering with the mouse pointer.

  Note, using both `'keyMagnifier'` and `'mouseMagnifier` together can lead to unwanted side-effect.

align: 'top'
: This setting tells where to place the zoomed version of the selected sub-expression, when zoomed sub-expressions are being displayed during expression exploration.

Semantic Info Options

Semantic information explorers are a feature that displays some semantic information of a sub-expression when hovering over it with the mouse pointer. Note, multiple information explorers work well together.

infoType: false
: Activates an explorer that investigates the semantic type of sub-expressions. The type is an immutable property of an expression, that is independent of its particular position in a formula. Note, however that types can change depending on subject area of a document.

infoRole: false
: Activates an explorer to present the semantic role of a sub-expression, which is dependent on its context in the overall expression.

infoPrefix: false
: Activates explorer for prefix information, which pertains to the position of a sub-expression. Examples are `'exponent'`, `'radicand'`, etc. These would also be announced during interactive exploration with speech output.

  For more details on these concepts, see also the documentation of the Speech Rule Engine.

Note: While multiple keyboard-based exploration techniques work well together and can be easily employed simultaneously, switching on multiple mouse-based exploration tools can lead to unexpected interactions of the tools and often unpredictable side effects.

23.4.4 Assisitve-MML Extension Options

This extension adds visually hidden MathML to MathJax's output that can be voiced by some screen readers. See the *Screen Reader Support* section for more details on how this works.

The *assisitive-mml* extension is included in all the combined components, and is active by default, so screen reader users will not need to do anything to activate it. There is a menu item that controls whether to insert the assistive MathML, so visual users can turn it off if they wish.

The extension adds an action to the document's default *renderActions* object that does the MathML insertion. You can disable that by using the following configuration.

```
MathJax = {
  options: {
    enableAssistiveMml: false
  }
};
```

23.5 Contextual Menu Options

The *ui/menu* component implements the contextual menu that you get when you right-click (or control-click) on a typeset expression. The settings in the menu are "sticky", which means that they are saved from page to page and session to session (though they web-site specific, so each web site has its own saved settings).

As a page author, you can alter the default settings of the menu by using the `menuOptions` block of the `options` section of your MathJax configuration, as described below.

The *ui/menu* component adds a render action called `addMenu` that attaches the menu event handlers to the typeset output. (It also adds a second render action called `checkLoading` that mediates the loading of extensions needed by the contextual menu. For example, when the assistive *a11y/explorer* component is first activated, MathJax may need to load the *a11y/explorer* component; this render action makes sure that has happened before any math is typeset.)

If you want to disable the contextual menu, you can set the `enableMenu` option to `false`:

23.5.1 The Configuration Block

```
MathJax = {
  options: {
    enableMenu: true,          // set to false to disable the menu
    menuOptions: {
      settings: {
        texHints: true,        // put TeX-related attributes on MathML
        semantics: false,      // put original format in <semantic> tag in MathML
        zoom: 'NoZoom',        // or 'Click' or 'DoubleClick' as zoom trigger
        zscale: '200%',        // zoom scaling factor
        renderer: 'CHTML',     // or 'SVG'
        alt: false,            // true if ALT required for zooming
        cmd: false,            // true if CMD required for zooming
        ctrl: false,           // true if CTRL required for zooming
        shift: false,          // true if SHIFT required for zooming
        scale: 1,              // scaling factor for all math
        inTabOrder: true,      // true if tabbing includes math

        assistiveMml: true,    // true if hidden assistive MathML should be generated␣
↪for screen readers
        collapsible: false,    // true if complex math should be collapsible
        explorer: false,       // true if the expression explorere should be active
      },
      annotationTypes: {
        TeX: ['TeX', 'LaTeX', 'application/x-tex'],
        StarMath: ['StarMath 5.0'],
```

(continues on next page)

(continued from previous page)

```
        Maple: ['Maple'],
        ContentMathML: ['MathML-Content', 'application/mathml-content+xml'],
        OpenMath: ['OpenMath']
      }
    }
  }
};
```

23.5.2 Option Descriptions

enableMenu: true
: This controls whether the MathJax contextual menu will be added to the typeset mathematics or not.

settings: {...}
: These settings give the default menu settings for the page, though a user can change them using the menu. These are described in the comments in the example above.

annotationTypes: {...}
: These are the settings for the "Annotation" submenu of the "Show Math As" menu. If the `<math>` root element has a `<semantics>` child that contains one of the specified annotation formats, the source will be available via the "Show Math As" and "Copy to Clipboard" menus. Each format has a list of possible encodings. For example, the line

  ```
  TeX: ['TeX', 'LaTeX', 'application/x-tex']
  ```

  maps an annotation with an encoding of `TeX`, `LaTeX`, or `application/x-tex` to the "TeX" entry in the "Annotation" sub-menus.

23.5.3 Developer Options

```
MathJax = {
  options: {
    MenuClass: Menu,
    menuOptions: {
      jax: {
        CHTML: null,
        SVG: null
      }
    }
  }
};
```

menuClass: Menu
: The `Menu` object class to use for creating the menu. This allows you to create a subclass of `Menu` and pass that to the document in pace of the default one.

jax: {CHTML: null, SVG: null}
: This lists the output jax instances to be used for the different output formats. These will get set up automatically by the menu code if you don't specify one, so it is only necessary to set these if you want to manage the options specially.

23.6 Safe Extension Options

The *ui/safe* component provides a means of filtering the various attributes of the mathematics on the page so that certain limitations on their content is enforced. This allows you to prevent `javascript:` or `data:` URLs from appearing in `href` attributes, for example, which would otherwise cause potential security issues.

All mathematics processed by MathJax is converted into an internal MathML structure, regardless of its initial format in the page. The *ui/safe* extension works by walking the internal MathML tree for the mathematics and checking the attributes of the nodes in the tree to maker sure they comply with the restrictions you specify.

To load the *ui/safe* extension, add `'ui/safe'` to the `load` array of the `loader` block of your MathJax configuration.

```
window.MathJax = {
  loader: {load: ['ui/safe']},
};
```

The *ui/safe* extension can filter several classes of information: URLs, class names, css IDs, and css style declarations. The filtering for these can each be set to one of three different values: `'all'`, `'safe'` or `'none'`. When set to `'all'` no filtering is performed (all values are allowed); when set to `'none'` the value is always cleared (no value can be set for that attribute); and when set to `'safe'` the values are filtered using additional criteria given in the options, as listed below.

23.6.1 The Configuration Block

```
MathJax = {
  options: {
    safeOptions: {
      allow: {
        //
        //  Values can be "all", "safe", or "none"
        //
        URLs:    'safe',   // safe are in safeProtocols below
        classes: 'safe',   // safe start with mjx- (can be set by pattern below)
        cssIDs:  'safe',   // safe start with mjx- (can be set by pattern below)
        styles:  'safe'    // safe are in safeStyles below
      },
      //
      //  Which URL protocols are allowed
      //
      safeProtocols: {
        http: true,
        https: true,
        file: true,
        javascript: false,
        data: false
      },
      //
      //  Which styles are allowed
      //
      safeStyles: {
        color: true,
        backgroundColor: true,
```

(continues on next page)

(continued from previous page)

```
        border: true,
        cursor: true,
        margin: true,
        padding: true,
        textShadow: true,
        fontFamily: true,
        fontSize: true,
        fontStyle: true,
        fontWeight: true,
        opacity: true,
        outline: true
      },
      lengthMax: 3,                           // Largest padding/border/margin, etc.␣
↪in em's
      scriptsizemultiplierRange: [.6, 1],     // Valid range for scriptsizemultiplier
      scriptlevelRange: [-2, 2],              // Valid range for scriptlevel
      classPattern: /^mjx-[-a-zA-Z0-9_.]+$/,  // Pattern for allowed class names
      idPattern: /^mjx-[-a-zA-Z0-9_.]+$/,     // Pattern for allowed ids
      dataPattern: /^data-mjx-/               // Pattern for data attributes
    }
  }
};
```

23.6.2 Option Descriptions

`allow: {...}`
: These settings control what level of filtering to perform for each of the categories provided. When set to `'all'` no filtering is performed (all values are allowed); when set to `'none'` the value is always cleared (no value can be set for that attribute); and when set to `'safe'` the values are filtered using additional criteria given in the remaining options.

`safeProtocols: {...}`
: This object controls which internet protocols are allowed to be used in URLs within the mathematics (in `href` and `src` attributes). A protocol whose value is give as `true` will be allowed, and one given as `false` will not be. For example, the default is to allow `http:`, `https:`, and `file:` protocols, but not `javascript:` or `data:` protocols. A protocol that is not listed is considered to be `false`.

`safeStyles: {...}`
: This object specifies which CSS style properties are allowed to be specified in the `style` attribute of a MathML node. When set to `true` that style (and any sub-styles of the style) are allowed; when `false` or not listed, the style is not allowed to be specified. For example, since `border` is `true`, the `style` attribute can include `border`, `border-top`, `border-top-width`, and so on. Some style values may be further filtered based on other configuration options.

`lengthMax: 3`
: This specifies the largest dimension allowed for styles like `padding`, `border`, `margin`, etc. These are limited in order to prevent users from making borders that are gigantic, for example. The values of these attributes must have absolute value less than this value (in ems).

`scriptsizemultiplierRange: [.6, 1]`
: This specifies the range of values allowed for the `scriptsizemultiplier` MathML attribute (for `<math>` and `<mstyle>` nodes). These are filtered to prevent users from making super- and subscripts too large (or too small).

scriptlevelRange: [-2, 2]
: This specifies the range of values allowd for the `scriptlevel` MathML attribute (for `<math>` and `<mstyle>` nodes). These are filtered to prevent users from making text that is too large (via negative `scriptlevel`) or too small (via large `scriptlevel`).

classPattern: /^mjx-[-a-zA-Z0-9_.]+$/
: This gives a regular expression used to determine if a class name is allowed to be specified. The default is to allow names starting with `mjx-` and containing letters, numbers, minus, period, and underscore.

idPattern: /^mjx-[-a-zA-Z0-9_.]+$/
: This gives a regular expression used to determine what node `id` values are allowed to be specified. The default is to allow ids starting with `mjx-` and containing letters, numbers, minus, period, and underscore.

dataPattern: /^data-mjx-/
: This gives a regular expression used to determine what `data-` attribute names are allowed to be specified. The default is to allow `data-` attributes whose names begin with `data-mjx-`.

23.6.3 Developer Options

```
MathJax = {
  options: {
    safeOptions: {
      //
      //  CSS styles that have Top/Right/Bottom/Left versions
      //
      styleParts: {
        border: true,
        padding: true,
        margin: true,
        outline: true
      },
      //
      //  CSS styles that are lengths needing max/min testing
      //    A string value means test that style value;
      //    An array gives [min,max] in em's
      //    Otherwise use [-lengthMax,lengthMax] from above
      //
      styleLengths: {
        borderTop: 'borderTopWidth',
        borderRight: 'borderRightWidth',
        borderBottom: 'borderBottomWidth',
        borderLeft: 'borderLeftWidth',
        paddingTop: true,
        paddingRight: true,
        paddingBottom: true,
        paddingLeft: true,
        marginTop: true,
        marginRight: true,
        marginBottom: true,
        marginLeft: true,
        outlineTop: true,
        outlineRight: true,
        outlineBottom: true,
        outlineLeft: true,
        fontSize: [.707, 1.44]
```

(continues on next page)

(continued from previous page)

```
      }
    }
  }
};
```

styleParts: {...}
: This object indicates which safe styles have `Top`/`Right`/`Bottom`/`Left` versions (so that the sub-parts can be properly checked). If you extend the `safeStyles` to include others that have these four sub-properties, be sure to add them here.

styleLengths: {...}
: This object lists the styles that are lengths that need to be tested. A string value means test that style's value (e.g., `borderTop` is set to `'borderTopWidth'`, so the border's width is tested). An array value gives the minimum and maximum value (in ems) that the property can have, and `true` means use `[-lengthMax, lengthMax]` using the `lengthMax` option listed above.

23.7 Startup and Loader Options

MathJax's components system is based on two tools that handler loading the various components and setting up the objects and methods needed to use the loaded components. They both use options to control their actions, as described below.

23.7.1 Loader Options

The *loader* component is the one responsible for loading the requested MathJax components. It is configured using the `loader` block in your MathJax configuration object. The `loader` block can also contain sub-blocks of configuration options for individual components, as described below in *Component Configuration*.

The Configuration Block

In the example below, `Loader` represents the `MathJax.loader` object, for brevity.

```
MathJax = {
  loader: {
    load: [],                                   // array of components to load
    ready: Loader.defaultReady.bind(Loader),    // function to call when everything␣
↪is loaded
    failed: function (error) {                  // function to call if a component␣
↪fails to load
      console.log(`MathJax(${error.package || '?'}): ${error.message}`);
    },
    paths: {mathjax: Loader.getRoot()},         // the path prefixes for use in␣
↪specifying components
    source: {},                                 // the URLs for components, when␣
↪defaults aren't right
    dependencies: {},                           // arrays of dependencies for each␣
↪component
    provides: {},                               // components provided by each␣
↪component
```

(continues on next page)

(continued from previous page)

```
    require: null                                   // function to use for loading␣
↪components
  }
};
```

Option Descriptions

load: []
: This array lists the components that you want to load. If you are using a combined component file, you may not need to request any additional components. If you are using using the *startup* component explicitly, then you will need to list all the components you want to load.

ready: MathJax.loader.defaultReady.bind(MathJax.loader)
: This is a function that is called when all the components have been loaded successfully. By default, it simply calls the *startup* component's *ready()* function, if there is one. You can override this with your own function, can can call `MathJax.loader.defaultReady()` after doing whatever startup you need to do. See also the *Component Configuration* section for how to tie into individual components being loaded.

failed: (error) => console.log(`MathJax(${error.package || '?'}): ${error.message}`)}
: This is a function that is called if one or more of the components fails to load properly. The default is to print a message to the console log, but you can override it to trap loading errors in MathJax components. See also the *Component Configuration* section below for how to trap individual component errors.

paths: {mathjax: Loader.getRoot()}
: This object links path prefixes to their actual locations. By default, the `mathjax` prefix is predefined to be the location from which the MathJax file is being loaded. You can use `[mathjax]/...` to identify a component, and this prefix is prepended automatically for any that doesn't already have a prefix. For example, `input/tex` will become `[mathjax]/input/jax` automatically.

  When the TeX *require* extension is loaded, an additional `tex` path is created in order to be able to load the various TeX extensions.

  You can define your own prefixes, for example,

```
MathJax = {
  loader: {
    paths: {custom: 'https://my.site.com/mathjax'},
    load: ['[custom]/myComponent']
  }
};
```

  which defines a `custom` prefix that you can used to access custom extensions. The URL can even be to a different server than where you loaded the main MathJax code, so you can host your own custom extensions and still use a CDN for the main MathJax code.

  You can define as many different paths as you need. Note that paths can refer to other paths, so you could do

```
MathJax = {
  loader: {
    paths: {
      custom: 'https://my.site.com/mathjax',
      extensions: '[custom]/extensions'
    },
    load: ['[extensions]/myExtension']
```

(continues on next page)

(continued from previous page)

```
  }
};
```

to define the `extensions` prefix in terms of the `custom` prefix.

`source: {}`
This object allows you to override the default locations of components and provide a specific location on a component-by-component basis. For example:

```
MathJax = {
  loader: {
    source: {
      'special/extension': 'https://my.site.com/mathjax/special/extension.js'
    },
    load: ['special/extension']
  }
};
```

gives an explicit location to obtain the `special/extension` component.

`dependencies: {}`
This object maps component names to arrays of names of components that must be loaded before the given one. The *startup* component pre-populates this object with the dependencies among the MathJax components, but you can add your own dependencies if you make custom components that rely on others. For example, if you make a custom TeX extension that relies on another TeX component, you would want to indicate that dependency so that if your extension is loaded via `\require`, for example, the loader will automatically load the dependencies first.

```
MathJax = {
  loader: {
    source: {
      '[tex]/myExtension: 'https://my.site.com/mathjax/tex/myExtension.js'},
    },
    dependencies: {
      '[tex]/myExtension': ['input/tex-base', '[tex]/newcommand', '[tex]/enclose']
    }
  }
};
```

This would cause the *newcommand* and *enclose* components to be loaded prior to loading your extension, and would load your extension from the given URL even though you may be getting MathJax from a CDN.

`provides: {}`
This object indicates the components that are provided by a component that may include several sub-components. For example, the *input/tex* component loads the *newcommand* component (and several others), so the `provides` object indicates that via

```
loader: {
  provides: {
    'input/tex': [
      'input/tex-base',
      '[tex]/ams',
      '[tex]/newcommand',
      '[tex]/noundefined',
      '[tex]/require',
      '[tex]/autoload',
```

(continues on next page)

(continued from previous page)

```
      '[tex]/configmacros'
    ]
  }
}
```

The *startup* component pre-populates this object with the dependencies among the MathJax components, but if you define your own custom components that include other components, you may need to declare the components that it provides, so that if another component has one of them as a dependency, that dependency will not be loaded again (since your code already includes it).

For example, if your custom component *[tex]/myExtension* depends on the *newcommand* and *enclose* components, then

```
MathJax = {
  loader: {
    source: {
      '[tex]/myExtension: 'https://my.site.com/mathjax/tex/myExtension.js'},
    },
    dependencies: {
      '[tex]/myExtension': ['input/tex-base', '[tex]/newcommand', '[tex]/enclose']
    },
    load: ['input/tex', '[tex]/myExtension']
  }
};

will load the `input/tex` component, which provides both
`input/tex-base` and `[tex]/newcommand`, and then load
`[tex]/enclose` before loading your `[tex]/myExtension`.
```

require: null
: This is a function to use for loading components. It should accept a string that is the location of the component to load, and should do whatever is needed to load that component. If the loading is asynchronous, it should return a promise that is resolved when the component is loaded, ortherwise it should return nothing. If there is an error loading the component, it should throw an error.

  If set `null`, the default is to insert a `<script>` tag into the document that loads the component.

  For use in *node* applications, set this value to `require`, which will use node's `require` command to load components. E.g.

```
MathJax = {
  loader: {
    require: require
  }
};
```

Component Configuration

In addition to the options listed above, individual compoments can be configured in the `loader` block by using a sub-block with the component's name, and any of the options listed below. For example,

```
MathJax = {
  loader: {
```

(continues on next page)

(continued from previous page)

```
    load: ['input/tex'],
    'input/tex': {
      ready: (name) => console.log(name + ' ready'),
      failed: (error) => console.log(error.package + ' failed')
    }
  }
};
```

which sets up `ready()` and `failed()` functions to process when the *input/tex* component is either loaded successfully or fails to load.

`ready: undefined`
: This is a function that has an argument that is the name of the component being loaded, and is called when the component and all its dependencies are fully loaded.

`failed: undefined`
: This is a function that has an argument that is a `PackageError` object (which is a subclass of `Error` with an extra field, that being `pacakge`, the name of the component being loaded). It is called when the component fails to load (and that can be because one of its dependencies fails to load).

`checkReady: undefined`
: This is a function that tages no argument and is called when the component is loaded, but before the `ready()` function is called. It can be used o do post-processing after the component is loaded, but before other components are signaled that it is ready. For example, it could be used to load other components; e.g., the *output/chtml* component can use its configuration to determine which font to load, and then load that. If this function returns a promise object, the `ready()` function will not be called until the promise is resolved.

23.7.2 Startup Options

The *startup* component is responsible for creating the objects needed by MathJax to perform the mathematical typesetting of your pages, and for setting up the methods you may need to call in order to do that. It is configured using the `startup` block in your configuration object.

The Configuration Block

In the example below, `Startup` represents the `MathJax.startup` object, for brevity.

```
MathJax = {
  startup: {
    elements: null,           // The elements to typeset (default is document body)
    typeset: true,            // Perform initial typeset?
    ready: Startup.defaultReady.bind(Startup),          // Called when components are␣
↪loaded
    pageReady: Startup.defaultPageReady.bind(Startup),  // Called when MathJax and␣
↪page are ready
    document: document,       // The document (or fragment or string) to work in
    input: [],                // The names of the input jax to use from among those␣
↪loaded
    output: null,             // The name for the output jax to use from among those␣
↪loaded
    handler: null,            // The name of the handler to register from among those␣
↪loaded
```

(continues on next page)

(continued from previous page)

```
    adaptor: null             // The name for the DOM adaptor to use from among those␣
↪loaded
  }
};
```

Option Descriptions

elements: null
: This is either `null` or an array of DOM elements whose contents should be typeset. The elements can either be actual DOM elements, or strings that give CSS selectors for the elements to typeset.

typeset: true
: This determines whether the initial typesetting action should be performed when the page is ready.

ready: MathJax.startup.defaultReady.bind(Startup)
: This is a function that is called when MathJax is loaded and ready to go. It is called by the *loader* when all the components are loaded. The default action is to create all the objects needed for MathJax, and set up the call to the `pageReady()` function below. You can override this function if you want to modify the setup process; see *Performing Actions During Startup* for more details. Note that this function may be called before the page is complete, so unless you are modifying the objects created by the *startup* module, replacing `pageReady()` may be the better choice.

pageReady: MathJax.startup.defaultPageReady.bind(Startup)
: This is a function that is called when MathJax is ready to go and the page is ready to be processed. The default action is to perform the initial typesetting of the page and return the promise that resolves what that is complete, but you can override it to do whatever you would like, though you sholud return the promise from the `MathJax.startup.defaultPageReady()` function if you call it. See *Performing Actions During Startup* for more details and examples of how to do this.

document: document
: This is the document (or fragment or string of serialized HTML) that you want to process. By default (for in-browser use) it is the browser document. When there is no global `document` variable, it is an empty HTML document.

input: []
: This is an array of names of input processors that you want to use, from among the ones that have been loaded. So if you have loaded the code for several input jax, but only want to use the `tex` input jax, for example, set this to `['tex']`. If set to an empty array, then all loaded input jax are used.

output: null
: This is the name of the output processor that you want to use, from among the ones that have been loaded. So if you have loaded the code for several output jax, but only want to use the `svg` output jax, for example, set this to `'svg'`. If set to `null` or an empty string, then the first output jax that is loaded will be used.

handler: null
: This is the name of the document handler that you want to use, from among the ones that have been loaded. Currently, there is only one handler, the HTML handler, so unless you are creating your own handlers, leave this as `null`.

adaptor: null
: This is the name of the DOM adaptor that you want to use, from among the ones that have been loaded. By default the components load the `browser` adaptor, but you can load the `liteDOM` adaptor for use in *node* applications; if you do, it will set this value so that it will be used automatically.

These modules use the global `MathJax` object to determine what you want loaded, and alter that object to include the methods and objects that they set up. The initial value of `MathJax` is saved as `MathJax.config`, and other properties are added to `MathJax` depending on the components that get loaded. For example, the *startup* component adds `MathJax.startup()`, which contains the objects that the *startup* module creates, like the input and output jax, the math ocument object, the DOM adaptor, and so on. See the *MathJax API* documentation for more information.

The `MathJax` variable can also contain configuration blocks intended for individual components when they are loaded. For example, it can have a `tex` block to configure the *input/tex* component. See *Configuring MathJax* for more details.

Note that you must set up the global `MathJax` object **before** loading MathJax itself. If you try to that afterward, you will overwrite the `MathJax` variable, and all the values that MathJax has set in them. See the *Configuring MathJax After it is Loaded* section for more about how to change the configuration after MathJax is loaded if you need to do that.

CHAPTER 24

MathJax in Dynamic Content

This page is under construction

If you are writing a dynamic web page where content containing mathematics may appear after MathJax has already typeset the rest of the page, then you will need to tell MathJax to look for mathematics in the page again when that new content is produced. To do that, you need to use the `MathJax.typeset()` method. This will cause MathJax to look for unprocessed mathematics on the page and typeset it, leaving unchanged any math that has already been typeset.

This command runs synchronously, but if the mathematics on the page uses `\require` or causes an extension to be auto-loaded (via the *autoload* component), this will cause the typeset call to fail. In this case, you should use `MathJax.typesetPromise()` instead. This returns a promise that is resolves when the typesetting is complete.

You should not start more than one typesetting operation at a time, so if you are using `MathJax.typesetPromise()` and will be calling it more than once, you may want to retain the promise it returns and chain your subsequent typeset calls to it. See the *Handling Asynchronous Typesetting* section for more details.

More information will be coming to this section in the future.

CHAPTER 25

Custom Extensions

This page is uncer construction

See the *Building a Custom Component* section for an example of building custom extensions to MathJax.

See also the MathJax Web Demos repository for some customization examples for use in the browser, and the MathJax Node Demos for a custom extension in node.

More information will be coming to this section in the future.

CHAPTER 26

The MathJax Processing Model

This page is under construction

CHAPTER 27

Synchronizing your code with MathJax

This page is under construction

MathJax version 2 used *queues*, *callbacks*, and *signals* as a means of coordinating your code with the actions of MathJax. Version 3 uses the more modern tool know as a promise to synchronize your code with MathJax. See the *Handling Asynchronous Typesetting* section for examples of typesetting using promises.

In addition to promises, MathJax version 3 introduces a *renderActions* configuration option that provides a means of linking into MathJax's processing pipeline. This is a priorities list of functions to call during processing, which includes the default actions of finding the math in the page, compiling it into the internal format, getting font metrics for the surrounding text, typesetting the mathematics, inserting the math into the page, adding menu actions, and so on. You can insert your own functions into this chain to add more functionality, or even remove the existing steps to trim down what MathJax does.

More information will be coming to this section in the future.

CHAPTER 28

Using the MathJax API

This page is under construction

28.1 The Component API

This page is under construction

28.2 The Direct API

This page is under construction

CHAPTER 29

MathJax Frequently Asked Questions

29.1 Which license is MathJax distributed under?

MathJax is distributed under the Apache License, Version 2.0.

29.2 Will MathJax make my page load slower even if there's no math?

It depends on how you have configured and loaded MathJax. The combined component files like *tex-chtml.js* contain a full copy of MathJax and all the components needed for it to process the given input and output format, including all the font data (but not the actual fonts themselves). So these files can be quite large, and can take some time to download. On the other hand, it is a single file (unlike in version 2, where multiple files needed to be loaded), so there

should not be the delays associated with establishing multiple connections to a server. If you use the *async* attribute on the script that loads MathJax, that allows the browser to put off loading MathJax until the rest of the page is ready, so that can help speed up your initial page loading as well.

29.3 Mathematics is not rendering properly in IE. How do I fix that?

Currently, MathJax version 3 only supports IE11, so if you are using an earlier version, you will need to update your copy, or use a different browser.

If you are using IE11, then please open the MathJax homepage at www.mathjax.org in IE to see if that loads correctly. If the MathJax website does not display mathematics properly, there may be an issue with your security settings in Internet Explorer. Please check the following settings:

- "Active Scripting" under the Scripting section should be enabled, as it allows JavaScript to run.
- "Run ActiveX controls and Plugins" should be enabled (or prompted) in the "ActiveX Controls and Plugins" section.
- "Script ActiveX controls marked safe for scripting" needs to be enabled (or prompted) in the same "ActiveX Controls and Plugins" section. Note that it requires a restart of IE if you change this setting.
- "Font Download" has to be enabled (or prompted) in the "Downloads" section. This is required for MathJax to use web-based fonts for optimal viewing experience.

You may need to select Custom Level security to make these changes. If you have verified that the above settings are correct, tried clearing your cache and restarting IE. If you are still experiencing problems with displaying mathematics on `www.mathjax.org`, we would appreciate it if you reported the problem to the MathJax issue tracker so we can look into it. See the section on *issue tracking* for details.

If the MathJax site *does* render properly, this indicates that there may be something wrong with the webpage you were trying to view initially. If you manage that website, then make sure that it is using *the latest version of MathJax*, and that you have included the line

<script src="https://polyfill.io/v3/polyfill.min.js?features=es6"></script>

before the script that loads MathJax itself. If you *don't* manage the website yourself, you may have to report the issue to the maintainers of the site in order to have it resolved.

29.4 What should IE's X-UA-Compatible meta tag be set to?

We strongly suggest to follow Microsoft's suggestion to use `IE=edge`. That is, in the document `<head>` include

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

before any other tags in the `<head>`. This will force all IE versions to use their latest engine which is the optimal setting for MathJax. For more information, see the Microsoft documentation on compatibility modes.

29.5 Some of my mathematics is too large or too small. How do I get it right?

MathJax renders mathematics dynamically so that formulas and symbols are nicely integrated into the surrounding text — with matching font size, margins, and baseline. In other words: it should look right. If your mathematics is too large or too small in comparison to its surroundings, you may be using the incorrect typesetting style. Following

LaTeX conventions, MathJax supports two typesetting styles: in-line and "display" equations (one set off from the paragraph as a separate line). For in-line equations, MathJax tries hard to maintain the inter-line spacing. This means things like fractions and roots are vertically compressed, and smaller fonts are used. Display equations are shown as a separate paragraph and can be rendered with more space and slightly larger fonts. The standard delimiters for in-line equations in TeX notation are `\(...\)`, while those for display equations are `$$...$$` or `\[...\]`, but both types of delimiters can be customized. For how to configure MathJax to scale all mathematics relative to the surrounding text, check our documentation for *Output Processor Options*.

29.6 My mathematics is private. Is it safe to use MathJax?

Yes. MathJax is JavaScript code that is runs within the user's browser, so your site's actual content never leaves the browser while MathJax is rendering. If you are using MathJax from a CDN, it interacts with a web server to get font data and MathJax code, but this is all put together in the browser of the reader. If you have concerns about cross-site scripting, you can access the CDN service using the secure `https` protocol to prevent tampering with the code between the CDN and a browser; or, if you prefer, you can install MathJax on your own web server, or for off-line use. MathJax does not reference scripts from other websites. The MathJax code is, of course, open source which means that you can review it and inspect its integrity.

29.7 Does MathJax support Presentation and/or Content MathML?

MathML comes in two types: Presentation MathML, which describes what an equation looks like, and Content MathML, which describes what an equation means. By default, MathJax works with Presentation MathML and offers an extension for Content MathML, see *the documentation on MathML support*, which has not yet been converted to version 3.

You can also convert your Content MathML expressions to Presentation MathML using `xslt`, see for example David Carlisle's web-xslt collection. A more detailed explanation of the difference between Content and Presentation MathML can be found in the module "Presentation MathML Versus Content MathML" at `cnx.org`.

29.8 How do I create mathematical expressions for display with MathJax?

MathJax is a method to display mathematics. It is not an authoring environment, and so you will need another program to create mathematical expressions. The most common languages for mathematics on the computer are (La)TeX and MathML, and there are many authoring tools for these languages.

LaTeX code is essentially plain text, and so you do not need a special program to write it (although complete LaTeX authoring environments do exist). If you are not familiar with LaTeX, you will need some determination to learn and master the language due to its specialized nature and rich vocabulary of symbols. There are various good tutorials on the net, but there is no one-size-fits-all best one. A good starting point is the TeX User Group, or have a look at the LaTeX Wiki book.

MathML is an XML-based web format for mathematical expressions. MathML3, the latest version, has been an official W3C recommendation since October 2010. MathML is widely supported by Computer Algebra Systems and can be created with a choice of authoring tools, including Microsoft Office with the MathType equation editor. A list of software the supports MathML may be found in The W3C MathML software list.

29.9 I ran into a problem with MathJax. How do I report it?

See the section on *Reporting Issues* for the steps to take when you think you have found a bug in MathJax.

29.10 Why doesn't the TeX macro `\something` work?

It really depends on what `\something` is. We have a full list of the *Supported TeX/LaTeX commands*. If the command you want to use is not in this list, you may be able to define a TeX macro for it yourself, or if you want to get really advanced, you can define custom JavaScript that implements it (see the *Custom Extensions* section for details).

Keep in mind that MathJax is meant for typesetting **math** on the web. It only replicates the math functionality of LaTeX and not the text formatting capabilities. Any text formatting on the web should be done in HTML and CSS, not TeX. If you would like to convert full TeX documents into HTML to publish online, you should use a TeX to HTML converter like LaTeXML, Tralics, or tex4ht, but you should realize that TeX conversion tools are unlikely produce results as good as controlling the HTML and CSS source yourself.

29.11 Does MathJax support user-defined TeX macros?

Yes, you can define TeX macros in MathJax the same way you do in LaTeX with `\newcommand`, or `\def`. An example is `\newcommand{\water}{{\rm H_{2}O}}`, which will output the chemical formula for water when you use the `\water` command. The `\renewcommand` command works as well. You can also store macros in the MathJax configuration. For more information, see *the documentation*.

CHAPTER 30

MathJax Badges

We are proud of the work we have done on MathJax, and we hope you are proud to use it. If you would like to show your support for the MathJax project, please consider including one of our "Powered by MathJax" web badges on your pages that use it.

30.1 The MathJax Badges

Thanks to our friends at OER Glue for designing the last two badges.

30.2 The MathJax Logo

30.3 Alternative versions

While we do not allow the modification of the badges or the logo, we are open to requests for different versions.

- An SVG version of the square badge is available.
- Smaller versions of the main logo are available
  - 96x20
  - 60x20
  - 60x12
  - 60x12 (gif)

30.4 Rules

We are committed to maintaining the highest standards of excellence for MathJax, and part of that is avoiding confusion and misleading impressions; therefore, if you do use our badge or logo, we ask that you observe these simple rules (for the fine print, see below):

30.4.1 Things You Can Do

- Use the MathJax Logo or Badges in marketing, and other publicity materials related to MathJax.
- Distribute unchanged MathJax products (code, development tools, documentation) as long as you distribute them without charge.
- Describe your own software as "based on MathJax technology", or "incorporating MathJax source code" if your software includes modified MathJax products.
- Link to MathJax's website(s) by using the logos and badges we provide.
- Use MathJax's word marks in describing and advertising your services or products relating to a MathJax product, so long as you don't do anything that might mislead customers. For example, it's OK if your website says, "Customization services for MathJax available here".
- Make t-shirts, desktop wallpaper, or baseball caps though only for yourself and your friends (meaning people from whom you don't receive anything of value in return).

30.4.2 Things You Cannot Do

- Alter our logo or badges in any way.
- Use our logo or badge online without including the link to the MathJax home page.
- Place our logo or badges in such close proximity to other content that it is indistinguishable.
- Make our logo or badges the most distinctive or prominent feature on your website, printed material or other content.
- Use our logo or badges in a way that suggests any type of association or partnership with MathJax or approval, sponsorship or endorsement by MathJax (unless allowed via a license from us).
- Use our logo or badges in a way that is harmful, deceptive, obscene or otherwise objectionable to the average person.
- Use our logo or badges on websites or other places containing content associated with hate speech, pornography, gambling or illegal activities.
- Use our logo or badges in, or in connection with, content that disparages us or sullies our reputation.

30.4.3 And now the fine print:

The words and logotype "MathJax," the MathJax badges, and any combination of the foregoing, whether integrated into a larger whole or standing alone, are MathJax's trademarks. You are authorized to use our trademarks under the terms and conditions above, and only on the further condition that you download the trademarks directly from our website. MathJax retains full, unfettered, and sole discretion to revoke this trademark license for any reason whatsoever or for no specified reason.

CHAPTER 31

Articles and Presentations

31.1 Articles

- Towards Universal Rendering in MathJax (W4A 2016) by Davide Cervone, Peter Krautzberger, Volker Sorge, *MathJax*, *2016*
- Towards ARIA Standards for Mathematical Markup (DEIMS 2016) by Davide Cervone, Peter Krautzberger, Volker Sorge, *MathJax*, *2016*
- Employing Semantic Analysis for Enhanced Accessibility Features in MathJax (ADS, CCNC 2016) by Davide Cervone, Peter Krautzberger, Volker Sorge, *MathJax*, *2016*
- Whitepaper: Towards MathJax v3.0 by Peter Krautzberger, Davide Cervone, Volker Sorge, *MathJax*, *2015*
- Towards Meaningful Visual Abstraction of Mathematical Notation (MathUI, CICM 2015) by Davide Cervone, Peter Krautzberger, Volker Sorge, *MathJax*, *2016*
- MathML forges on by Peter Krautzberger, *MathJax*, *2014*
- MathJax: A Platform for Mathematics on the Web", Notices of the AMS by Davide Cervone, *MathJax*, *2012*
- Accessible Pages with MathJax by Neil Soiffer *Design Science, Inc.*, *2010*
- Mathematics E-learning Community Benefits from MathJax by Hylke Koers, *MathJax*, *2010*

31.2 Presentations

- Barriereereie Mathematik im Netz (in German) by Peter Krautzberger, *MathJax*, *FernUni Hagen, Global Accessibility Awareness Day, 2016*
- Evolving Math Web Standards from a Usability Perspective by Peter Krautzberger, Davide Cervone, Volker Sorge, *MathJax*, *2016 Joint Mathematics Meetings in Seattle*
- MathJax – beautiful mathematics on the web by Peter Krautzberger, *MathJax*, *2014*
- MathML: math made for the web and beyond by Peter Krautzberger, *MathJax*, *2013*

- MathJax: The Past and the Future by Davide P. Cervone *2013 Joint Mathematics Meetings in San Diego*
- MathJax from an Author's Point of View by Davide P. Cervone *2013 Joint Mathematics Meetings in San Diego*
- MathJax: a JavaScript-based engine for including TeX and MathML in HTML by Davide P. Cervone *2010 Joint Mathematics Meetings in San Francisco*
- MathType, Math Markup, and the Goal of Cut and Paste by Robert Miner *2010 Joint Mathematics Meetings in San Francisco*

CHAPTER 32

Upgrading from v2 to v3

MathJax v3 is a complete rewrite of MathJax from the ground up (see *What's New in MathJax v3.0*), and so its internal structure is quite different from that of version 2. That means MathJax v3 is **not** a drop-in replacement for MathJax v2, and upgrading to version 3 takes some adjustment to your web pages. The sections below describe the changes you will need to make, and the most important differences between v2 and v3.

Warning: If you are using the `latest.js` feature of MathJax v2 on a CDN, note that this will **not** update to version 3 automatically, since there are significant and potentially breaking changes in version 3. There is, however, a bug in `latest.js` in versions 2.7.5 and below; when the current version is 3.0 or higher, `latest.js` will not use the highest version of 2.x, but instead will use the version from which `latest.js` has been taken. For example, if you load `latest.js` from version 2.7.3, it currently is giving you version 2.7.5 as the latest version, when version 3 is released to the CDN, your pages will revert to using version 2.7.3 again. This behavior has been corrected in version 2.7.6, so if you change to loading `latest.js` from version 2.7.6, you should get the latest 2.x version regardless of the presence of version 3 on the CDN.

MathJax v3 is still a work in progress; not all features of version 2 have been converted to version 3 yet, and some may not be. MathJax v2 will continue to be maintained as we work to move more features into version 3, but MathJax v2 likely will not see much further development, just maintenance, once MathJax v3 is fully converted.

32.1 Configuration Changes

There are a number of changes in version 3 that affect how MathJax is configured. In version 2, there were several ways to provide configuration for MathJax; in MathJax 3, when you are using *MathJax components*, there is now only one, which is to set the `MathJax` global to contain the configuration information prior to loading MathJax. In particular, you no longer call `MathJax.Hub.Config()`, and this function does not exist in MathJax v3. See the section *Configuring MathJax* for more details on how to configure MathJax.

In addition to requiring the use of the `MathJax` global variable for setting the configuration, the organization of the configuration options have been changed to accommodate the new internal structure of MathJax, and some of their names have changed as well. To help you convert your existing version 2 configurations to version 3, we provide a conversion tool that you can use to obtain a version 3 configuration that is as close as possible to your current one.

Not all configuration parameters can be converted directly, however. For some of these, it is because the version 2 features have not yet been ported to version 3, but for others, the version 2 feature may simply not exist in the new architecture of version 3. For example, MathJax v2 updates the page in phases, first removing the math source expressions (e.g., the TeX code), then inserts a preview expression (fast to create, but not as accurately laid out), and then goes back and produces high-quality typeset versions, which it inserts in chunks between page updates. MathJax version 3 does not work that way (it does not change the page until the math is entirely typeset), and so the options that control the math preview and the chunking of the equations for display simply have no counterparts in version 3.

Finally, configurations that change the MathJax code via augmenting the existing MathJax objects, or that hook into MathJax's processing pipeline via `MathJax.Hub.Register.StartupHook()` or one of the other hook mechanisms will not carry over to version 3. MathJax v3 does not use the queues, signals, and callbacks that are central to version 2, so code that relies on them will have to be updated. See the *Configuring and Loading MathJax* section for some approaches to these issues.

32.2 Changes in Loading MathJax

Just as there are changes in how MathJax is configured, there are also changes in how MathJax is loaded. With version 2, you load `MathJax.js` and indicate a combined configuration file using `?config=` followed by the name of the configuration file. This always required at least two files to be loaded (and often more than that), and the second file was always loaded asynchronously, meaning MathJax always operated asynchronously.

In version 3, there is no longer a `MathJax.js` file, and you load a combined component file directly. E.g., you load `tex-chtml.js` to get TeX with CommonHTML output. This reduces the number of files that need to be requested, and improves performance. See *Loading MathJax* for more details.

Just as there is no need to use `?config=` in version 3, the other parameters that could be set in this way also are absent from version 3. So, for example, you can't set `delayStartupUntil` in the script that loads MathJax.

The startup sequence operates fundamentally differently in version 3 from how it did in version 2. In version 2, MathJax would begin its startup process immediately upon MathJax being loaded, queuing action to perform configuration blocks, load extensions and jax, do the initial typesetting, and so on. It was difficult to insert your own actions into this sequence, and timing issues could occur if you didn't put your configuration in the right place.

In version 3, synchronization with MathJax is done through ES6 promises, rather than MathJax's queues and signals, and MathJax's startup process is more straight-forward. You can insert your own code into the startup process more easily, and can replace the default startup actions entirely, if you wish. The actions Mathjax takes during startup are

better separated so that you can pick and choose the ones you want to perform. See the *Startup Actions* section for more details on how to accomplish this.

32.3 Changes in the MathJax API

Because the internals have been completely redesigned, its *API* has changed, and so if you have been calling MathJax functions, or have modified MathJax internals by augmenting the existing MathJax objects, that code will no longer work with version 3, and will have to be modified. Some of the more important changes are discussed below.

- The `Mathjax.Hub.Typeset()` function has been replaced by the `MathJax.typesetPromise()` and `MathJax.typeset()` functions. In fact, the `MathJax.Hub` has been removed entirely.
- The queues, signals, and callbacks that are central to version 2 have been replaced by ES6 promises in version 3. In particular, you can use `MathJax.startup.promise` as a replacement for `MathJax.Hub.Queue()`. See the *Handling Asynchronous Typesetting* section for how this is done. See the *Version 2 Compatibility Example* below for code that may make it possible for you to use your version 2 code in version 3.
- The `MathJax.Hub.Register.StartupHook()` and other related hooks have been replaced by `ready()` functions in the *loader* component. So code that relies on these hooks to alter MathJax need to be reworked. The *Startup Actions* section shows some mechanisms that can be used for this.
- Version 2 configurations could include an `Augment()` block that could be used to add (or override) methods and data in the main MathJax objects. In version 3, this should be handled through subclassing the MathJax object classes, and passing the new classes to the objects that use them. This can be done during the *startup* component's `ready()` function, when the MathJax classes are available, but before any of their instances have been created. See the *Startup Actions* section for some ideas on how this can be done.
- The `Augment` configuration blocks and `StartupHooks()` function described above could be used in version 2 to extend MathJax's capabilities, and in particular, to extend the TeX input jax by adding new javascript-based macros. These version-2 mechanisms are not available in version 3; instead, TeX extensions are more formalized in version 3. See the *Building a Custom Component* section for an example of how this can be done.
- In version 2, the mathematics that is located by MathJax is removed from the page and stored in special `<script>` tags within the page. These are not visible to the reader, but mark the location and content of the math on the page. It was possible in version 2 for programs to create these `<script>` tags themselves, avoiding the need for MathJax to look for math delimiters, and for the page author to encode HTML special characters like `<`, `>`, and `&` in their mathematics. Version 3 does not alter the document in this way, and does not store the math that it locates in tags in the page. Instead, it keeps an external list of math objects (of the `MathItem` class). So if you wish to use such scripts to store the math in the page initially, you can replace the `find` action in the *renderActions* list to use a function that locates the scripts and creates the needed `MathItem` objects. For example

```
MathJax = {
  options: {
    renderActions: {
      find: [10, function (doc) {
        for (const node of document.querySelectorAll('script[type^="math/tex"]'))
↪{
          const display = !!node.type.match(/; *mode=display/);
          const math = new doc.options.MathItem(node.textContent, doc.inputJax[0],
↪ display);
          const text = document.createTextNode('');
          node.parentNode.replaceChild(text, node);
          math.start = {node: text, delim: '', n: 0};
```

(continues on next page)

(continued from previous page)

```
          math.end = {node: text, delim: '', n: 0};
          doc.math.push(math);
        }
      }, '']
    }
  }
};
```

should find the scripts that MathJax version 2 normally would have created.

Note that this will *replace* the standard `find` action that looks for math delimiters with this one that looks for the MathJax v2 script tags instead. If you want to do *both* the original delimiter search *and* the search for script tags, then change the `find:` above to `findScript:` so that it doesn't replace the default `find` action. That way, both actions will occur.

32.4 Changes in Input and Output Jax

The input and output processors (called "jax") are core pieces of MathJax. All three input processors from version 2 are present in version 3, but the *AsciiMath* processor has not been fully ported to version 3, and currently consists of the legacy version 2 code patched onto the version 3 framework. This is larger and less efficient than a full version 3 port, which should be included in a future release.

In version 2, MathJax used preprocessors (*tex2jax*, *mml2jax*, *asciimath2jax*, and *jsMath2jax*) to locate the mathematics in the page and prepare it for the input jax. There was really no need to have these be separate pieces, so in version 3, these have been folded into their respective input jax. That means that you don't load them separately, and the configuration options of the preprocessor and input jax have been combined. For example, the `tex2jax` and `TeX` options now both occur in the `tex` configuration block.

MathJax version 2 included six different output jax, which had been developed over time to serve different purposes. The original HTML-CSS output jax had the greatest browser coverage, but its output was browser-dependent, its font detection was fragile, and it was the slowest of the output processors. The CommonHTML output jax was a more modern remake of the HTML output that was both browser independent, and considerably faster. The SVG output jax produced SVG images rather than HTML DOM trees, and did not require web fonts in order to display the math, so the results could be made self-contained. MathJax version 3 includes the CommonHTML and SVG output jax, but has dropped the older, slower HTML-CSS output format.

MathJax 2 also included an output format that produced MathML for those browsers that support it. Since only Firefox and Safari currently implement MathML rendering (with no support in IE, Edge, or Chrome), and because MathJax can't control the quality or coverage of the MathML support in the browser, MathJax version 3 has dropped the NativeMML output format for now. Should the browser situation improve in the future, it could be added again. See *MathML Support* for more on this, and for an example of how to implement MathML output yourself.

There are few changes within the supported input and output jax, as described below:

32.4.1 Input Changes

There are two changes in the TeX input jax that can affect backward compatibility with existing TeX content in your pages.

The first concerns the `\color` macro; in version 2, `\color` is a non-standard in that it takes two arguments (the color an the math to be shown in that color), while the authentic LaTeX version is a switch that changes the color of everything that follows it. The LaTeX-compatible one was available as an extension. In version 3, both versions are

extensions (see), with the LaTeX-compatible one being autoloaded when `\color` is first used. See the *color* and *colorv2* extensions for more information, and how to configure MathJax to use the original version-2 `\color` macro.

The other incompatibility is that the names of some extensions have been changed in version 3. For example, *AMScd* in version 2 is now *amscd* in version 3. This means that you need to use `\require{amscd}` rather than `\require{AMScd}` to load the *CD* environment. In order to support existing content that uses `\require`, you can use the code in the *Version 2 Compatibility Example* section below.

Some other changes include:

- The *autoload-all* extension has been rename *autoload*, and is more flexible and configurable than the original.
- There are two new extensions, *braket* and *physics*.
- The configuration options for controlling the format of equation numbers have been moved to an extension; see the *tagformat* documentation for details.
- The `useMathMLspacing` options for the various input jax have been moved to the output jax instead, as the `mathmlSpacing` option.
- The `processEscapes` option for the *tex2jax* preprocessor (now for the TeX input jax) had a default value of `false` in version 2, but has default value `true` in version 3.
- The functionality of the *MathChoice* extension has been moved to the base TeX package.
- The non-standard `UPDIAGONALARROW` and `ARROW` notations have been removed from the `menclose` element. These have been replaced by the standard `northeastarrow` notation.

32.4.2 Output Changes

There are several important changes to the output jax in version 3, and several things that aren't yet implemented, but will be in a future version. One such feature is linebreaking, which hasn't been ported to version 3 yet. Another is that only the MathJax TeX font is currently available in version 3. See *Not Yet Ported to Version 3* for a list of features that are still being converted.

In addition, there a few other changes of importance:

- There are no more image fonts. These were for use with the HTML-CSS output jax, and since that is not included in MathJax version 3, neither are the image fonts. Since those took up a lot of disk space, this should make locally hosted MathJax installations smaller.
- For expressions with equation numbers, the SVG output jax now has these expressions float with the size of the container element, just like they do in HTML output. This was not the case in version 2, so this is an important improvement for dynamic pages.
- The font used for characters that aren't in the font used by MathJax used to be controlled by the `undefinedFont` configuration parameter in version 2, but in version 3, you should use CSS to set this instead. For example,

```
mjx-container mjx-utext {
  font-family: my-favorite-font;
}
mjx-container svg text {
  font-family: my-favorite-font;
}
```

  would select the `my-favorite-font` to be used for unknown characters. The first declaration is for the CommonHTML output, and the second for the SVG output. Once advantage of this approach is that you can specify the CSS separately for each variant; e.g.,

```
mjx-container mjx-utext[variant="sans-serif"] {
  font-family: my-sans-serif-font;
}
mjx-container svg text[data-variant="sans-serif"] {
  font-family: my-sans-serif-font;
}
```

would set the font to use for characters that aren't in the MathJax fonts and that have requested the sans-serif variant.

- Version 3 only implements the CommonHTML and SVG output jax. The original HTML-CSS output jax has been dropped, a has the NativeMML. The PreviewHTML and PlainSource output jax have not been ported to version 3, though they may be in the future, if there is interest.

32.5 No Longer Applies to Version 3

A number of version 2 features have been removed as part of the redesign of MathJax version 3. These are described below.

- In version 3, MathJax no longer updates the page in small "chunks", but instead updates the page as a whole (a future version may include an extension that updates in smaller pieces). This has an impact on a number of version 2 features. First, because there is no incremental update, the MathJax message bar (usually in the lower left corner) that indicated the progress of the typesetting is no longer needed, and is not part of MathJax version 3. Of course, the configuration options that control it have also been removed, as have the options for equation chunking (that controlled how many equations to process between screen updates.
- Similarly, since the page updating is done all at once, there is no need for the math preview versions that were displayed while the equations where being typeset. So the *fast-preview* extension and *PreviewHTML* output jax have been removed, along with the configuration options for them.
- The *PlainSource* output jax has not be ported to version 3, though it may be in the future; it can be handled in other ways in version 3. As mentioned above, the *NativeMML* has been dropped from version 3, though it is not hard to *implement a replacement* if you want.
- The *autobold* TeX extension is no longer available in version 3, and is unlikely to be ported in the future.
- The *mhchem* TeX extension in version 2 came in two forms: the original extension that didn't match the LaTeX implementation perfectly, and a rewrite by the author of the original LaTeX package that made it compatible with LaTeX. The legacy version could be selected by a configuration option. This is no longer possible in version 3 (the legacy version is no longer provided).
- The *handle-floats* extension for HTML output has been removed, as its functionality is now part of the standard CommonHTML output.
- The *jsMath2jax* preprocessor has been dropped. This was used to help bridge jsMath users to MathJax, but since it has been a decade since MathJax was introduced, the need for jsMath conversion should be very small at this point.
- The *MatchWebFonts* extension is no longer available. This was sometimes needed for HTML-CSS output, which relied on the fonts being in place when it ran. The CommonHTML output is less susceptible to font issues, and this is no longer necessary.
- The *FontWarnings* extension is no longer available, since it was for the HTML-CSS output jax, which is not part of MathJax version 3.
- The *HelpDialog* extension is not included in version 3. Its functionality is incorporated into the *ui/menu* directly.

- The *toMathML* extension is no longer provided in version 3. Instead, you can use `MathJax.startup.toMML()` if you are using MathJax components, or can use the `SerializedMMLVisitor` object if you are calling MathJax modules directly.
- The configuration blocks no longer allow the `style` option that were available in version 2. Instead, you should use CSS stylesheets and CSS style files directly.
- Synchronization with MathJax in version 2 was handled via queues, signals, and callbacks. In version 3, these have been replaced by ES6 promises. See *Synchronizing your code with MathJax* for more details.

32.6 Not Yet Ported to Version 3

As MathJax 3 is still a work in progress, not all of the version 2 features have been converted to the new code base yet, though we hope to include them in version 3 in a future release. Among the most important ones are the following.

- Currently, automatic line breaking support is missing from version 3. This is a key feature to be included in a future release.
- The MathJax v3 output jax currently only support one font, the MathJax TeX fonts. Improved font support is an important goal for version 3, and this is one of the next features to be addressed. We will be rebuilding the fonts used for MathJax, and making additional web fonts available in a future release. We also plan to make the tools used for creating the necessary font data available for use in porting your own fonts for use with MathJax.
- The localization mechanism available in version 2 has not yet been incorporated into version 3, so currently MathJax v3 is available only in English. This is an important feature that will be added to MathJax v3 in a future release.
- The *Safe* extension has not yet been ported to version 3, but should be for a future release.
- The *begingroup* and *mediawiki-texvc* TeX extensions haven't been ported to version 3 yet, but should be in the future.
- The *mml3* and *content-mml* extensions for the MathML input jax are not yet available in version 3. We do hope to have these in a future release.
- The *auto-collapse* assistive extension is not yet available for version 3. If there is enough interest, that will also be ported to the new code base.

32.7 Contextual Menu Changes

The contextual menu has been reorganized to make it easier to access some functions, and to add new ones. One major new features is the *Copy to Clipboard* submenu, which mirrors the *Show Math As* menu, but sends the output to the clipboard rather than displaying it on screen. This is a feature that has been requested for a long time, and we are pleased to be able to offer it in version 3.

There is also a new *Reset to defaults* item that resets all the saved settings to their original values (effectively clearing any custom settings).

The contextual menu now stores its data using the `localStorage` object in the browser, rather than using cookies like version 2 does. This should be more efficient and more secure, but does mean older browsers may not be able to save their settings from session to session (if they don't support `localStorage`).

The accessibility menu options are now built into the contextual menu, so there is no longer an *accessibility-menu* extension. They also have been reorganized in the menu to make it easier to access the more important features. The

auto-collapse extension has not yet been ported to version 3, however. The equation explorer has been expanded and improved; see *Accessibility Features* for details.

Finally, the `showMathMenu` and `showMathMenuMSIE` options have been removed. The need for separate handling of the menu in IE is no longer applicable, and you control whether the contextual menu is attached to the typeset mathematics using the `enableMenu` property of the `options` block of the MathJax configuration (see the *Contextual Menu Options* documentation).

32.8 MathJax in Node

Version 2 of MathJax was designed to work in a browser, and relied heavily on the presence of the browser window, document, DOM, and other browser-specific objects. Using MathJax on a server to pre-process mathematics (e.g., to convert a TeX string to an SVG image, for example), was not easy in version 2. The *mathjax-node <https://github.com/mathjax/mathjax-node>* project made that possible, but required a completely different way of interacting with MathJax, and was not as easy to use or as reliable as we would have liked.

Version 3 has server-side use as an important use-case to support, and so it is possible to use MathJax in a *node* application in essentially the same way as in a browser, with only a few minor adjustments to the configuration to allow for that. This should make it much easier to use MathJax on a server, as it will work the same there as for your web-based applications. It is also possible to link to MathJax at a lower level and access the MathJax modules directly. See the section on *using MathJax in node*, and the *MathJax API* for more information on these possibilities.

32.9 Version 2 Compatibility Example

The following example causes the `\color` macro to be the original one from version 2, and sets up the `\require` macro to translate the old package names into the new ones. This should make MathJax v3 handle existing content properly.

Be sure to convert your version-2 configuration to a version-3 one via the conversion tool that we provide.

```
<script>
MathJax = {
  startup: {
    //
    //  Mapping of old extension names to new ones
    //
    requireMap: {
      AMSmath: 'ams',
      AMSsymbols: 'ams',
      AMScd: 'amscd',
      HTML: 'html',
      noErrors: 'noerrors',
      noUndefined: 'noundefined'
    },
    ready: function () {
      //
      //  Replace the require command map with a new one that checks for
      //    renamed extensions and converts them to the new names.
      //
      var CommandMap = MathJax._.input.tex.SymbolMap.CommandMap;
```

(continues on next page)

(continued from previous page)

```
      var requireMap = MathJax.config.startup.requireMap;
      var RequireLoad = MathJax._.input.tex.require.RequireConfiguration.RequireLoad;
      var RequireMethods = {
        Require: function (parser, name) {
          var required = parser.GetArgument(name);
          if (required.match(/[^_a-zA-Z0-9]/) || required === '') {
            throw new TexError('BadPackageName', 'Argument for %1 is not a valid␣
↪package name', name);
          }
          if (requireMap.hasOwnProperty(required)) {
            required = requireMap[required];
          }
          RequireLoad(parser, required);
        }
      };
      new CommandMap('require', {require: 'Require'}, RequireMethods);
      //
      //  Do the usual startup
      //
      return MathJax.startup.defaultReady();
    }
  },
  tex: {
    autoload: {
      color: [],          // don't autoload the color extension
      colorv2: ['color'], // do autoload the colorv2 extension
    }
  }
};
</script>
<script id="MathJax-script" async
 src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js"></script>
```

This uses the `tex-chtml.js` combined component, so change this to whichever one you want.

If your website uses the MathJax API to queue typeset calls via

```
MathJax.Hub.Queue(['Typeset', MathJax.Hub]);
```

for example, these calls will need to be converted to use the *MathJax 3 API*. You may be able to use the following code to patch into MathJax version 3, which provides implementations for `MathJax.Hub.Typeset()`, and `MathJax.Hub.Queue()`. It also flags usages of `MathJax.Hub.Register.StartupHook()` and the other hook-registering commands, and that you have converted your `MathJax.Hub.Config()` and `x-mathjax-config` scripts to their version 3 counterparts (use the conversion tool).

Add the following lines right after the `new CommandMap()` call in the code above:

```
//
// Add a replacement for MathJax.Callback command
//
MathJax.Callback = function (args) {
  if (Array.isArray(args)) {
    if (args.length === 1 && typeof(args[0]) === 'function') {
      return args[0];
    } else if (typeof(args[0]) === 'string' && args[1] instanceof Object &&
               typeof(args[1][args[0]]) === 'function') {
      return Function.bind.apply(args[1][args[0]], args.slice(1));
```

(continues on next page)

(continued from previous page)

```
    } else if (typeof(args[0]) === 'function') {
      return Function.bind.apply(args[0], [window].concat(args.slice(1)));
    } else if (typeof(args[1]) === 'function') {
      return Function.bind.apply(args[1], [args[0]].concat(args.slice(2)));
    }
  } else if (typeof(args) === 'function') {
    return args;
  }
  throw Error("Can't make callback from given data");
};
//
// Add a replacement for MathJax.Hub commands
//
MathJax.Hub = {
  Queue: function () {
    for (var i = 0, m = arguments.length; i < m; i++) {
       var fn = MathJax.Callback(arguments[i]);
       MathJax.startup.promise = MathJax.startup.promise.then(fn);
    }
    return MathJax.startup.promise;
  },
  Typeset: function (elements, callback) {
     var promise = MathJax.typesetPromise(elements);
     if (callback) {
       promise = promise.then(callback);
     }
     return promise;
  },
  Register: {
     MessageHook: function () {console.log('MessageHooks are not supported in version␣
↪3')},
     StartupHook: function () {console.log('StartupHooks are not supported in version␣
↪3')},
     LoadHook: function () {console.log('LoadHooks are not supported in version 3')}
  },
  Config: function () {console.log('MathJax configurations should be converted for␣
↪version 3')}
};
//
//  Warn about x-mathjax-config scripts
//
if (document.querySelector('script[type="text/x-mathjax-config"]')) {
  throw Error('x-mathjax-config scripts should be converted to MathJax global variable
↪');
}
```

With this you may be able to get away with using your existing version 2 code to interact with version 3. But if not, either a more sophisticated compatibility module will be needed, or better yet, convert to the new version 3 API.

CHAPTER 33

What's New in MathJax

33.1 What's New in MathJax v3.1

Version 3.1 includes a number of new features, as well as bug fixes for several issues with version 3.0. These are described below.

33.1.1 TeX Package Name Changes

The names of several tex packages have been changed to conform to a new naming convention. All package names are now entirely in lower case. The mixed case naming used in the past proved to be problematic, and so four extensions have been renamed to all lower case: `amscd`, `colorv2`, `configmacros`, and `tagformat`.

If you are using the component system to load MathJax, the old names will continue to work for now, but the backward-compatibility support may be removed in the future, so you should change the names to their lower case versions for protection against future changed. Note that the names need to be changed in not only in the `tex.packages` array but also in the name of their configuration options, if any, and in the *autoload* configuration (e.g., if you are disabling the autoloading of the `colorv2` extension).

If you are using direct imports of the MathJax modules, you will need to change to the new names now, as there is no backward-compatibility option for that.

33.1.2 TeX Error Formatting

There is a new `formatError` option for the TeX input jax that provides a function that is called when a syntax or other error occurs during the processing of a TeX expression. This can be used to trap the errors for reporting purposes, or to process the errors in other ways. See the *formatError* documenation.

33.1.3 Noundefined Package Options

The `noundefined` package now has configuration options similar to the ones available in the ones available in version 2. These include tha ability to set the text color, background color, and size of the text to use for disoplaying undefined macro names within TeX formulas. See the *noundefined options* for details.

33.1.4 New *textmacros* Package

There is a new *textmacros* package for the TeX input jax that provides support for processing a number of text-mode macros when they appear inside `\text{}` or other similar settings that produce text-mode material. This allows you to quote TeX special characters, create accented characters, change fonts and sizes, add spacing, etc., within text-mode material. See the *textmacros* page for complete details.

33.1.5 New Safe Extension

The *Safe* extension hs now been ported from v2 to v3. This extensions allows you to filter the values used in the attributes of the underlying MathML that is generated from the TeX, AsciiMath, or MathML input. This can be used to prevent certain URLs from being used, or certain CSS styles from being used, etc. See *Typesetting User-Supplied Content* for more details.

33.1.6 New Accessibility Features

MathJax's accessibility code has undergone some internal improvements for speed and reliability. In addition, there is now a localization of the speech output for the German lanugage. The accessibility contextual menu has been updated to include the ability to select the localization language (in the *speech* submenu), and to expose additional features, such as the ability to set the opacity of the foreground and background colors in the *highlight* submenu. Finally, there is a new control panel for managing the Clearspeak preferences available in the *Clearspeak rules* submenu of the *Speech* menu. See the *Accessibility Extension* for more details.

33.1.7 MathML Verification Options

The MathML input jax has the ability to check and report or (sometimes) correct errors in MathML trees, but the options that control this checking were not documented, and could not be changed easily. Version 3.1 exposes these options so they can be set in the configuration block for the MathML input jax.

33.1.8 New Output Configuration Options

There are two new output configuration options, and updated behavior and defaults for two existing options. These options control the fonts used for `<mtext>` and `<merror>` elements. The original `mtextInheritFont` and `merrorInheritFont` properties controlled whether these elements used the same font as the surrounding text, but neither worked properly in version 3.0. This has been fixed in version 3.1 so these now properly cause the surrounding font to be used for the contents of the specified elements when set to `true`.

If these are set to `false`, the new `mtextFont` and `merrorFont` properties specify a font family (or list of families) to use fort the content of these elemements. This allows you to force a specific font to be used for the text within mathematics. If these are set to an empty string, then the MathJax fonts will be used.

The defaults for these are

```
mtextInheritFont: false,
merrorInheritFont: false,
mtextFont: '',
merrorFont: 'serif',
```

which means that the MathJax fonts will be used for `<mtext>` elements, and the browser's serif font will be used for `<merror>` text. See the *Options Common to All Output Processors* for more information.

Note: the default for `merrorInheritFont` has been changed from `true` to `false` now that `merrorFont` is available.

33.1.9 Startup Promise Revisions

The `MathJax.startup.promise` now works in a more intuitive way. In the past, it was initially set to be a promise that resolves when MathJax is ready and the `DOMContentLoaded` event occurs, and was changed by the `startup.pageReady()` function to one that resolve when the initial typesetting is finished. So you could not use `MathJax.startup.promise` to tell when the initial typesetting is complete without overriding the `startup.pageReady()` method as well.

In version 3.1, the `MathJax.startup.promise` has been changed to one that resolves when the action of the `startup.pageReady()` method is finished (which includes the initial typesetting action). That makes this promise a reliable way to determine when the initial typesetting is finished.

See the sections on *Performing Actions During Startup*, on *Handling Asynchronous Typesetting*, and on the *pageReady()* for more details.

33.1.10 New API for Clearing Typeset Content

If you are dynamically adding and removing content from your page, you need to tell MathJax abiout what you are doing so that it can typeset any new mathematics, and forget about any old typeset mathematics that you have removed. In version 3.0, the `MathJax.typesetClear()` method could be used to tell MathJax to forget about *all* the mathematics that is ahs typeset, but if you only removed some of it, there was no easy way to tell it to forget about only the math you removed. This situation has been improved in version 3.1 by allowing the `MathJax.typesetClear()` method to accept an array of elements whose contents should be forgotten. See *Updating Previously Typeset Content* for more details.

33.1.11 New API for Getting Math within a Container

MathJax keeps track of the math that you have typeset using a list of objects called *MathItems*. These store the original math string, the locatino of the math in the document, the input jax used to process it, and so on. In the past, you had access to these through a list stored in the *MathDocument* object stored at `MathJax.startup.document`, but it was not easy to get access to the individual MathItems in a convenient way. In v3.1 there is now a function `MathJax.startup.document.getMathItemsWithin()` that returns all the MathItems for the typeset math within a DOM container element (or collection of DOM elements). See *Looking up the Math on the Page* for details.

33.1.12 Change to SRE Interface

In version 3.0.5, The *a11y/sre* module exposed a value `sreReady` that was a promise that would be resolved when the Speech-Rule Engine was ready to use. Due to changes in SRE (which can now be configured to load localized translation data, and so may become un-ready while that is happening), the `sreReady` value in version 3.1.0 is now a function returning a promise, so should be called as `sreReady()`.

33.1.13 Fixes to the LiteDOM and DOMAdaptors

The *LiteDOM* in version 3.0.5 failed to process comments correctly: they were properly read and ignored, but where not included in the output when the DOM is serialized. In version 3.1.0, this has been fixes so that comments are properly maintained. In addition, the `doctype` of the document is now retained by the *LiteDOM*, and can be accessed by a new `doctype()` method of the *DOMAdaptor* class (and its subclasses).

33.1.14 Updated Demos

The web and node examples have been updated to use the new features available in version 3.1.0, and to include more examples. In particular, the node examples now include demonstrations of using the simpler loading mechanism for node applications, using puppeteer to perform server-side processing, and using JSDOM for server-side processing.

33.2 What's New in MathJax v3.0

MathJax version 3 is a complete rewrite from the ground up, with the goal of modernizing MathJax's internal infrastructure, bringing it more flexibility for use with contemporary web technologies, making it easier to use with NodeJS for pre-processing and server-side support, and making it faster to render your mathematics.

33.2.1 Improved Speed

There were a number of design goals to the version 3 rewrite. A primary one was to improve the rendering speed of MathJax, and we feel we have accomplished that. Because the two versions operate so differently, it is difficult to make precise comparisons, but in tests that render a complete page with several hundred expressions, we see a reduction in rendering time of between 60 and 80 percent, depending on the browser and type of computer.

33.2.2 More Flexibility

Another goal was to make MathJax 3 more flexible for web developers using MathJax as part of a larger framework, while still keeping it easy to use in simple settings. To that end, we have broken down the actions that MathJax takes into smaller units than in version 2, and made it possible to call on them individually, or replace them with alternative versions of your own. For example, the typesetting process has been broken into a number of pieces, including finding

the math in the page, compiling it into the internal format (MathML), getting metric data for the location of the math, converting the math into the output format, inserting it into the page, adding menu event handlers, and so on. You have control over which of these to perform, and can modify or remove the existing actions, or add new ones of your own. See the *renderActions* documentation for details.

33.2.3 Synchronous Conversion

A key feature that we wanted to include in version 3 is the ability to run MathJax synchronously, and in particular, to provide a function that can translate an input string (say a TeX expression) into an output DOM tree (say an SVG image). This was not really possible in version 2, since its operation was inherently asynchronous at a fundamental level. With MathJax version 3, this is straight-forward, as we provide a synchronous typesetting path, both within the page, and for individual expressions, provided you load all the components you need ahead of time. See *Typesetting and Converting Mathematics* for details.

33.2.4 No Queues, Signals, Callbacks

One of the more difficult aspects of working with MathJax version 2 was having to synchronize your actions with those of MathJax. This involved using *queues*, *callbacks*, and *signals* to mediate the asynchronous actions of MathJax. Since these were not standard javascript paradigms, they caused confusion (and headaches) for many developers trying to use MathJax. With version 3, MathJax has the option of working synchronously (as described above), but it still allows for asynchronous operation (e.g., to allow TeX's `\require` command to load extensions dynamically) if you wish. This no longer relies on queues, callbacks, and signals, however. Instead, these actions are managed through the ES6 promise, which is a javascript standard, and should make integrating MathJax into your own applications more straight-forward.

33.2.5 Package Manager Support

Because MathJax version 2 used its own loading mechanism for accessing its components, and because there was no method for combining all the pieces needed by MathJax into one file, MathJax did not work well with javascript packaging systems like `webpack`. Version 3 resolves that problem, so it should interoperate better with modern web workflows. You can make your own custom single-file builds of MathJax (see *Making a Custom Build of MathJax*) or can include it as one component of a larger asset file.

33.2.6 MathJax Components

MathJax 3 still provides a loading mechanism similar to the one from version 2, however, so you can still customize the extensions that is loads, so that you only load the ones you need (though this does require that you use MathJax in its asynchronous mode). The various pieces of MathJax have been packaged into "components" that can be mixed and matched as needed, and which you configure through a global `MathJax` variable (see *Examples in a Browser*). This is how MathJax is being distributed through the various CDNs that host it. When loaded this way, MathJax will automatically set up all the objects and functions that you need to use the components you have loaded, giving you easy access to typesetting and conversion functions for the input and output formats you have selected. See the section on *The MathJax Components* for more information. You can also create your own custom components to complement or replace the ones provided on the CDN (see *A Custom Extension* for more).

33.2.7 Startup Actions

If you use any of the *combined component* files, MathJax will perform a number of actions during its startup process. In particular, it will create the input and output jax, math document, DOM adaptor, and other objects that are needed in order to perform typesetting in your document. You can access these through the `MathJax.startup` object, if you

need to. MathJax will also set up functions that perform typesetting for you, and conversion between the various input and output formats that you have loaded. This should make it easy to perform the most important actions available in MathJax. See *Typesetting and Converting Mathematics* for more details.

33.2.8 Server-Side MathJax

While MathJax 2 was designed for use in a web browser, an important use case that this left unaddressed is pre-processing mathematics on a server. For version 2, we provided mathjax-node to fill this gap, but it is not as flexible or easy to use as many would have liked. MathJax 3 resolves this problem by being designed to work with *node* applications in essentially the same way as in a browser. That is, you can load MathJax components, configure them through the `MathJax` global variable, and call the same functions for typesetting and conversion as you do within a browser. This makes parallel development for both the browser and server much easier.

Moreover, node applications can access MathJax modules directly (without the packaging used for MathJax components). This gives you the most direct access to MathJax's features, and the most flexibility in modifying MathJax's actions. See *Examples of MathJax in Node* for examples of how this is done.

33.2.9 ES6 and Typescript

MathJax 3 is written using ES6 modules and the Typescript language. This means the source code includes type information (which improves the code reliability), and allows MathJax to be down-compiled to ES5 for older browsers while still taking advantage of modern javascript programming techniques. It also means that you can produce pure ES6 versions of MathJax (rather than ES5) if you wish; these should be smaller and faster than their ES5 equivalents, though they will only run in modern browsers that support ES6, and so limit your readership. We may provide both ES6 and ES5 versions on the CDN in the future.

33.2.10 New Features for Existing Components

In addition to the new structure for MathJax described above, some new features have been added to existing pieces of MathJax.

TeX Input Extensions

There are two new TeX input extensions: *braket* and *physics*. Also, some functionality that was built into the TeX input jax in version 2 has been moved into extensions in version 3. This includes the *macros* configuration option, the *tag formatting* configuration options, and the *require* macro. The new *autoload* extension replaces the older *autoload-all* extension, is more configurable, and is included in the TeX input components by default. There a several extensions that are not yet ported to version 3, including the *autobold*, *mediawiki-texvc*, and the third-party extensions.

SVG Output

The SVG output for equations with labels has been improved so that the positions of the labels now react to changes in the container width (just like they do in the HTML output formats).

Improved Expression Explorer

The interactive expression explorer has been improved in a number of ways. It now includes better heuristics for creating the speech text for the expressions you explore, provides more keyboard control of the features in play during your exploration, adds support for braille output, adds support for zooming on subexpressions, and more. See the *Accessibility Features* page for more details.

33.3 What's New inEarlier Versions

33.3.1 What's New in MathJax v2.7

MathJax v2.7 is primarily a bug-fix release with over 60 important bug fixes, in particular to the CommonHTML output. In addition, this release adds several new features as an opt-in. The following are some of the highlights.

Features

- *Common HTML output improvements* Several important bugs in the layout model have been fixed, in particular tabular layout is now much more robust.
- *Accessibility improvements.* After the completion of the MathJax Accessibility Extensions, we are integrating the opt-in for the MathJax menu into the core distribution. We are grateful to the web accessibility community for their guidance, support, and feedback in our efforts towards making MathJax completely accessible to all users. This allows end-users to opt into the following features via the MathJax Menu:
  - *Responsive Equations.* An innovative responsive rendering of mathematical content through collapsing and exploration of subexpressions.
  - *Universal aural Rendering.* An aural rendering tool providing on-the-fly speech-text for mathematical content and its subexpressions using various rule sets.
  - *Full Exploration.* A fully accessible exploration tool, allowing for meaningful exploration of mathematical content including multiple highlighting features and synchronized aural rendering.
  - For more information check the release announcement and the dedicated repository at mathjax/mathjax-a11y.

For a detailed listing please check the release milestone.

Accessibility

- mathajx-dev/#20 Add the Menu extension from the MathJax Accessibility tools to all combined configuration files.
- #1465 CHTML and HTML-CSS output: do not add `role=math` by default.
- #1483 Catch IE8 errors with inserting MathML from AssistiveMML extension.
- #1513 Disable the AssistiveMML extension when the output renderer is PlainSource.

Interface

- #1463 Reset message strings for `messageStyle=simple` for each typeset.
- #1556 Improve menu placement.

- #1627 Add Accessibility submenu.

HTML/SVG/nativeMML display

- #1454 SVG output: Use full location URL for `xlink` references in SVG `<use>` elements.
- #1457 Common-HTML output: Fix problem with characters from Unicode Plane 1 not being mapped to the MathJax fonts properly
- #1458 SVG output: Fix problem with container width when math is scaled.
- #1459 CommonHTML output: Improve `getNode()` to fix processing errors when line-breaking.
- #1460 HTML-CSS output: Adjust position of rule for square root when it is made via `createRule()`.
- #1461 HTML-CSS output: Make sure `0` remains `0` when rounding to pixels (plus a bit).
- #1462 CommonHTML output: Bubble percentage widths up while line breaking.
- #1475 PreviewHTML: Avoid error when `\overset` or `\underset` is empty.
- #1479 All outputs: Properly determine (shrink-wrapping) container widths.
- #1503 CommonHTML output: Handle adjusting table cell heights properly.
- #1507 SVG output: Remove invalid `src` attribute from `<mglyph>` output.
- #1510 CommonHTML output: Prevent CSS bleed-through for box-sizing.
- #1512 CommonHTML output: make `<mglyph>` scale image size by hand.
- #1530 All outputs: Fix problem with Safari inserting line breaks before in-line math.
- #1533 CommonHTML output: improve aligning labels with their table rows.
- #1534 CommonHTML output: ensure output stays a table-cell when focused.
- #1538 All outputs: Don't let preview width interfere with the determination of the container width.
- #1542 CommonHTML output: improve stretching `<mover>` in `<mtd>` elements.
- #1547 HTML-CSS output: improve line breaks within fractions.
- #1549 All outputs: Improve determination of line-breaking parent element.
- #1550 CommonHTML output: Improve vector arrow positioning.
- #1552 All outputs: Handle `href` correctly when line breaking.
- #1574 HTML-CSS and SVG output: Use `currentColor` for `menclose` with no `mathcolor`.
- #1595 CommonHTML output: Properly scale elements with `font-family` specified.

TeX emulation

- #1455 Fix `TeX.Environment()` to use the correct end environment.
- #1464 Make sure `resetEquationNumbers` is always defined.
- #1484 Mark accented operators as not having movable limits.
- #1485 Allow line breaks within `TeXAtom` elements
- #1508 Surround `\middle` with `OPEN` and `CLOSE` TeXAtoms to match TeX spacing
- #1509 Make delimiters (in particular arrows) symmetric for `\left` and `\right`.

- #1514 Don't unwrap rows when creating fenced elements.
- #1523 Don't copy environment into `array` environments.
- #1537 mhchem: add config parameter to select mhchem v3.0.
- #1596 Prevent `\require{mhchem}` to override one already loaded.
- #1551 Allow `<wbr>` in TeX code.
- #1565 Handle `\+SPACE` in macro definitions.
- #1569 Treat control sequences as a unit when matching a macro template.
- #1587 Make sure `trimSpaces()` doesn't remove tailing space in `\+SPACE`.
- #1602 Handle `\ref` properly when there is a `<base>` tag.

Asciimath

- asciimath/f649ba4 Add `newsymbol` command for adding a new symbol object

MathML

- #1505 Handle `rowlines=""` and `rowlines=" "` like `rowlines="none"`.
- #1511 Don't convert attribute to boolean unless the default is a boolean.
- #1526 Make minus in `<mn>` produce `U+2212` rather than `U+002D`.
- #1567 Fix spacing for initial fraction in exponent position.

Fonts

- #1521 STIX fonts: Make left arrow use combining left arrow for accents.
- #1092 STIX fonts: Make `U+222B` (integral) stretchy.
- #1154 STIX fonts: Remap `|` to variant form (with descender) and map variant to original form.
- #1175 Use `U+007C` and `U+2016` for delimiters rather than `U+2223` and `U+2225`.
- #1421 MathJax TeX fonts: Fix SVG font data for stretchy characters.
- #1418 Alias `U+2206` to `U+0394` and remove incorrect `U+2206` from SVG font files.
- #1187 Make height and depth of minus match that of plus (needed for TeX-layout super/subscript algorithm to work properly), and adjust for that when it is used as an extender in stretchy characters.
- #1546 MathJax TeX fonts: Add stretchy data for `U+20D7`.

Localization

- #1604 Updated locales thanks to the contributors at Translatewiki.net; activate locale for Zazaki.

APIs

- #1504 Make `getJaxForMath()` work even during chunking.
- #1522 Add Third Party Extensions Repository to the Ajax paths as `[Contrib]`.
- #1525 Allow MathJax root to be configured.

Misc.

- #1456 Prevent removal of DOM elements while MathJax is running from stopping processing, or to leaving duplicate math in place.
- #1524 Prevent pre-processors from adding duplicate preview elements.
- #1554 Safe extension: Add filtering of CSS styles like `padding`, `margin`.
- #1590 Set previews to have `display:none`.
- #1591 Change `rev=` to `V=` in cache breaking code.

33.3.2 What's New in MathJax v2.6

MathJax v2.6 includes a number of new features, as well a more than 30 important bug fixes. The following are some of the highlights.

Features

- *Improved CommonHTML output.* The CommonHTML output now provides the same layout quality and MathML support as the HTML-CSS and SVG output. It is on average 40% faster than the other outputs and the markup it produces are identical on all browsers and thus can also be pre-generated on the server via MathJax-node. The fast preview mechanism introduced in v2.5 continues to develop as a separate output as *PreviewHTML* and the *fast-preview* extension.
- *Accessibility improvements.* We thank the AT community for their guidance, support, and feedback in our efforts towards making MathJax completely accessible to all users.
  - *Screenreader compatibility.* The new `AssistiveMML` extension enables compatibility with most MathML-capable screenreaders by inserting visually-hidden MathML alongside MathJax's visual output. See *screenreader support* for details on the expected behavior as well as background on the limitations due to lack of web standards and browser/OS technology.
  - *Accesssible UI.* We have improved the accessibility of the MathJax menu to enable assistive technology users to easily access its features, cf. *MathJax UI*.
- *PlainSource Output.* The new PlainSource output will revert the rendering back to the input format; in the case of MathML, the output will prefer TeX and AsciiMath from `<annotation-xml>` elements. This helps with accessibility and copy&paste of document fragments.
- *Semi-slim MathJax repository for bower.* You can now use `bower install components/MathJax` to install a fork of MathJax without PNG fonts. **Many thanks** to @minrk from the IPython/Jupyter team and to the team at components!
- *MathJax via npm.* You can now use `npm install mathjax` to install a copy of MathJax without PNG fonts.

- *Deprecated: MMLorHTML extension.* We have deprecated the `MMLorHTML` extension. For a detailed guide on configuring MathJax to choose different outputs on different browsers, please see *Automatic Selection of the Output Processor* for more information.

Numerous bugs and issues have also been resolved; for a detailed listing please check the release milestone.

Interface

- #938 Expose MathML for accessibility; cf. *screenreader support*.
- #939 Make MathJax contextual menu properly accessible.
- #1088 MathJax Menu: drop PNG images in menu.
- #1210 Update `MathZoom.js`: global border-box support. **Special thanks** to @CalebKester
- #1273 Improve handling of hash in URL.

HTML/SVG/nativeMML display

- #1095 HTML-CSS output: prevent collapse of table borders.
- #596 SVG Output: Fix overlapping equation labels in SVG output
- #994 SVG Output: Change default `blacker` setting to *1*.
- #995 SVG output: fix baseline alignment issues.
- #995 SVG output: fix failure to scale all but the first glyph in a fraction when `useFontCache=false`.
- #1035 PreviewHTML output: fix fractions formatting in WebKit and IE.
- #1233 SVG output: make maligngroup and malignmark produce no output.
- #1282 HTML-CSS output: reduce "bumpiness" of focus outline.
- #1314 HTML-CSS output: prevent clipping of extremely long strings.
- #1316 SVG output: preserve non-breaking space in `mtext` elements.
- #1332 HTML-CSS output: fix width calculations for mrows with embellished operators that could stretch but don't actually.

TeX emulation

- #567 Add macro for `overparen` and `underparen` to provide stretchy arcs above/below
- #956 Simplify the `mhchem` extension to use multiscripts, cf. #1072.
- #1028 Fix spacing in `\alignedat`.
- #1194 Fix problem where automatic numbering affects `\binom` and friends.
- #1199 Fix problem with dot delimiter not being recognized as a delimiter.
- #1224 Handle braces properly in text mode when looking for matching math delimiters.
- #1225 Fix `\operatorname` not ignoring `\limits` that follow immediately after.
- #1229 Fix wrong spacing of trailing binary operators.
- #1272 Fix spacing of `\eqnarray` environment.
- #1295 Handle `scriptlevel` set on arrays via an `mstyle` node (affects `\smallmatrix`).

- #1312 Improve heuristics for adding U+2061 (invisible function application).

Asciimath

- asciimath/#31 Add support for `overparen`, `underparen` to produce `mover` and `munder` constructs.
- asciimath/#35 Add support for `bowtie`, `ltimes` and `rtimes`.
- asciimath/#40 Improve parsing of brackets within brackets.
- asciimath/#43 Improve detection of non-matrices.

MathML

- #1072 Right-justify prescripts in `mmultiscript` elements (after clarification in MathML 3 editors' draft); cf. #956.
- #1089 Fix `toMathML` from changing `<maligngroup>` to `<malign>`
- #1188 Fix `mmultiscripts` with odd number of post-scripts not rendering correctly.
- #1231 Fix `<math>` element not being treated as an `<mrow>` for embellished operator spacing.
- #1233 Make `<maligngroup>` and `<malignmark>` be self-closing in MathML input.
- #1238 Fix Content MathML extension not handling namespace prefixes.
- #1257 Improve `mml3.js`: better RTL support in HTML-CSS; improved IE/Edge compatibility.
- #1323 Content-mathml extension: improve handling of empty Presentation MathML nodes.

Fonts

- #928 Add data for stretchy `U+2322 (FROWN)`, `U+2323 (SMILE)`, and also `U+2312 (ARC)` to be aliases for the top and bottom parentheses. This enables stretchy constructions; cf. also #567.
- #1211 Fix web font detection for Gyre-Pagella etc. in IE10+.
- #1251 Fix primes in STIX-web font being too small in SVG output.

Localization

- #1248 Updated locales thanks to the contributors at Translatewiki.net; activate locales for Bulgarian, Sicilian, Lithuanian, and Laki.

APIs

- #1216 Add debugging tips to console output.

Misc.

- #1074 Fix regression in v2.5 regarding MathPlayer on IE9.
- #1036 Improve CDN rollover behavior.
- #1085 Fix detection of Windows Phone mobile IE.

- #1155 Work around websites using user agent filtering
- #1173 Avoid warning message in debug mode.
- #1208 Fix CHTML preview from setting chunking parameters even when disabled.
- #1214 semi-slim official MathJax repository for bower; use `bower install components/MathJax` for a copy without PNG fonts. Special thanks to @minrk from the IPython/Jupyter team and to the team at components!
- #1254 Improve examples in `/test`: add viewport meta tags, improve dynamic examples.
- #1328 Add package.json for publishing on npm, excluding PNG fonts.

33.3.3 What's New in MathJax v2.5

MathJax v2.5 includes a number of new features, as well a more than 70 important bug fixes. The following are some of the highlights.

Features

- *Speed improvements.* The HTML-CSS output performance was improved by 30-40% (depending on content complexity, with higher gains in more complex content such as very long documents).
- *New output for fast preview.* The new CommonHTML output provides a rough but 10x-faster rendering. The `CHTML-preview` extension will use this fast output as a preview mode for HTML-CSS or SVG output.
- *Improved Content MathML support.* Content MathML is now fully supported via a new extension, in particular this allows customization of the conversion process.
- *Improved elementary math support* The experimental support for elementary math elements has been significantly improved special thanks to David Carlisle.
- *NodeJS compatibility*. Enable the implementation of a NodeJS API (released as MathJax-node).

Numerous display bugs, line-breaking problems, and interface issues have been resolved; for a detailed listing please check the release milestone.

Interface

- #834 Fix incorrect line-width when zooming which can cause line-breaking problems.
- #918 Fix zoom box size in NativeMML output.
- #835 Fix zoom for equations extending beyond their bounding box.
- #893 Fix outdated ARIA values for HTML-CSS and SVG output.
- #860, #502 Preserve RDFa, microdata, aria labels, and other attributes in HTML-CSS and SVG output.
- #935 Escape special characters in TeX annotations.
- #912 Fix missing `mstyle` attributes in `toMathML` output.
- #971 Fix lost attributes when `toMathML` is restarted.

Line-breaking

- #949 Fix processing error due to empty elements.

HTML-CSS/SVG/nativeMML display

- #863 Fix broken MathML preview in MathML pre-processor.
- #891 Fix deprecated regexp affecting mtable alignment.
- #323 Improve MathPlayer compatibility on Internet Explorer 10+.
- #826 Scale content in fallback fonts.
- #898 Fix invalid SVG output when using fallback characters.
- #800 Fix misplaced background color for stretched mphantom elements in SVG output.
- #490 Fix `\overline` issues in combination with text-style limits.
- #829 Implement `\delimitershortfall`, `\delimiterfactor`.
- #775 Fix lost text content in SVG output.
- #917 Fix cases of incorrect bounding boxes in HTML-CSS output.
- #807 Fix clipping of table columns in HTML-CSS output.
- #804 Fix cases of uneven subscripts.
- #944 Fix rendering error when scaling-all-math of labeled equations.
- #930 Fix SVG output failure when `<math>` element has inline styles with border or padding.
- #931 Fix baseline alignment in Safari 6.2/7.1/8.0.
- #937 Fix incorrect width in MathJax font data affecting underlining.
- #966 Fix SVG output overlapping when using prefix notation.
- #993 Add workaround for Native MathML in Gecko to re-enable `mlabeledtr` etc.
- #1002 Enable SVG output to inherit surrounding text color.

TeX emulation

- #881 Allow `\newenvironment` to process optional parameters.
- #889 remove extra space around some parenthesis constructs.
- #856 Recognize comma as decimal delimiter in units.
- #877 Fix bug related to multiple accent having different width.
- #832 Fix multline environment not being centered in HTML-CSS output.
- #776 Fix stretchy delimiters of `binom` and `choose`.
- #900 Fix `\buildrel` getting TeX class ORD instead of REL.
- #890 Enable px as dimension in `\\[...]`.
- #901 Allow `\limits` in more cases and add errors for some cases of multiple subscripts.
- #903 Allow `\hfill` to set alignment in matrices and arrays (for old fashioned TeX layout).
- #902 Convert `\eqalignno` and `\leqalignno` into `mlabeledtr`.
- #906 Allow comma separated parameters in `\mmlToken`.
- #913 Allow attributes in `\mmlToken` whose defaults are false or blank.
- #972 Fix autoload of the `color` extension.

- #375 Add `\{`, `\}`, and `\\` to macros working within `\text{}` etc.
- #969 Fix incorrect spacing with some `\frac` constructs.
- #982 Fix incorrect spacing in `aligned` environments.
- #1013 Fix processing error caused by `'` in commutative diagrams using `AMScd.js`.
- #1005 Add `wikipedia-texvc.js` extension.

Asciimath

- #851 Prevent leading space in quote from causing processing errors.
- #431 Fix handling of special characters in exponents.
- #741 Add underbrace macro.
- #857 Update AsciiMathML to 2.2; changes include improve entity handling, add triangle macro, map ast to asterisk, allow input of row vectors, allow lamda, switch phi/varphi mapping, add underbrace macro, handle empty nodes better, add vector norm macro, improve @ macro.

MathML Handling

- #847 Fix line-breaks in annotation elements.
- #805 Prevent empty annotation elements from causing math processing errors.
- #769 Update `indentshift` implementation to meet clarified MathML specification.
- #768 Fix processing of percentage values for `indenshift`.
- #839 Update inheritance of `displaystyle` in `mtable` to meet clarified MathML specification.
- #695 Allow Content MathML converion to be customized.
- #964 Move experimental support for elementary math and RTL to its own extension.

Fonts

- #845 Fix webfont bug in Safari 7.
- #950 Fix webfont bug in IE 11.

Localization

- #979 Updated locales thanks to Translatewiki.net; activate locales for Scots and Southern Balochi.

APIs

- #873 Combine array of elements when typesetting.
- #693 Add API to allow listeners to be cleared.

Misc.

- #870 Add Composer package information.
- #872 Add small delay between input and output phase to prevent performance degredation.
- #1016 Fix bug related to `<script>` elements with namespace prefix, e.g., in xHTML.

33.3.4 What's New in MathJax v2.4

MathJax v2.4 is primarily a bug fix release. Over 80 display bugs, line-breaking problems, and interface issues have been resolved; for a detailed listing please check the release milestone. The following are some of the highlights.

Security

- #256 Enable Content Security Policy compatibility.

Interface

- #240 prevent two identical uses of `\tag` to cause identical element id.
- #348 fix `Show Math as` window crashing in IE8.
- #559 remove user cookie configuration.
- #821 resolve cookie-related error in sandboxed iframes on Chrome.
- #623 fix localization on IE6–8.
- #685 fix MathMenu and MathZoom extensions loading when `showMathMenu` set to false.
- #734 compress menu PNGs.
- #814 add TeX/Asciimath as annotation-xml to MathML output.

Line-breaking

- #617 add linebreaking support for `mmultiscript` elements.
- #687 fix forced line breaking aligning badly.
- #707 fix ignored line breaks between two `mtext` elements.

HTML-CSS/SVG/nativeMML display

- #387 fix missing styling for `merror` in SVG output.
- #391 fix linebreaking within fractions in SVG output.
- #423, #460, #749, #824 Zoom improvements: fix zoom box overflow in mobile Safari, fix zoom box for widths in `px`, fix zoom box overlay in Chrome.
- #470 fix AMScd rendering in native MathML output.
- #473 override `text-ident` of enclosing paragraph.
- #476 improve big /Downarrows.
- #580 prevent CSS from overriding MathJax's em/ex detection.

- #619 fix: vertical stretching arrows in table cells can cause extra space between rows.
- #699 fix table column spacing in NativeMathML output on Firefox.
- #701 fix clipping of stretched delimiters in HTML-CSS output.
- #703 fix math axis not scaled in script sizes.
- #715 fix hat ^ too large with local STIX fonts in HTML-CSS.
- #744 improve root symbol rendering in ever-changing but always buggy Chrome.
- #770 add support for dotted borders to SVG output.
- #820 fix integral overlapping with superscript using STIX fonts.
- #813 remove some redundant fixes for Native MML on Firefox 29+.

TeX emulation

- #367 prevent `\mmltoken` from creating `annotation` elements.
- #377 improve ` ` handling.
- #389 fix operating spacing in `\split` and `\multiline` environments.
- #477, #459 add `\textsf` and `\texttt` macros and enable `mtextInheritFont` for them.
- #547 fix misalignment in nested fractions in HTML-CSS and SVG output.
- #624 fix AMScd on IE6–7.
- #632 fix `\Big` not accepting delimiters in braces
- #667 fix loop in `bbox`.
- #691 enable multiple `\label` in multiline environments like `align`, `eqnarray`, and `gather`.
- #719 empty array lines should get correct height.
- #739 fix `\operatorname*` and `\DeclareMathOperator*`.
- #746 fix spacing for `\left ... \right`.
- #793 allow unmatched groups in `\begin` \end‘ substitutions.
- #794 fix spacing for `\bmod`.

Asciimath

- #353 add option for TeX-like `\phi` and `\varphii` behavior.
- #743 add `mmlSpacing` option and set to true.
- #747 fix processing error with invisible grouping.

MathML Handling

- #328 remove `_moz-*`-attributes and improve MathML processing in Firefox.
- #460 fix default value of `mo@symmetric`.
- #478 make `mfenced` element equivalent to its expanded form
- #561 implement `menclose` notation `phaseorangle`.

- #578 fix quote attributes for `ms` elements.
- #614 handle nested `math` elements better.
- #684 fix handling of double primes in superscripts.
- #691, #692, update Content MathML extension: fix IE11, plus with leading negative number.
- #763 fix `mglyph` elements rendering too small.

Fonts

- #501 add workaround for broken Fedora STIX fonts configuration.
- #517 reset min/max width for MathJax font test.
- #576 improve font matching.
- #615 check validity of font names.
- #681 fix MathJax font test breaking responsive layout.
- #711 detect new webfonts when locally installed.
- #697 fix bold-italic for new webfonts.

Localization

- #753 update locales from translatewiki.net; add Vietnamese, Asturia, Polish, Catalan, Czech, Kannada locales.
- #777 fix menu orientation for RTL languages.

Misc.

- #586 add all input processors to `default.js`.
- #658 fix IE 11 recognized as Firefox.
- #730 ignore rendering targets that have been removed from document.
- #735 work around webfont bug in Chrome 32+.
- #738 improve workaround for fixed position bug in old IE versions.
- #737 add third-party path variable (for centralized custom extension hosting).

33.3.5 What's New in MathJax v2.3

MathJax v2.3 includes a number of new features, as well a more than 30 important bug fixes.

Features:

- *New webfonts:* MathJax v2.3 adds new webfonts for `STIX`, `Asana Math`, `Neo Euler`, `Gyre Pagella`, `Gyre Termes`, and `Latin Modern`.
- *Localization improvements:* MathJax has been accepted into TranslateWiki.net. Thanks to the TWN community we could add 12 complete and over 20 partial translations.

- *MathML improvements:* MathJax's "Show Math as" menu will now expose the MathML annotation features. There are also two new preview options for the MathML input mode: `mathml` (now the default), which uses the original MathML as a preview, and `altimage`, which uses the `<math>` element's `altimg` (if any) for the preview.
- *Miscellaneous improvements:* A new extension `MatchWebFonts` improves the interaction with the surrounding content when that uses a webfont. A new configuration method allows configurations to be specified using a regular JavaScript variable `window.MathJax`.
- MathJax is now available as a Bower package thanks to community contributions.

TeX input:

- Prevent the TeX pre-processor from rendering TeX in MathML annotation-xml elements. (Issue #484)
- Fix sizing issue in `cases` environment (Issue #485)

Fonts:

- Fix block-letter capital I (U+2111) appearing as J in MathJax font (Issue #555)

MathML:

- Improved workarounds for MathML output on WebKit (Issue #482)
- Handle empty `multiscript`, `mlabeledtr`, and other nodes in Native MathML output (Issue #486)
- Replace non-standard `MJX-arrow` class by new `menclose` notation (Issue #481)
- Fix incorrect widths in Firefox MathML output (Issue #558)
- Fix display math not being centered in XHTML (Issue #650)
- Fix problem when LaTeX code appears in `annotation` node (Issue #484)

HTML-CSS/SVG output

- Fix MathJax not rendering in Chrome when sessionStorage is disabled (Issue #584)
- Fix `\mathchoice` error with linebreaking in SVG output (Issue #604)
- Fix poor linebreaking of "flat" MathML with unmatched parentheses (Issue #523)

Interface:

- Fix Double-Click zoom trigger (Issue #590)

Miscellaneous:

- Localization: improved fallbacks for IETF tags (Issue #492)
- Localization: support RTL in messages (Issue #627)
- Improve PNG compression (Issue #44)

33.3.6 What's New in MathJax v2.2

MathJax v2.2 includes a number of new features, as well a more than 40 important bug fixes.

Features:

- Localization of MathJax user interface. (German and French translations currently available in addition to English.)
- Commutative diagrams via the `AMScd` extension.
- New Safe-mode extension that allows you to restrict potentially dangerous features of MathJax when it is used in a shared environment (e.g., href to javascript, styles and classes, etc.)
- Improve MathML rendering for `mfenced` and `mlabeldtr` elements in browsers that don't support them well.
- Experimental Content MathML support.

TeX input:

- Avoid potential infinite loops in `\mathchoice` constructs. (Issue #373)
- Add error message when an evironment closes with unbalanced braces. (Issue #454)
- Allow spaces in the RGB, rgb, and greyscale color specifications. (Issue #446)
- Process `\$` in `\text` arguements. (ssue #349)
- Preserve spaces within `\verb` arguments. (Issue #381)
- Make `\smallfrown` and `\smallsmile` come from the variant font so they have the correct size. (Issue #436)
- Make the input TeX jax generate mrow plus mo elements rather than mfenced elements (for better compatibility with native MathML implementations).
- Make `\big` and its relatives use script or scriptscript fonts (although size is still absolute, as it is in TeX) so that it balances the text weight in scripts. (Issue #350)
- Convert true and false attributes to booleans in `\mmlToken`. (Issue #451)

AsciiMath:

- Rename AsciiMath config option from `decimal` to `decimalsign`. (Issue #384)

Fonts:

- Add Greek Delta to SVG fonts. (Issue #347)
- Fix monospace space character to be the same width as the other monospace characters. (Issue #380)
- Better handling of unknown or invalid values for mathvariant or values not supported by generic fonts.

MathML:

- Handle empty child nodes better.
- Improved MathML rendering for `mfenced` and `mlabeldtr` elements.
- Ignore `linebreak` attribute on `mspace` when dimensional attributes are set. (Issue #388)
- Implement `rowspacing`/`columnspacing` for `mtable` in native MathML output in Firefox using cell padding.

HTML-CSS/SVG output

- Allow `\color` to override link color in SVG output. (Issue #427)
- Add min-width to displayed equations with labels so that they cause their containers to have non-zero width (like when they are in a table cell or an absolutlye positioned element). (Issue #428)
- Fix a processing error with elements that contain hyperlinks. (Issue #364)
- Try to isolate MathJax from CSS transitions. (Issue #449)
- Go back to using em's (rounded to nearest pixel) for Chrome. Rounding makes the placement work more reliably, while still being in relative units. (Issue #443)
- Prevent error when math contains characters outside of the MathJax fonts. (Issue #441)
- Make final math size be in relative units so that it prints even if print media has a different font size. (Issue #386)
- Don't scale line thickness for `menclose` elements (so lines won't disapear in scripts). (Issue #414)
- Fix `fontdata.js` to allow it to be included in combined configuration files. (Issue #413)
- Makes math-based tooltips be spaced properly when rendered. (Issue #412)
- Fix Math Processing Error when `&ApplyFunction;` is used without preceeding content. (Issue #410)
- Fix a problem using an empty table as a super- or subscript. (Issue #392)
- Handle the case where selection in maction is invalid or out of range. (Issue #365)
- Add a pixel extra around the SVG output to accommodate antialiasing pixels. (Issue #383)
- Fix Math Processing Error for `msubsup`/`msub`/`msup` elements.
- Limit the number of repetition to build stretchy chars in HTML-CSS. (Issue #366)
- Fix Math Processing Error in `mmultiscripts`/`menclose`. (Issue 362)

Interface:

- Make zoom work properly with expressions that have full width (e.g., tagged equations).
- Handle zooming when it is inside a scrollable element when it is not the main body element. (Issue #435)
- Update math processing errors to include original format and actual error message in the "Show Math As" menu. (Issue #450)
- Add a Help dialog box (rather than link to mathjax.org).
- Remove the v1.0 configuration warning. (Issue #445)
- Trap errors while saving cookies (and go on silently). (Issue #374)
- Fix typo in IE warning message. (Issue #397)

- Use UA string sniffing for identifying Firefox and handle detecting mobile versions better.
- Make MathML source show non-BMP characters properly. (Issue #361)
- Make tool tips appear above zoom boxes. (Issue #351)

Miscellaneous:

- Allow preview for preprocessors to be just a plain string (rather than requiring `[string]`).
- Remap back-tick to back-quote. (Issue #402)
- Handle script tags in `HTML.Element()` so they work in IE. (Issue #342)
- Add the `MathJax_Preview` class to the `ignoreClass` list so that `tex2jax` and `asciimath2jax` won't process previews accidentally. (Issue #378)
- Fix processing errors with various table and menclose attributes. (Issue #367)
- Use `hasOwnProperty()` when checking file specification objects (prevents problems when `Object.prototype` has been modified). (Issue #352)

33.3.7 What's New in MathJax v2.1

MathJax v2.1 is primarily a bug-fix release. Numerous display bugs, line-breaking problems, and interface issues have been resolved. The following lists indicate the majority of the bugs that have been fixed for this release.

Interface

- Make NativeMML output properly handle iOS double-tap-and-hold, and issue warning message when switching to NativeMML output.
- Use `scrollIntoView` to handle `positionToHash` rather than setting the document location to prevent pages from refreshing after MathJax finishes processing the math.
- Handle positioning to a hash URL when the link is to an element within SVG output.
- Make `href`'s work in SVG mode in all browsers.
- Fix problem with opening the "Show Math As" window in WebKit (affected Chrome 18, and Safari 5.1.7).
- Use MathJax message area rather than window status line for `maction` with `actiontype='statusline'` to avoid security restrictions in some browsers.
- Fix issue where zoom box for math that has been wrapped to the beginning of a line would be positioned at the end of the previous line.
- Fix a problem where IE would try to typset the page before it was completely available, causing it to not typeset all the math on the page (or in some cases *any* of the math).
- Allow decimal scale values in the dialog for setting the scale.
- Fix SVG output so that setting the scale will rescale the existing mathematics.
- Add close button to About box and don't make clicking box close it (only clicking button).
- Make About box show 'woff or otf' when otf fonts are used (since both are requested).
- Have output jax properly skip math when the input jax has had an internal failure and so didn't produce any element jax.
- Produce `MathJax.Hub` signal when `[Math Processing Error]` is generated.

Line-breaking

- Fix problem with SVG output disappearing during line breaks when equation numbers are also present.
- Fix problem with potential infinite loop when an `<mspace>` is an embellished operator that causes a linebreak to occur.
- Allow line breaks within the base of `<msubsup>` to work so that the super and subscripts stay with the last line of the base.
- Fix `<mfenced>` so that when it contains a line break the delimiters and separators are not lost.
- Allow line breaks at delimiters and separators in **<mfenced>** elements.
- Fix issue with line breaking where some lines were going over the maximum width.
- Fix problem with line breaking inside `<semantics>` elements.
- Fix problem with line breaking where the incorrect width was being used to determine breakpoint penalties, so some long lines were not being broken.

HTML-CSS/SVG display

- Fix several Chrome alignment and sizing issues, including problems with horizontal lines at the tops of roots, fraction bars being too long, etc.
- Resolve a problem with how much space is reserved for math equations when a minimum font size is set in the browser.
- Force final math span to be remeasured so that we are sure the container is the right size.
- Fix alignment problem in `<msubsup>`.
- Fix processing error when rowalign has a bad value.
- Fix a vertical placement problem with stretched elements in mtables in HTML-CSS, and improve performace for placeing the extension characters.
- Handle spacing for U+2061 (function apply) better.
- Better handling of primes and other pseudo scripts in HTML-CSS and SVG output.
- Fixed a problem with `<mmultiscripts>` in SVG mode that caused processing error messages.
- Fix misplaced `\vec` arrows in Opera and IE.
- Make `<mi>` with more than one letter have `texClass` OP rather than ORD in certain cases so it will space as a function.
- Make HTML snippet handler accept a string as contents, even if not enclosed in braces.
- Fix spacing for functions that have powers (e.g., `\sin^2 x`).
- Fix problem with SVG handling of `\liminf` and `\limsup` where the second half of the function name was dropped.
- Fixed a problem where HTML-CSS and SVG output could leave partial equations in the DOM when the equation processing was interrupted to load a file.
- Fix problems with `<mtable>`, `<ms>`, and `<mmultiscripts>` which weren't handling styles.
- Make column widths and row heights take minsize into account in `<mtable>`.
- Fix typo in `handle-floats.js` that caused it to not compile.
- Fix problem in HTML-CSS output with `<msubsup>` when super- or subscript has explicit style.

TeX emulation

- Allow negative dimensions for `\\[]` but clip to 0 since this isn't really allowed in MathML.
- Fixed problem where \ with whitespace followed by [would incorrectly be interpretted as \[dimen].
- Make `jsMath2jax` run before other preprocessors so that `tex2jax` won't grab environments from inside the jsMath spans and divs before jsMath2jax sees them.
- Fix issue with `\vec` not producing the correct character for `\vec{\mathbf{B}}` and similar constructs.
- Combine multiple primes into single unicode characters.
- Updated the unicode characters used for some accents and a few other characters to more appropriate choices. See issues #116, #119, and #216 in the MathJax issue tracker on GitHub.
- Remove unwanted 'em' from `eqnarray columnwidth` values.
- Make eqnarray do equation numbering when numbering is enabled.
- Make vertical stretchy characters stand on the baseline, and improve spacing of some stretchy chars.
- Make `mtextFontInherit` use the style and weight indicated in the math, so that `\textbf` and `\textit` will work properly.
- Add `\textcolor` macro to the color extension.
- Added RGB color model to the color extension.
- Automatically load the AMSmath extension when needed by the `mhchem` extension.
- Add <<=> arrow to `mhchecm` extension
- Fix alignment of prescripts in `mhchem` to properly right-justify the scripts.
- Expose the CE object in the `mhchem` extension.
- Make `autoload-all` skip extensions that are already loaded, and not redefine user-defined macros.
- Fix most extensions to not overwrite user defined macros when the extension is loaded.
- Ignore `\label{}` with no label.
- Make `\injlim` and friends produce single `<mi>` elements for thier names rather than one for each letter.
- Handle primes followed by superscript as real TeX does in TeX input jax.
- Handle a few more negations (e.g., of arrows) to produce the proper Unicode points for these.
- Don't produce a processing error when `\limits` is used without a preceding operator.

MathML Handling

- Prevent align attribute on `<mtable>` from applying to `<mover>/<munder>/<munderover>` elements.
- Ignore `_moz-math-*` attributes in MathML input so they don't appear in MathML output.
- Prevent duplicate `xmlns` attributes in "Show Math As -> MathML".
- Fixed a problem in MathML output where dimensions given to `<mpadded>` with leading +'s could lose the plus and become absolute rather than relative.
- Fix `setTeXclass` for `TeXatom` so that it handles the spacing for relations correctly.
- Add more CSS to isolate `NativeMML` output from page.

- Handle setup of MathPlayer better for IE10, and avoid some IE10 bugs in setting the document namespace for MathML.

Fonts

- Fix a problem where bold-script didn't work properly in STIX fonts.
- Work around Chrome bug with MathJax web fonts that affects some combining characters.
- Remove dependencies of TeX->MathML conversion on the choice of fonts (TeX versus STIX).
- For stretchy characters that don't have a single-character version in the MathJax fonts, make sure they are properly sized when not stretched or stretched to a small size.
- Fix an error with `U+u005E` (^) which caused it to show as a plus when used as a stretchy accent.
- Fix a problem with greek letters in STIX font producing the wrong letter (an offset was off by one).
- Handle more characters in sans-serif-italic and bold-italic STIX fonts.

33.3.8 What's New in MathJax v2.0

MathJax version 2.0 includes many new and improved features, including much better speeds in Internet Explorer, a new AsciiMath input processor, a new *SVG* output processor, support for additional LaTeX commands, and many bug fixes, to name just a few of the changes.

Major speed improvement for HTML-CSS output, particularly in IE

The HTML-CSS output processing was redesigned to avoid the page reflows that were the main source of the speed problem in Internet Explorer 8 and 9. For test pages having between 20 and 50 typeset expressions, we see an 80% reduction in output processing time for IE8, a 50% reduction for IE9, and between 15% and 25% reduction for most other browsers over the corresponding v1.1a times. Since the processing time in v1.1a grows non-linearly in IE, you should see even larger savings for pages with more equations when using v2.0. Forcing IE7 emulation mode is no longer necessary (and indeed is no longer recommended).

Reduced flickering during typsetting

In the past, each expression was displayed as soon as it was typeset, which caused a lot of visual flickering as MathJax processed the page. In v2.0, the output is processed in blocks so that typeset expressions are revealed in groups. This reduces the visual distraction, and also speeds up the processing. The number of equations in a block can be controlled through the `EqnChunk` parameter in the HTML-CSS or SVG block of your configuration. See the *configuration options for HTML-CSS* and *configuration options for SVG* pages for details.

If the page URL includes a hash reference (a link to a particular location within the page), MathJax v2.0 will jump to that location after the page has finished typsetting. (Since the size of the page may have changed due to the mathematical typsetting, that location may no longer be visible on screen, so MathJax moves there when it is done with the initial typesetting.) You can control this behavior with the `positionToHash` parameter in the main section of your configuration. See the *core configuration options* page for details.

Automatic equation numbering of TeX formulas

The TeX input jax now can be configured to add equation numbers (though the default is not to number equations so that existing pages will not change their appearance). This is controlled through the `equationNumbers` section of the `TeX` block of your configuration (see the *equation numbering* section for details). You can request that the

numbering follow the AMS-style numbering of environments, or you can request that every displayed equation be numbered. There are now `\label`, `\ref`, and `\eqref` commands to make it easier to link to particular equations within the document.

Automatic line breaking of long displayed equations

MathJax now implements the MathML3 specification for automatic line breaking of displayed equations in its HTML-CSS output. This is disabled by default, but can be enabled via the `linebreaks` section of the `HTML-CSS` or `SVG` block of your configuration (see the *automatic line breaking* section for details). Note that automatic line breaking only applies to displayed equations, not in-line equations, unless they are themselves longer than a line. The algorithm uses the nesting depth, the type of operator, the size of spaces, and other factors to decide on the breakpoints, but it does not know the meaning of the mathematics, and may not choose the optimal breakpoints. We will continue to work on the algorithm as we gain information from its actual use in the field.

New AsciiMath input jax and SVG output jax

MathJax currently processes math in either *TeX* and *LaTeX* format, or *MathML* notation; version 2.0 augments that to include *AsciiMath* notation (see the ASCIIMathML home page for details on this format). This is a notation that is easier for students to use than TeX, and has been requested by the user community. See the *AsciiMath support* page for details.

In addition to the HTML-CSS and Native MathML output available in v1.1, MathJax v2.0 includes an *SVG*-based output jax. This should prove to be more reliable than the HTML-CSS output, as it avoids some CSS, web-font, and printing issues that the HTML-CSS output suffers from, and it currently has no browser-dependent code. The SVG mode even works in some ebook readers (like Apple iBooks and Calibre). See the *output formats* documentation for details.

New combined configuration files

Pre-defined configuration files that include the AsciiMath and SVG processors are now available with MathJax v2.0. These include `AM_HTMLorMML`, `TeX-AMS-MML_SVG`, and `TeX-MML-AM_HTMLorMML`. See the *common configurations* section for details.

MathJax contextual menu now available on mobile devices

MathJax v2.0 provides access to its contextual menu in mobile devices that are based on the WebKit (Safari) and Gecko (Firefox) engines. For Mobile Firefox, the menu is accessed by a tap-and-hold on any expression rendered by MathJax (this is Mobile Firefox's standard method of triggering a contextual menu). In Mobile Safari, use a double-tap-and-hold (you may need to zoom in a bit to be able to accomplish this). This is the first step toward providing a better interface for mobile devices.

Improved support for screen readers

Some issues surrounding the use of screen readers and their interaction with MathPlayer have been resolved in MathJax v2.0. In particular, there are additional menu items that allow the user finer control over some aspects of MathJax's interface that were interfering with some screen readers' ability to properly identify the mathematics. Several stability issues with MathPlayer have also been addressed. In Internet Explorer when MathPlayer is installed, there is now a new contextual menu item to allow you to specify what events are handled by MathJax and what should be handled by MathPlayer. This gives you finer control over MathPlayer's interaction with some screen readers.

Many new TeX additions and enhancements

- New *mhchem* chemistry extension (adds `\ce`, `\cf`, and `\cee` macros)
- New *cancel* extension (adds `\cancel`, `\bcancel`, `\xcancel`, and `\cancelto` macros)
- New *extpfeil* extension (adds more stretchy arrows)
- New *color* extension (makes `\color` work as a switch, as in LaTeX). Adds `\definecolor`, other color models, LaTeX named colors, `\colorbox`, `\fcolorbox`, etc.
- New *begingroup* extension to allow macro definitions to be localized. Adds `\begingroup` and `\endgroup` for isolating macro declarations, and defines `\let`, `\renewenvironment`, `\global`, and `\gdef`.
- New *enclose* extension to give TeX access to `<menclose>` elements. Adds `\enclose{type}[attributes]{math}` macro.
- New *action* extension to give TeX access to `<maction>` elements. Adds `\mathtip{math}{tip}`, `\texttip{math}{tip}`, and `\toggle{math1}{math2}...\endtoggle` macros.
- New `\mmToken{type}[attributes]{text}` macro for producing `<mo>`, `<mi>`, `<mtext>`, and other token MathML elements directly.
- New `\bbox[color;attributes]{math}` macro to add background color, padding, borders, etc.
- New `\middle` macro for stretchy delimiters between `\left` and `\right`.
- New `\label`, `\ref`, and `\eqref` macros for numbered equations.
- Better implementation of `\not` so it produces proper MathML when possible.
- Better implementation of `\dots` that selects `\ldots` or `\cdots` depending on the context.
- Better implementation of `\cases` that automatically uses `\text` on the second entry in each row.
- Safer implementation of `\require` that only allows loading from extensions directory.
- Allow `\newcommand` to provide a default parameter.
- Allow `\\` to take an optional argument that specifies additional space between lines.
- Allow `\\` to be used anywhere (to force a line break), not just in arrays.
- Allow optional alignment parameter for array, aligned, and gathered environments.

See the *TeX support* page for details on these extensions and macros.

Font enhancements

- Work around for the OS X Lion STIX font problem.
- Support for STIX-1.1 fonts (detection of which version you have, and use data appropriate for that).
- New WOFF versions of the web fonts (smaller, so faster to download).
- Data for more stretchy characters in HTML-CSS output.
- Add support for Unicode planes 1 through 10 (not just the Math Alphabet block) in HTML-CSS output.
- Increased timeout for web fonts (since it was switching to image fonts too often, especially for mobile devices).
- Only switch to image fonts if the first web font fails to load (if we can access one, assume we can access them all).
- Allow `<mtext>` elements to use the page font rather than MathJax fonts (optionally). This is controlled by the `mtextFontInerhit` configuration parameter for HTML-CSS and SVG output jax.

- Provide better control over the font used for characters that are not in the MathJax fonts.
- Allow Firefox to use web-based fonts when a local URL uses MathJax from the CDN (in the past it would force image fonts when that was not necessary).

Interface improvements

- The MathJax contextual menu has been reorganized to make it easier to get the source view, and to control the parameters for MathPlayer in IE.
- The MathJax contextual menu is available in mobile devices (see description above).
- Warning messages are issued if you switch renderers to one that is inappropriate for your browser.
- MathJax now starts processing the page on the `DOMContentLoaded` event rather than the page `onload` event (this allows the mathematics to appear sooner).
- Native MathML output is now scaled to better match the surrounding font (like it is for HTML-CSS output).
- Better CSS styling for NativeMML output in Firefox in order to handle `\cal` and other fonts.
- MathML output now (optionally) includes class names to help mark special situations generated by the TeX input jax. (This lets the MathML from the Show Source menu item better reproduce the original TeX output.)
- MathJax now loads the menu and zoom code (if they haven't been loaded already) after the initial typesetting has occured so that they will be available immediately when a user needs those features, but do not delay the initial typesetting of the mathematics.
- For the *tex2jax* preprocessor, the `processClass` can now be used to override the `skipTags` to force a tag that is usually skipped to have its contents be processed.
- The *noErrors* and *noUndefined* extensions can now be disabled via a configuration option (since they are included in many of the combined configuration files). See the *noErrors* and *noUndefined* sections of the *TeX support* page for more information.
- There is a new `MathJax.Hub.setRenderer()` function that can be used to switch the current renderer. See the *MathJax Hub API* documentation for details.
- A user-defined macros is no longer overridden if an extension is loaded that redefines that macro.
- Improved web-font detection reliability.

Important changes from previous versions

- The default renderer for Firefox has been changed from *NativeMML* to *HTML-CSS* (in those configurations that choose between the two). The only browser that defaults to *NativeMML* is now IE with MathPlayer installed. You can configure this to your liking using the *MMLorHTML configuration options*.
- *NativeMML* output will now be selected in IE9 when MathPlayer is present (since IE9 was released the same day as MathJax v1.1a, and there had been problems with IE9 beta releases, we weren't sure if MathPlayer would work with the official release, and so did not select NativeMML by default.)
- The performance improvements in IE8 and IE9 now make it unnecessary to use a `<meta>` tag to force IE7 emulation mode. In fact IE9 in IE9 standards mode now runs faster than IE9 in IE7 standards mode, and IE8 in IE8 standards mode is comparable to IE8 in IE7 standards mode. We now recommend that you use

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

  to obtain the highest emulation mode available in IE, which will be the fastest one for MathJax 2.0.

- The *tex2jax* preprocessor now balances braces when looking for the closing math delimiter. That allows expressions like

```
$y = x^2 \hbox{ when $x > 2$}$
```

  to be properly parsed as a single math expression rather than two separate ones with unbalanced braces. The old behavior can be obtained by setting `balanceBraces` to false in the `tex2jax` block of your configuration. (See the *tex2jax configuration options* for details.)

- If you are hosting your own copy of MathJax on your server, and that copy is being used from pages in a different domain, you will have set up the access control paramters for the font directory to allow Firefox to access the font files properly. Since MathJax 2.0 includes fonts in WOFF format, you will need to include `woff` in you access control declaration for the fonts. E.g., use

```
<FilesMatch "\.(ttf|otf|eot|woff)$">
<IfModule mod_headers.c>
Header set Access-Control-Allow-Origin "*"
</IfModule>
</FilesMatch>
```

  in the `.htaccess` file for the ``Mathjax/fonts` directory if you are using the Apache web server. See *Notes about shared installations* for details.

- The `\cases` macro now properly places the second column in text mode not math mode. In the past, one needed to use `\text` in the second column to achieve the proper results; pages that did this will still work properly in v2.0. Pages that took advantage of the math mode in the second column will need to be adjusted.

- The `\dots` macro now produces `\ldots` or `\cdots` depending on the context (in the past, `\dots` always produced `\ldots`).

- A one pixel padding has been added above and below HTML-CSS and SVG output so that math on successive lines of a paragraph won't bump into each other.

- There is a new *MathPlayer* submenu of the *Math Settings* menu in the MathJax contextual menu that allows the user to control what events are passed on to MathPlayer. This allows better control for those using assistive devices like screen readers. When menu events are being passed on to MathPlayer, the MathJax menu can be obtained by ALT-clicking on a typeset expression (so the user can still access MathJax's other features).

- In order to improve stability with IE when MathPlayer is installed, MathJax now adds the namespace and object bindings that are needed for MathPlayer at the time that Mathjax is first loaded, rather than waiting for the *NativeMML* output jax to be loaded. Since this is before the configuration information has been obtained, this will happen regardless of whether the *NativeMML* output jax is requested. This means that IE may ask the user to allow MathPlayer to be used, and may show the MathPlayer splash dialog even when MathPlayer is not in the end used by MathJax. Note that this setup can only be performed if MathJax is loaded explicitly as part of the initial web page; if it is injected into the page later by adding a `<script>` tag to the page dynamically, then MathPlayer will be set up when the *NativeMML* jax is loaded as in the past, and some stability issues may occur if events are passed to MathPlayer.

- The MathJax typesetting is now started on `DOMContentLoaded` rather than at the page `onload` event, when possible, so that means MathJax may start typesetting the page earlier than in the past. This should speed up typesetting one pages with lots of images or side-bar content, for example.

- MathJax now attempts to determine whether the page's `onload` event had already occurred, and if it has, it does not try to wait for the `DOMContentLoaded` or `onload` event before doing its initial typeset pass. This means that it is no longer necessary to call `MathJax.Hub.Startup.onload()` by hand if you insert MathJax into the page dynamically (e.g., from a GreaseMonkey script).

- If the page URL includes a hash reference (a link to a particular location within the page), MathJax v2.0 will jump to that location after the page has finished typsetting. Since the size of the page may have changed due to

the mathematical typsetting, that location may no longer be visible on screen, so MathJax moves there when it is done with the initial typesetting. You can control this behavior with the `positionToHash` parameter in the main section of your configuration (see *core configuration options*).

- In the event that MathJax is not able to load the configuration file you have specified in the script tag that loads `MathJax.js` via `config=filename`, it will no longer issue the warning message about a missing configuration. The configuration process changed in v1.1, and that message was to help page maintainers update their configurations, but it turns out that for users with slow network connections, MathJax could time out waiting for the configuration file and would issue the warning message in that case, even though the page included the proper configuration. That should no longer occur in MathJax v2.0.

Other enhancements

- Use prioritized lists of callbacks for StartupHooks, MessageHooks, LoadHooks, PreProcessors, and pre- and post-filters on the input jax.
- Updated operator dictionary to correspond to current W3C version.
- Improved browser detection for Gecko and WebKit browsers.
- Make prefilters and postfilters for all input jax, and make them into hook lists rather than a single hook.
- Use `<mi>` rather than `<mo>` for `\sin`, `\cos`, and other such functions, for `\mathop{\rm...}` and `\operatorname`.
- Add `&ApplyFunction;` after `\mathop{}` and other macros that are functions (e.g., `\sin`).
- The `MathJax_Preview` style has been moved from `HTML-CSS/jax.js` to `MathJax.js`, since it is common to all output.
- The *autobold* extension now uses `\boldsymbol` rather than `\bf` so that it will affect more characters.
- Make units of `mu`'s be relative to the scriptlevel (as they are supposed to be).
- Reorganized the event-handling code to make it more modular and reduce redundancy in the different output jax.
- Modified CSS in *NativeMML* output for Firefox to use local copies of the web fonts, if they are available.
- Error messages now have the MathJax contextual menu.
- Better handling of some characters not in the web fonts (remap to locations where they exist, when possible).
- Better choice of accent characters in some cases.
- Better handling of pseudo-scripts (like primes).
- Better sizing of characters introduced by `\unicode{}`, or otherwise outside of the fonts known to MathJax.
- Provide a new extension to handle tagged equations better in *HTML-CSS* output when there are floating elements that might reduce the area available to displayed equations. (See the HTML-CSS extensions section of the *output formats* documentation for detais.)
- Use a text font for `\it` rather than the math italics, so spacing is better.
- Handle italic correction better in *HTML-CSS* output
- Handle `href` attributes better, especially when on `<math>` elements.
- Allow `\sqrt\frac{}{}` without producing an error.

Other bug fixes

- MathPlayer setup changed to prevent crashes.
- Moved remapping of `<mo>` contents to the output jax so that the original contents aren't changed.
- Don't combine `mathvariant` with `fontstyle` or `fontweight` (as per the MathML specification).
- Isolate non-standard attributes on MathML elements so that they don't interfere with the inner workings of MathJax.
- Properly handle width of border and padding in merrors in *HTML-CSS* output.
- Properly handle lower-case Greek better.
- Process weight and style of unknown characters properly.
- Fixed spacing problems with `\cong` in MathJax web fonts .
- Choose better sizes for `\widehat` and `\widetilde`
- Fixed problem with detecting em/ex sizes when uses in mobile devices with small screen widths.
- Fixed MathML output when dimensions of `mu`'s are used in TeX input.
- Better handling of table borders from TeX.
- Fixed some problems with table widths and heights, and spacing.
- Better handling of colored backgrounds in *HTML-CSS* output.
- Handle border and padding CSS styles better in *HTML-CSS* output.
- Fixed multline environment to put tags on bottom row when `TagSide` is set to `right`.
- Force reflow after equations are typeset so that some rendering problems in tables are corrected in Firefox and WebKit browsers.
- Fixed a number of bugs with the size of zoom boxes and the size of their content.
- Have equations with tags zoom into a full-width zoom box to accommodate the tag.
- Fixed positioning problem with zoom boxes in NativeMML mode.
- Don't allow mouse events on zoomed math.
- Fixed `MathJax.Hub.getJaxFor()` and `MathJax.Hub.isJax()` to properly handle elements that are part of an output jax's output (in particular, you can find the element jax from any DOM element in the output).
- Fixed a number of font anomalies (problems in the data files).
- Fixed problem where `<mspace>` with a background color would not always overlay previous items.
- Fixed a problem with colored `<mspace>` elements being too tall in IE/quirks mode.
- Fixed problem where `<mtable>` with `equalrows="true"` would not produce equal height rows.
- Allow `<mpadded>` background color to be specified exactly (i.e., without the 1px padding) when one of its dimensions is given explicitly (or there is no content).
- Avoiding flicker problem with hover zoom trigger in Firefox.
- Fix `\unicode` bug with font names that include spaces.
- Remove internal multiple spaces in token elements as per the MathML specification.
- Work around HTML5 removing namespaces, so that `xmlns:xlink` becomes `xlink` with no namespace, which confuses the XML parsers.

- Fix `MathJax.Message.Set()` and `MathJax.Message.Clear()` so that a delay of 0 is properly handled.
- Produce better MathML for `\bmod`, `\mod`, and `\pmod`.
- Don't allow Safari/Windows to use STIX fonts since it can't access characters in Plane1 (the mathematical alphabets).
- Fix `\thickapprox` to use the correct glyph in *HTML-CSS* output with MathJax web fonts.
- Make style attributes work on `<mstyle>` elements.
- Better handling of border and padding on MathML elements in *HTML-CSS* output.
- Fixed error with size of `\:` space.
- Allow delimiter of `.` on `\genfrac` (it was accidentally rejected).
- Handle AMSmath control sequences with stars better (`\cs{*}` no longer counts as `\cs*`).
- Fixed wrong character number in stretchy data for *U+221A*.
- Fixed `<annotation-xml>` to use the proper scaling in *HTML-CSS* output.
- Fixed a problem with combining characters when they are used as accents.
- Fixed a problem in Firefox with `\mathchoice` when the contents have negative width.
- TeX input jax no longer incorrectly combines `<mo>` elements that have different variants, styles, classes, or id's.
- Fixed the `scriptlevel` when `<munderover>` has base with `movablelimits="true"` in non-display mode.
- Fixed typo in implementation of `SimpleSUPER`.
- Fixed typo in self-closing flag for `<mprescript>` tag.
- Prevent infinite loop if one of the jax fails to load (due to failure to compile or timeout waiting for it to load).
- Fixed a whitespace issue in token elements with IE/quirks mode in the *MathML* input jax.
- Make sure height is above depth when making spaces and rules in *HTML-CSS* and *SVG* output.
- Fixed *HTML-CSS* tooltip to be work properly when a restart occurs within the tooltip.
- Fixed problem with size of colored backgrounds on `<mo>` in some circumstances in *HTML-CSS* output.
- Make `\ulcorner`, etc. use more approprate unicode positions, and remap those positions to the locations in the MathJax web fonts.

Some technical changes

- Break the processing phase into two separate phases to do input processing separately from output processing (they used to be interleaved). This makes it easier to implement forward references for the `\ref` macro.
- Make `Font Preference` menu honor the `imageFont` setting.
- Changed the name of the preview filter commands to `previewFilter` in all preprocessors.
- Make ^ and _ be stretchy even though that isn't in the W3C dictionary.
- Fixed *HTML-CSS* output problem when a multi-character token element has characters taken from multiple fonts.
- Force message text to be black in FontWarnings and configuration warnings.
- Added `Find()` and `IndexOf()` commands to menus to locate menu items.

- Added menu signals for post/unpost and activation of menu items.
- Added signals for typesetting of unknown characters.
- Added signals for zoom/unzoom.
- Added More signals for error conditions.
- Allow preferences to select MathML output for Safari with late enough version.
- Improved *About MathJax* box.
- Have *tex2jax* handle empty delimiter arrays and don't scan page if there is nothing to look for.
- Make delay following a *processing* message configurable and lengthen it to make browser more responsive during typesetting.
- Make thin rules be in pixels to try to improve results in IE (disappearing division lines).
- Mark all output elements as `isMathJax`, so it can be used to identify what elements are part of mathematical output.
- Force MathZoom and MathMenu to wait for the `Begin Styles` message before inserting their styles so when they are included in the combined files, the author can still configure them.
- Add default id's to the jax base object classes.
- Mark top-level math element as having a `texError` when it is one (to make it easier to recognize).
- Have `Update()` method ask ElementJax to determine if it needs updating (which in turn asks the associated input jax).
- Make `Remove()` work for just clearing output (without detaching) if desired.
- Have ElementJax store input and output jax ID's rather than pointers (to help avoid circular references for cleanup purposes).
- Move input/output jax and preprocessor registries from `Hub.config` to Hub itself (they are not user configurable through `Hub.Config`, and so even though they are configurations, they don't belong there).
- Make sure embelished large ops are type `OP` not `ORD` to get spacing right.
- Added `MathJax.HTML.getScript()` to get the contents of a script (needed since it works differently in different browsers).
- Move code that prevents numbers from being treated as a unit for super- and subscripts to the super- and subscript routine in the *TeX* input jax (prevents making changes to `\text{}`, `\hbox{}`, `\href{}`, etc.).
- Make *mml2jax* work better with IE namespaces (IE9 no longer seems to list the `xmlns` entries on the `<html>` element).

33.3.9 What's New in MathJax v1.1

MathJax version 1.1 includes a number of important improvements and enhancements over version 1.0. We have worked hard to fix bugs, improve support for browsers and mobile devices, process TeX and MathML better, and increase MathJax's performance.

In addition to these changes, MathJax.org now offers MathJax as a network service. Instead of having to install MathJax on your own server, you can link to our content delivery network (CDN) to get fast access to up-to-date and past versions of MathJax. See *Loading MathJax from the CDN* for more details.

The following sections outline the changes in v1.1:

Optimization

- Combined configuration files that load all the needed files in one piece rather than loading them individually. This simplifies configuration and speeds up typesetting of the mathematics on the page.
- Improved responsiveness to mouse events during typesetting.
- Parallel downloading of files needed by MathJax, for faster startup times.
- Shorter timeout for web fonts, so if they can't be downloaded, you don't have to wait so long.
- Rollover to image fonts if a web font fails to load (so you don't have to wait for *every* font to fail.
- The MathJax files are now packed only with *yuicompressor* rather than a custom compressor. The CDN serves gzipped versions, which end up being smaller than the gzipped custom-packed files.
- Improved rendering speed in IE by removing `position:relative` from the style for mathematics.
- Improved rendering speed for most browsers by isolating the mathematics from the page during typesetting (avoids full page reflows).

Enhancements

- Allow the input and output jax configuration blocks to specify extensions to be loaded when the jax is loaded (this avoids needing to load them up front, so they don't have to be loaded on pages that don't include mathematics, for example).
- Better handling of background color from style attributes.
- Ability to pass configuration parameters via script URL.
- Support HTML5 compliant configuration syntax.
- Switch the Git repository from storing the fonts in *fonts.zip* to storing the *fonts/* directory directly.
- Improved About box.
- Added a minimum scaling factor (so math won't get too small).

TeX Support

- Added support for `\href`, `\style`, `\class`, `\cssId`.
- Avoid recursive macro definitions and other resource consumption possibilities.
- Fix for `\underline` bug.
- Fix for bug with `\fbox`.
- Fix height problem with `\raise` and `\lower`.
- Fix problem with `\over` used inside array entries.
- Fix problem with nesting of math delimiters inside text-mode material.
- Fix single digit super- and subscripts followed by punctuation.
- Make sure *movablelimits* is off for `\underline` and related macros.
- Fix problem with dimensions given with `pc` units.

MathML Support

- Fix `<` and `&` being translated too early.
- Handle self-closing tags in HTML files better.
- Combine adjacent relational operators in `<mo>` tags.
- Fix entity name problems.
- Better support for MathML namespaces.
- Properly handle comments within MathML in IE.
- Properly consider `<mspace>` and `<mtext>` as space-like.
- Improved support for `<maction>` with embellished operators.

Other Bug Fixes

- Fixed CSS bleed through with zoom and other situations.
- Fixed problems with `showMathMenuMSIE` when set to `false`.
- Replaced illegal prefix characters in cookie name.
- Improved placement of surd for square roots and n-th roots.
- Fixed layer obscuring math from MathPlayer for screen readers.
- Newlines in CDATA comments are now handled properly.
- Resolved conflict between *jsMath2jax* and *tex2jax* both processing the same equation.
- Fixed problem with `class="tex2jax_ignore"` affecting the processing of sibling elements.

Browser Support

Android

- Added detection and configuration for Android browser.
- Allow use of OTF web fonts in Android 2.2.

Blackberry

- MathJax now works with OS version 6.

Chrome

- Use OTF web fonts rather than SVG fonts for version 4 and above.

Firefox

- Added Firefox 4 detection and configuration.
- Fix for extra line-break bug when displayed equations are in preformatted text.
- Updated fonts so that FF 3.6.13 and above can read them.

Internet Explorer

- Changes for compatibility with IE9.
- Fix for IE8 incorrectly parsing MathML.

- Fix for IE8 namespace problem.
- Fix for null `parentNode` problem.
- Fix for `outerHTML` not quoting values of attributes.

iPhone/iPad

- Added support for OTF web fonts in iOS4.2.

Nokia

- MathJax now works with Symbian[3].

Opera

- Prevent Opera from using STIX fonts unless explicitly requested via the font menu (since Opera can't display many of the characters).
- Fixed bad em-size detection in 10.61.
- Fixed a problem with the About dialog in Opera 11.

Safari

- Use OTF web fonts for Safari/PC.

WebKit

- Better version detection.

33.3.10 Migrating from MathJax v1.0 to v1.1

MathJax v1.1 fixes a number of bugs in v1.0, and improves support for new versions of browsers and mobile devices. It includes changes to increase its performance, and to make it more compliant with HTML5. It has more flexible configuration options, and the ability to load configuration files that combine multiple files into a single one to increase loading speed when MathJax starts up. Finally, MathJax.org now offers MathJax as a web service through a distributed "cloud" server.

This document describes the changes you may need to make to your MathJax configurations in order to take advantage of these improvements.

Configuration Changes

The main changes that you will see as a page author are in the way that MathJax can be loaded and configured. If you have been using in-line configuration by putting a `MathJax.Hub.Config()` call in the body of the `<script>` tag that loads MathJax, then your site should work unchanged with version 1.1 of MathJax. You may wish to consider moving to the new HTML5-compliant method of configuring MathJax, however, which uses a separate `<script>` tag to specify the configuration. That tag should come **before** the one that loads `Mathjax.js`, and should have `type="text/x-mathjax-config"` rather than `type="text/javascript"`. For example,

```
<script type="text/javascript" src="/MathJax/MathJax.js">
  MathJax.Hub.Config({
    jax: ["input/TeX","output/HTML-CSS"],
    extensions: ["tex2jax.js"]
  });
</script>
```

would become

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    jax: ["input/TeX","output/HTML-CSS"],
    extensions: ["tex2jax.js"]
  });
</script>
<script type="text/javascript" src="/MathJax/MathJax.js"></script>
```

instead. This will make sure your pages pass HTML5 validation. Be sure that you put the configuration block **before** the script that loads MathJax. See *Loading and Configuring MathJax* for more details.

If your page simply loads `MathJax.js` and relies on `config/MathJax.js`, then you will need to modify your `<script>` tag in order to use MathJax v1.1. This is because MathJax no longer loads a default configuration file; you are required to explicitly specify the configuration file if you use one. Furthermore, the name of the `config/MathJax.js` file was a source of confusion, so it has been renamed `config/default.js` instead. Thus, if you used

```
<script type="text/javascript" src="/MathJax/MathJax.js"></script>
```

in the past, you should replace it with

```
<script type="text/javascript" src="/MathJax/MathJax.js?config=default"></script>
```

instead. If you don't do this, you will receive a warning message that directs you to a page that explains how to update your script tags to use the new configuration format.

Combined Configurations

New with version 1.1 is the ability to combine several files into a single configuration file, and to load that via the same script that loads MathJax. This should make configuring MathJax easier, and also helps to speed up the initial loading of MathJax's components, since only one file needs to be downloaded.

MathJax comes with four pre-built configurations, and our hope is that one of these will suit your needs. They are described in more detail in the *Using a Configuration File* section. To load one, add `?config=filename` (where `filename` is the name of the configuration file without the `.js`) to the URL that loads `MathJax.js`. For example

```
<script type="text/javascript" src="/MathJax/MathJax.js">
  MathJax.Hub.Config({
    jax: ["input/TeX","output/CommonHTML"],
    extensions: ["tex2jax.js","AMSmath.js","AMSsymbols.js"]
  });
</script>
```

could be replaced by the single line

```
<script type="text/javascript" src="/MathJax/MathJax.js?config=TeX-AMS_CHTML"></
↪script>
```

In this way, you don't have to include the in-line configuration, and all the needed files will be downloaded when MathJax starts up. For complete details about the contents of the combined configuration files, see the *Common Configurations* section.

If you want to use a pre-defined configuration file, but want to modify some of the configuration parameters, you can use both a `text/x-mathjax-config` block and a `config=filename` parameter in combination. For example,

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    tex2jax: {
      inlineMath: [ ['$','$'], ['\\(','\\)'] ],
      processEscapes: true
    }
  });
</script>
<script type="text/javascript" src="/MathJax/MathJax.js?config=TeX-AMS_CHTML"></
↪script>
```

would load the `TeX-AMS_HTML` configuration file, but would reconfigure the inline math delimiters to include `$...$` in addition to `\(...\)`, and would set the `processEscapes` parameter to `true`.

Loading MathJax from a CDN

The MathJax installation is fairly substantial (due to the large number of images needed for the image fonts), and so you may not want to (or be able to) store MathJax on your own server. Keeping MathJax up to date can also be a maintenance problem, and you might prefer to let others handle that for you. In either case, using the MathJax distributed network service may be the best way for you to obtain MathJax. That way you can be sure you are using an up-to-date version of MathJax, and that the server will be fast and reliable.

See *Loading MathJax from a CDN* for more information.

Change in default TeX delimiters

In addition to the fact that MathJax v1.1 no longer loads a default configuration file, there is a second configuration change that could affect your pages. The `config/MathJax.js` file properly configured the *tex2jax* preprocessor to use only `\(...\)` and not `$...$` for in-line math delimiters, but the *tex2jax* preprocessor itself incorrectly defaulted to including `$...$` as in-line math delimiters. The result was that if you used in-line configuration to specify the `tex2jax` preprocessor, single-dollar delimiters were enabled by default, while if you used file-based configuration, they weren't.

This inconsistency was an error, and the correct behavior was supposed to have the single-dollar delimiters disabled in both cases. This is now true in v1.1 of MathJax. This means that if you used in-line configuration to specify the *tex2jax* preprocessor, you will need to change your configuration to explicitly enable the single-dollar delimiters if you want to use them.

For example, if you had

```
<script type="text/javascript" src="/MathJax/MathJax.js">
  MathJax.Hub.Config({
    jax: ["input/TeX","output/HTML-CSS"],
    extensions: ["tex2jax.js"]
  });
</script>
```

and you want to use single-dollar delimiters for in-line math, then you should replace this with

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    jax: ["input/TeX","output/HTML-CSS"],
    extensions: ["tex2jax.js"],
    tex2jax: {
      inlineMath: [ ['$','$'], ['\\(','\\)'] ],
```

(continues on next page)

(continued from previous page)

```
      processEscapes: true
    }
  });
</script>
<script type="text/javascript" src="/MathJax/MathJax.js"></script>
```

The same technique can be used in conjunction with a combined configuration file. For example

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    tex2jax: {
      inlineMath: [ ['$','$'], ['\\(','\\)'] ],
      processEscapes: true
    }
  });
</script>
<script type="text/javascript" src="/MathJax/MathJax.js?config=TeX-AMS_CHTML"></
↪script>
```

will load the pre-defined `TeX-AMS_CHTML` configuration, but will modify the settings to allow `$...$` delimiters, and to process `\$` to produce dollar signs within the text of the page.

New Distribution Location

Version 1.0 of MathJax was distributed through *SourceForge*, but the development of MathJax has switched to GitHub, which is now the primary location for MathJax source code and distributions. The SourceForge repository will no longer be actively maintained (and hasn't been since November 2010), and so you will not be able to obtain updates through `svn` if you checked out MathJax from there.

You may be able to switch to using the MathJax CDN (see above) rather than hosting your own copy of MathJax, and avoid the problem of updates all together. If you must install your own copy, however, you should follow the instructions at *Installing and Testing MathJax*, using either `git` or `svn` as described to obtain your copy from GitHub. This will allow you to keep your copy of MathJax up to date as development continues.

We apologize for the inconvenience of having to switch distributions, but the git-to-svn bridge we tried to implement to keep both copies in synch turned out to be unreliable, and so the SourceForge distribution was retired in favor of the GitHub site.

33.3.11 Converting to MathJax from jsMath

MathJax is the successor to the popular jsMath package for rendering mathematics in web pages. Like jsMath, MathJax works by locating and processing the mathematics within the webpage once it has been loaded in the browser by a user viewing your web pages. If you are using jsMath with its `tex2math` preprocessor, then switching to MathJax should be easy, and is simply a matter of configuring MathJax appropriately. See the section on Loading and Configuring MathJax for details.

On the other hand, if you are using jsMath's `<span class="math">...</span>` and `<div class="math">...</div>` tags to mark the mathematics in your document, then you should use MathJax's `jsMath2jax` preprocessor when you switch to MathJax. To do this, include `"jsMath2jax.js"` in the *extensions* array of your configuration, with the *jax* array set to include `"input/TeX"`. For example,

```
<script type="text/x-mathjax-config">
   MathJax.Hub.Config({
    extensions: ["jsMath2jax.js"]
```

(continues on next page)

(continued from previous page)

```
  });
</script>
<script
  src="https://example.com/MathJax.js?config=TeX-AMS_CHTML">
</script>
```

would load the `jsMath2jax` preprocessor, along with a configuration file that processes TeX input and produces HTML-with-CSS output.

There are a few configuration options for `jsMath2jax`, which you can find in the `config/default.js` file, or in the jsMath configuration options section.

If you are generating your jsMath documents programmatically, it would be better to convert from generating the jsMath `<span>` and `<div>` tags to producing the corresponding MathJax `<script>` tags. You would use `<script type="math/tex">` in place of `<span class="math">` and `<script type="math/tex; mode=display">` in place of `<div class="math">`. See the section on How mathematics is stored in the page for more details.

The links above may refer to sections of the documentation for version 2.7 that are no longer present in the documentation for version 3. In such cases, the links have been removed. The original versions are available in the version 2 documentation pages.

MathJax is a Sponsored Project of NumFOCUS, a 501(c)(3) nonprofit charity in the United States. NumFOCUS provides MathJax with fiscal, legal, and administrative support to help ensure the health and sustainability of the project. Visit numfocus.org for more information.

This version of the documentation was built Apr 28, 2021.