MathJax Documentation

Release 4.0

Davide Cervone, Volker Sorge

May 04, 2026

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, including automatic speech generation and an expression explorer that can be used to investigate typeset mathematics on a more granual level than the complete expression.

Version 4.0 of MathJax adds significant new features to MathJax, including support for selecting one of a number of different fonts to use for mathematical typesetting, for in-line and displayed equation line breaking, and for HTML within MathML and LaTeX expressions. See the *What's New in MathJax* section for more details. MathJax was rewritten from the ground up in v3.0, and the usage and configuration in v4 (and v3) is significantly different from that of MathJax v2. Use the menu at the bottom of the screen to access the version 2 documentation if you need it.

CHAPTER

ONE

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 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 and above 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. Although version 2 was written directly in javascript, versions 3 and 4 are rewritten using Typescript (a version of javascript that includes type-checking and the ability to transpile to ES6 or ES5 versions of javascript). While version 3 was provided as CommonJS modules using ES5, version 4 comes packaged as ES Modules using ES6 as well CommonJS modules in ES5 format.

Since version 3, MathJax can 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

TWO

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 MathJax Menu follows WCAG 2.0 guidelines. Each expression typeset by MathJax is included in the tab order by default (though this can be disabled via a menu option). When an expression is focused, the menu can be triggered via the space or menu key and navigation in the menu is possible using the arrow keys. A menu selection can be made using the Enter or Return key, and the menu can be close without making a selection using the Escape or Esc key.

Note

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.

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, they provide:

- 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 several user-selectable rule sets.
- Tactile rendering enabling Nemeth or Euro Braille output on a connected Braille display.
- An exploration tool, allowing for meaningful exploration of mathematical content including multiple highlighting features, magnification, and synchronized aural rendering.

The accessibility extensions attempt to 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, the `aria-label` and `aria-braillelabel` attributes, among others. We test the assistive tools on 13 different browser/os/screen-reader combinations to ensure that the user experience is an seamless as possible.

In version 4, the extensions for generating speech and exploring expressions are included and enabled in all the *combined components* so that your pages should be accessible to users with screen or Braille readers automatically in that case. (In version 3, the user had to activate the accessibility features to turn them on.)

> **Note**
>
> Support for tactile Braille output devices varies across screen readers, operative systems, and browsers. Users of Braille output devices may need to select the "Combine with Speech" option in the MathJax contextual menu's Braille submenu in order to obtain Nemeth or Euro Braille output rather than the speech text on their Braille device (this can also be accomplished by pressing the b key while exploring an expression interactively). NVDA users in Windows will want to do this, though JAWS users should be fine with the default settings.

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. In particular, the *a11y/explorer* component makes expression exploration possible, and it loads the *a11y/speech* component automatically.

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

> **Note**
>
> In v3, the *a11y/assistive-mml* was included and enabled in the combined components. That is no longer the case in v4, which now uses the speech and explorer components instead. Users can still use the MathJax contextual menu to turn on the hidden MathML, which will turn off the speech and Braille generation that underlies the explorer component, if they want to have a similar experience to the one from v3.

2.3 Screen Reader Support

Screen reader support in Mathjax v3 was based on the *a11y/assistive-mml* component, which embedded a MathML representation of each expression that was visually hidden, but available to screen readers, in addition to its typeset version that was visible for sighted users, but hidden from screen readers. The quality of MathML support in screen-readers varies greatly, with different levels of MathML feature support, different speech rule sets, and different voicing technologies.

Using hidden MAthML only worked with those screen readers that understand MathML, and even then, the experience was different depending on the screen reader and browser that was being used. As screen-reader and browser versions changed, they sometimes failed to work properly with the hidden MathML; for example, some screen readers skip the math entirely when reading the page as a whole, even when they voice the math while stepping through the page in smaller units. This makes it difficult to maintain this feature.

In version 4, screen reader support is now being handled through the *a11y/explorer* and *a11y/speech* components of MathJax. These tools generate speech strings from the math expressions in the page and inserts them via `aria-label` and `aria-braillelabel` attributes so that screen readers will be able to voice the mathematics regardless of whether they understand MathML or not. The *a11y/semantic-enrich* component analyses the expression and improves the internal representation to make screen reading and line breaking more accurate.

The explorer component then makes it possible to investigate an expression interactively by allow the user to descend into the expression and read it term by term rather than all at once. The code that handles the exploration has been rewritten in v4 to make it more compliant with the ARIA standards, and to make it work more reliably across browsers, operating systems, and screen readers so that everyone should have an effective experience.

The explorer is designed to:

- Work across the major browser/OS/screen-reader combinations.

- Properly read the full expression when reading the whole page or stepping through the page sentence-by-sentence (or by other units).
- Automatically enter "focus mode" when the expression is focused via tabbing or clicking, in screen readers that have separate focus/browser modes.
- Allow control over the description of the math (e.g., "clickable math") used during reading and focusing (via a menu option).
- Voice the phrase "press H for help" when the math is first focused (but this can be turned off by a menu preference).

2.3.1 Accessibility Extensions

MathJax offers accessibility support via its own built-in extensions that provide a choice of support options as well as a high degree of personalization. The extensions can be activated either via the context menu, which itself is fully accessible, or a page author can control the default settings using the MathJax configuration object. When the menu component is available, configuration should be done using the *MathJax menu options*, otherwise use the *accessibility options*.

Most features of the accessibility extensions are based on technology provided by the Speech Rule Engine, a powerful tool for enhancing MathML expressions with additional structural information, and for converting the enriched MathML into speech text that can be read by a screen reader.

The accessibility features are controlled via the MathJax contextual menu. In version 3, these were in an *Accessibility* sub-menu, but in v4, there is a new *Accessibility* section in the main menu, with sub menus for speech, Braille, the explorer, and other options.

Interactive Exploration

The main feature is an interactive exploration mode that allows a reader to traverse and explore sub-expressions step-by-step. Expressions typeset by MathJax are focusable, and when the user tabs to or clicks on an expression, that starts the explorer, either on the expression as a whole, or on the clicked sub-expression. Double-clicking starts the explorer at the top level, just like tabbing.

Exploration of the expression is performed using keyboard commands. These are described in the *Explorer Keyboard Commands* section. During traversal, focused sub-expressions are highlighted and optionally magnified; an aural rendering is pushed to a screen reader, and a tactile rendering can be presented on a Braille display, if one is connected, as described in the next section.

The *Explorer* sub-menu controls how the highlighting of the selected sub-expression is performed. Its *Highlight* sub-menu allows you to select the background and foreground colors and their opacities. Several other options are described in the *Highlighting* section. The *Magnification* sub-menu is described below in the *Magnification* section.

The *Describe Math as* sub-menu lets you pick the phrase that will be used when the screen reader voices a typeset expression, while *Help message on focus* determines whether the "Press H for help" message is read when the math is first focused.

Speech & Braille Support

Both aural and tactile rendering can be controlled via the options in the *Speech* and *Braille* sub-menus. The *Generate* item in each menu tells MathJax whether to include that format in its output, so you can enable each form separately. The *Show subtitles* item determines whether the speech and/or Braille are shown on screen below the expression as it is being explored. This can even help sighted readers to know how an expression should be pronounced.

The *Speech* sub-menu includes an option for *Auto Voicing* the expression as it is navigated. When this is selected, the expression will be read by the browser's speech synthesis API, rather than a screen reader, and the terms will be highlighted as they are read. This is useful, for example, for dyslexic users who benefit from the synchronized highlighting. *Auto Voicing* can also be toggled using the S key.

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 *ClearSpeak*, however, the *Speech* sub-menu also provides the *MathSpeak* option.

Each rule set has several different preference settings — three in the case of MathSpeak, for example, which primarily influence the length of the speech text that it produces. 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 relevant for the sub-expression that is currently selected. The *Select Preferences* option opens a selection box for all possible ClearSpeak preference choices.

Some rule-set and preference settings can 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 language can be selected via the *Language* sub-menu in the *Speech* menu. MathJax currently supports speech in more than a dozen languages, including English, French, German, Hindi, Spanish and Korean. Braille output is available in two forms: Nemeth and what we call Eruo Braille. The latter uses 8-dot Braille to represent the underlying LaTeX expressions, when available, as is currently being taught in gradeschools in a number of European countries.

In addition to voicing the sub-expressions during exploration, the explorer allows queries on sub-expressions, such as getting positional information with respect to the context, as well as obtaining summaries of the sub-expression currently being explored. See the *Explorer Keyboard Commands* section for details.

Highlighting

During interactive exploration, the sub-expression that is being explored is automatically highlighted, by default with a blue background color. The highlighting can be customized by changing *Background* or *Foreground* colors in the *Highlight* sub-menu of the *Explorer* section 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 the mouse pointer is hovering over it.

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

Magnification

During expression exploration, the explorer can optionally magnify the sub-expression that is currently selected. The zoomed version of the expression is overlaid above the original one when traversing the formula. For keyboard exploration, this can be switched on in the *Magnification* sub-menu of the *Explorer* 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 where the mouse is pointering 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.

Collapsible Expressions

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.

Sub-expressions can be collapsed in this way either by clicking on them with the mouse (the pointer should become a "pointing hand" when that is possible), or by using the explorer to navigate to the expression and then pressing the Enter or Return key. Clicking or pressing one of the these keys again will return the expression to its original form. Collapsible expressions can also be discovered using some of the highlighting features, as described above.

The ability to collapse sub-expressions is controlled by the *Collapsible Math* setting in the *Options* sub-menu of the MathJax contextual menu. This feature is off by default, but can be selected by the user, or the default can be changed by the page author using the *contextual menu configuration options*.

2.3.2 Explorer Keyboard Commands

The expression explorer is used to interact with a mathematical expression using keyboard commands. This 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 starts when a MathJax expression is focused and can be quit at any time during the exploration. When navigation is restarted, the application will begin at the top level of the expression again. There is a menu setting that instead causes the explorer to start where the user has left off within the expression when it is re-focused. Note, however, that this may mean that only that sub-expression will be read when the page is read as a whole.

Overview of key bindings

Essential keys

Down
: Explore the next lower level of the formula by moving down in the sub-expression tree. Exploration will start at the left-most sub-expression on the level.

Right
: Navigate the expression horizontally by moving to the next sub-expression on the current level.

Left
: Navigate the expression horizontally by moving to the previous sub-expression on the current level.

Up
: Move up the sub-expression tree.

Tab
Move to the next focusable item in the page.

Escape
Leave exploration mode while keeping the expression focused.

Enter
Re-activate the explorer after leaving it, provided it still has the focus.

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

Advanced Keys

Space
Opens the MathJax contextual menu.

Enter
Collapse or expand the expression under cursor, if possible. The speech-text is regenerated to match.

Home
Jump directly to the top-most level of the expression.

S
Toggle auto-voicing with synchronous highlighting. This uses the browser's speech synthesis API directly, so is available even without a screen reader. This mode remains in effect until turned off. It can also be controlled via the MathJax contextual menu.

B
Toggle the "Combine with Speech" menu item, which tells MathJax whether to include the Braille rendering within the `aria-label` attribute along with the speech text or whether to put it in the `aria-braillelabel` attribute. See the notes below.

D
Get positional information; i.e., the current level in the sub-expression tree as well as collapsibility/expandability of the current subexpression.

X
Summarise the selected sub-expression, without collapsing it.

Z
Give a detailed description of the selected sub-expression, without expanding it.

V
Mark the current position for later retrieval.

P
Go to the last marked position in the expression. Repeated use cycles through the marked locations in the expression.

U
Undo all marked locations, and go to the position where navigation was started.

>
Switch rule sets between MathSpeak and ClearSpeak, if both are available for the current locale.

<
Cycle styles or preferences for the currently active rule sets.

Special keys for navigating tables

Shift + Down
: Move one cell vertically down in the table.

Shift + Up
: Move one cell vertically up in the table.

Shift + Right
: Move one cell horizontally right in the table.

Shift + Left
: Move one cell horizontally left in the table.

0-9 + 0-9
: Jump directly to cell (n,m) if it exists. (0,0) is cell (10,10).

Special Notes

Some screen readers have separate "focus" and "browse" modes. MathJax's explorer is set up to initiate focus mode when an expression is focused, but in some situations that may not occur automatically. In that case, you will have to enter focus mode manually in order to use the explorer. How this is done depends on the screen reader, so consult its documentation for details.

Once the focus mode is started, you may need to exit focus mode manually when you are done exploring the expression. How that is done depends on the screen reader, so consult the reader's documentation for details on how to accomplish that.

For users of tactile Braille devices, MathJax handles the Braille notation using the `aria-braillelabel` attribute (part of the ARIA specification), but not all screen readers process this attribute properly. To help accommodate this limitation, MathJax provides an option that will put the Braille into the `aria-label` attribute instead, along with the generated speech, as some screen readers will pass the Braille on to the Braille device that way. NVDA users will want to enable this feature, for example.

This feature can be enabled by selecting the "Combine with Speech" item in the Braille submenu of the MathJax contextual menu, or by pressing the b key while exploring an expression.

CHAPTER

THREE

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 TeX's `$...$` 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 typeset as mathematics, since it falls between dollar signs, and those dollar signs would be removed, as they would be treated as math delimiters. 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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-chtml.js">
</script>
</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 an HTML 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 and ampersands, since those are what the browser uses to indicate the start of a tag in HTML or a symbol entity name. Putting a space on both sides of these characters 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 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, although this is not strictly required, and this is preferred to the use of a namespace prefix like `m:` above, since those are deprecated in HTML5.

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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4/mml-chtml.js">
</script>
</head>
<body>

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

(continues on next page)

(continued from previous page)

```
        <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. MathJax does try to unravel the broken DOM tree, but this process is imperfect. Unfortunately, there is little 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 as one of its input processors. This has not been fully ported to MathJax version 3 and above, but there is a version of it that uses the legacy v2 code to patch it into MathJax v3 and v4. 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", "ui/menu"]},
  output: {font: "mathjax-newcm"}
```

(continues on next page)

(continued from previous page)

```
}
</script>
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/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>
```

Here we are loading the input and output components separately rather than using a combined configuration file. Since the `output/chtml` component does not have a font configured with it, we need to specify the font explicitly in the `output` section of the MathJax configuration. It is also possible to load a combined component like `tex-chtml.js` rather than `startup.js` and include `input/asciimath` in the `load` array of the `loader` block of the configuration, in which case the `mathjax-newcm` font will already be included, and won't need to be specified separately.

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,

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

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

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

to produce an array with two rows.

It is also possible to use the `String.raw` constructor to create strings with backslashes that don't need to be doubled. For example,

```
const math = String.raw`\frac{1}{\sqrt{x^2 + 1}}`;
```

is equivalent to the first declaration above.

CHAPTER

FOUR

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 Developers 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.

When you create an issue, you will need to choose between a bug report and a feature request. Each of these comes with a template that you should use to structure your report or request. **Do not just erase that template**, as the information that it requests is important for us to be able to analyze your issue and give you an answer. 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

FIVE

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 by filling out the issue template. **Do not just erase the template** and write a free-form report. The information requested in the template may be crucial to diagnosing your problem, and without it, we may not be able to give you a meaningful response. Be sure to include 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.
  - The MathJax configuration you are using, and the MathJax component you are loading (e.g., `tex-chtml.js`).
  - If a particular expression is not displaying properly, then please include the original LaTeX, MathML, or AsciiMath version of the expression as text (that can be copied), as well as a screen snapshot of the result you are seeing in your browser.
  - If you are making calls to the MathJax API, how and when are you doing so? It would help to include a code snippet with the actual calls you are making. Because the timing of such calls is often at issue, the surrounding code may also be important to include.
  - 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

SIX

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 these 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 *combined 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 the CDN copy of 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

MathJax no longer provides a means of obtaining the latest version itself. This used to be done by loading the `latest.js` file, but that has been removed in version 4.

The following are some of the CDNs that offer MathJax:

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

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

```
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/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 4.x.y 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 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 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@4` notation for obtaining the `tex-mml-chtml.js` file from the latest version (4.x.y) available on the CDN.

6.2.2 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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4.0.0/tex-mml-chtml.js"></script>
```

will always load version 4.0.0 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 src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/4.0.0/tex-mml-chtml.js"></
↪script>
```

to get the same file.

6.2.3 Browser Compatibility

MathJax supports all modern browsers (Chrome, Safari, Firefox, Edge), and most mobile browsers. We no longer test MathJax with IE11, so you should not expect it to work with any version of Internet Explorer. We used to recommend loading the `polyfill` library in order to help MathJax work with older browsers, but that is not necessary for modern browsers, and we no longer recommend it.

Warning

The original *polyfill* website was purchased by a Chinese company in 2024, and has been used to inject malware into pages that use it. You should **NOT** use the `polyfill.io` library any longer, and should either remove the reference entirely, or switch to a link from another source. See this post for more details.

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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-chtml.js"></script>
```

configures MathJax's TeX input component to add `$...$` as inline-math delimiters in addition to the default `\(...\)` delimiters (thus 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 instructions for *Configuring MathJax* for more details, and the section on *MathJax Configuration Options* 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 in your web page, and you should be able to use MathJax to write web pages that include mathematics. At this point, you can start making pages that contain mathematical content!

You could 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 *MathJax in Dynamic Content*, 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

SEVEN

CONFIGURING MATHJAX

MathJax provides a number of combined component files, like `tex-chtml.js`, that group various components that are commonly used together into a single file. If you use one of the combined component files, you may not need to do any configuration at all. You can modify the default configuration by using a javascript variable that you define before loading the combined configuration file, as described below.

Note

The configuration, loading, and startup processes for MathJax versions 3 and 4 are different from those of version 2 in a number of ways. Where version 2 had several different methods for configuring MathJax, v3 and v4 streamline the process and have only one. In version 2, you always loaded `MathJax.js`, and added a `config=...` parameter to provide a combined configuration file, but in version 3 and above you load one of several different files, depending on your needs, avoiding the multiple file transfers that was required in v2.

7.1 The Configuration Variable

To configure MathJax, you use a global javascript 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 *MathJax Configuration Options* pages.

7.2 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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-svg.js"></script>
```

This will configure the TeX input component to add 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.

7.3 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 defer src="mathjax-config.js"></script>
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-svg.js"></script>
```

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

> **Note**
>
> 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.

7.4 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@4/tex-svg.js';
  script.defer = 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.)

7.5 Configuring MathJax After it is Loaded

As described above, 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 `startup` *`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 those objects, 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.document.inputJax.tex.findTeX`; but in this case, the `FindTeX` object uses the configuration once when it is created, so changing `MathJax.startup.document.inputJax.tex.findTeX.options` after the fact will not affect it. (There is a `getPatterns()` method of 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.document`), and 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. This includes the data about the global font cache for SVG output, and the CHTML CSS cache, so this is not something you should do lightly.

7.6 Converting your old Configuration to v4

The configuration options for v4 are basically the same as for v3, with some new ones added, you should be able to use your current v3 configuration in v4 without change. The only major caveat is if you have used a *ready()* function in the `startup` section of your configuration to make modifications or additions to MathJax's code, in which case, those might need to be adjusted. See the *What's New in MathJax* section for more details.

Because the current MathJax 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 the current version. Simply paste your current `MathJax.Hub.Config()` call into the converter, press `Convert` and you should get the equivalent v3/v4 configuration, and comments about any options that could not be translated to the current version (some options are not yet implements, others no longer make sense in version 4). See the instructions on the converter page for more details.

CHAPTER

EIGHT

LOADING MATHJAX

Once you have *configured* MathJax, you can 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/LaTeX input and produce SVG output. To do so, use a script like the following

```
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-svg.js"></script>
```

to get the latest (4.x.y) version of the `tex-svg` component from the `jsdelivr` CDN.

Warning

Version 3 used `/es5` just before the component name in the URL for obtaining the MathJax. This is no longer the case for version 4.

The example above takes advantage of the feature of `jsdelivr` that allows you to get the latest version using the `mathjax@4` notation. To obtain a specific version, you would use a tag like

```
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4.0.0/tex-svg.js"></script>
```

to always get the 4.0.0 version of the `tex-svg` component, even when later versions become available.

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

```
<script defer src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/4.0.0/es5/tex-svg.js">
</script>
```

Some CDNs don't provide a means of getting the latest version automatically, so you should check the documentation for the CDN you are planning to use to see if they support that, and how to indicate it in your source URL.

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 sets the `MathJax` configuration variable should come **before** the script that loads the MathJax component file, otherwise MathJax will not know what configuration you need. If you use one of the combined component files, you may not need to do any configuration at all.

8.1 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 load the `startup.js` component.

For example

```
<script>
MathJax = {
  loader: {
    load: ['input/tex-base', 'output/svg', 'ui/menu', '[tex]/require']
  },
  output: {
    font: 'mathjax-newcm'
  },
  tex: {
    packages: {'[+]': ['require']}
  }
};
</script>
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/startup.js"></script>
```

would cause the base TeX input, the SVG output (with the `mathjax-newcm` font), the contextual menu code, and the TeX `\require` macro extension components to be loaded (and tells 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.

8.2 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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4/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).

8.3 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, and some mobile devices don't cache files that are larger than a certain limit.

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 () {
  const body = document.body.textContent;
  if (body.match(/(?:\$|\\\(|\\\[|\\begin\{.*?})/)) {
    if (!window.MathJax) {
      window.MathJax = {
        tex: {
          inlineMath: {'[+]': [['$', '$']]}
        }
      };
    }
    const script = document.createElement('script');
    script.src = 'https://cdn.jsdelivr.net/npm/mathjax@4/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-for-tex.js`, that you should change `tex-chtml.js` to whatever combined-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>`).

CHAPTER

NINE

PERFORMING ACTIONS DURING STARTUP

There are 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. The primary way to do this is to use one of two hooks that can be 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 requested MathJax components have been loaded, and MathJax is ready to set up the internal objects needed to process the mathematics on the page (like the input and output jax). This function builds those structures, creates functions in the `MathJax` object to make typesetting and format conversion easy for you, sets up the *pageReady()* call (described below), and creates a promise for when that is complete. You can override the *ready()* function with one of your own to replace that startup process completely, or to perform actions before or after the usual initialization. For example, you could do additional setup before MathJax creates 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 the HTML page is loaded and ready to be typeset, and MathJax itself is ready (all its components are loaded, and the internal objects have been created). 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 the default *pageReady()* is a promise that is resolved when the initial typesetting is finished. If you override the *pageReady()* method, your function should return a promise as well. For example, 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). If you don't, then MathJax will think that the initial typesetting is complete even though it isn't, which can lead to incorrect behavior if other typesetting needs to be performed later.

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 for common situations are given below.

9.1 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 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. It is also possible to create TeX extensions on the fly and add them at this point.

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.

In addition, the variable *MathJax.startup.promise* 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. You can use this promise to set up actions that should occur after the initial typesetting is complete. This is discussed further in the next section.

```
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, but hasn't␣
↪run");
      MathJax.startup.promise.then(() => {
        console.log("The initial typesetting is complete");
      });
    }
  }
};
```

The console messages above indicate the MathJax's state at each point in the code. For example, you can't do any typesetting in the section before *MathJax.startup.defaultReady()*, while you potentially could after it, thought it is better to wait for the *MathJax.startup.promise* before doing so, except in special circumstances where you know that the typesetting will not cause any extensions to be loaded dynamically.

Here is an example that uses the *ready()* function to convert the numbers in the full-width Unicode block to their ASCII counterparts for better formatting by MathJax.

```
MathJax = {
  startup: {
    ready() {
      MathJax.startup.defaultReady();
      MathJax.startup.document.inputJax.tex.preFilters.add(
        ({math}) => {
          math.math = math.math.replace(/[\uFF01-\uFF5E]/g,
            (c) => String.fromCodePoint(c.codePointAt(0) - 0xFF00 + 0x20));
        }
      );
    }
  }
}
```

This configuration adds a pre-filter to the TeX input jax that performs a substitution on the TeX source (`math.math`) for each expression being processed that looks for full-width numerals and replaces them with the corresponding ASCII numerals.

Here is an example that waits for the initial typesetting to complete and then prints to the console the TeX code for all the expressions on the page.

```
MathJax = {
  startup: {
    ready() {
      MathJax.startup.defaultReady();
```

(continues on next page)

(continued from previous page)

```
      MathJax.startup.promise.then(() => {
        for (const item of MathJax.startup.document.math) {
          console.log(item.math);
        }
      });
    }
  }
}
```

Finally, here is an example that modifies the AsciiMath input jax to allow both in-line and display-mode equations by using `` `...` `` for in-line delimiters and `` ``...`` `` for display-mode delimiters.

```
MathJax = {
  loader: {load: ['input/asciimath', 'output/chtml']},
  output: {font: 'mathjax-newcm'},
  asciimath: {
    delimiters: [['```','```'], ['``','``']]
  },
  startup: {
    ready() {
      const {AsciiMath} = MathJax._.input.asciimath_ts;
      Object.assign(AsciiMath.prototype, {
        _compile: AsciiMath.prototype.compile,
        compile(math, document) {
          math.display = (math.start?.delim === '```');
          const result = this._compile(math, document);
          const mstyle = result.childNodes[0].childNodes.pop();
          mstyle.childNodes.forEach(child => result.appendChild(child));
          if (math.display) {
            result.attributes.set('display', 'block');
          }
          return result;
        }
      });
      MathJax.startup.defaultReady();
    }
  }
};
```

This saves the old AsciiMath `compile()` function and replaces it with a new one that removes the original `mstyle` element created by AsciiMath that sets the display mode, and sets the mode on the outer `math` tag depending on the delimiter used.

9.2 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, which may cause other code to run too soon.

Our first example above shows how to use *MathJax.startup.promise* within the *ready()* function, but that promise is set up as soon as MathJax is loaded, so it can be used outside of the *ready()* function. You must be careful, however, that MathJax is loaded before you try to use it. For example, if you use `defer` or `async` attributes on the script tag that loads MathJax, then you need to be sure your code that uses *MathJax.startup.promise()* doesn't run until after MathJax has been loaded.

One way to do that is to use `defer` on both the script that loads MathJax and the one that uses *MathJax.startup.promise*, and to put your script **after** the one that loads MathJax. Since deferred scripts run in the order in which they appeared in the HTML document, that will guarantee that *MathJax.startup.promise* will be defined when you use it. For example,

```
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-mml-chtml.js"></script>
<script defer src="mathjax-dependent-code.js"></script>
```

where the `mathjax-dependent-code.js` file contains the *MathJax.startup.promise* reference, such as

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

In this case, the MathJax-dependent code won't run until after MathJax is loaded.

You can't use the `defer` attribute on a script tag without a `src` attribute, but if you want to use an in-line script that uses *MathJax.startup.promise*, then you can use a script with `type="module"`, as these have the `defer` attribute by default. For example,

```
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-mml-chtml.js"></script>
<script type="module">
MathJax.startup.promise.then(() => {
  console.log('MathJax initial typesetting complete');
});
</script>
```

will work to guarantee that the promise is defined when the script is executed.

9.3 Summary

The following terms were discussed above:

ready()

This is the function called when MathJax has loaded the needed components and is ready to start setting up the objects needed for typesetting the document. You can override it in the `startup` section of the `MathJax` configuration object in order to perform customization when MathJax loads, or to set up actions to perform after the initial typesetting is complete.

`MathJax.startup.`**defaultReady**()

This is the default for the *ready()* function above. You can call it from your `ready()` function in order to perform the usual `ready()` action.

pageReady()

This is the function called when the page is loaded and MathJax is ready to perform typesetting. You can override it in the `startup` section of the `MathJax` configuration object in order to do your own processing, or to set up actions to perform after the initial typesetting is complete.

Returns
A promise that resolves when the actions taken by your function is complete.

`MathJax.startup.`**defaultPageReady**()

This is the default for the *pageReady()* function above. You can call it from your `pageReady()` function in order to perform the usual `pageReady()` action.

Returns
A promise that resolves when the initial page typesetting is complete.

`MathJax.startup.`**promise**

A promise that resolves when MathJax completes its initially typesetting.

CHAPTER

TEN

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 individual TeX extensions, for the various font selections, and for other specialized pieces, such as the MathJax contextual menu and 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 almost 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, and MathJax may autoload other components when they are needed.

Note

In version 3, there were a number of combined components that included most of the TeX extensions. These components ended in `-full`. Because the number of extensions has grown, and continues to do so, including with third-party extensions, it is impractical to include all the extensions in a single file. In version 4, the `-full` versions have been removed, along with the `all-packages` extension.

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 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 *MathJax Configuration Options* section for details about how to configure the various MathJax components.

10.1 Combined Components

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

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

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

The combined components include everything needed to run MathJax in your web pages (though some TeX extensions and additional font data may be loaded dynamically as needed). Each includes at least one input processor, an output processor, the basic data needed for the mathjax-newcm font, the contextual menu code, the assistive tools, 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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-chtml.js"></script>
```

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

Warning

Version 3 used `/es5` just before the component name in the URL for obtaining the MathJax. This is no longer the case for version 4.

10.1.1 tex-chtml

The *tex-chtml* component includes the *input/tex* component and the *output/chtml* component configured to use the `mathjax-newcm` font, along with the contextual menu component, the assistive tools, and the startup component.

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

10.1.2 tex-svg

The *tex-svg* component includes the *input/tex* component and the *output/svg* component configured to use the `mathjax-newcm` font, along with the contextual menu component, the assistive tools, and the startup component.

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

10.1.3 tex-mml-chtml

The *tex-mml-chtml* component includes the *input/tex* and *input/mml* components and the *output/chtml* component configured to use the `mathjax-newcm` font, along with the contextual menu component, the assistive tools, and the startup component.

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

10.1.4 tex-mml-svg

The *tex-mml-svg* component includes the *input/tex* and *input/mml* components and the *output/svg* component configured to use the `mathjax-newcm` font, along with the contextual menu component, the assistive tools, and the startup component.

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

10.1.5 mml-chtml

The *mml-chtml* component includes the *input/mml* component and the *output/chtml* component configured to use the `mathjax-newcm` font, along with the contextual menu component, the assistive tools, and the startup component.

10.1.6 mml-svg

The *mml-svg* component includes the *input/mml* component and the *output/svg* component configured to use the `mathjax-newcm` font, along with the contextual menu component, the assistive tools, and the startup component.

10.1.7 tex-chtml-nofont

The *tex-chtml-nofont* component is the same as the *tex-chtml* component, but configured without a font, with the expectation that your configuration will specify the font explicitly. This reduces the size of the initial download when the `mathjax-newcm` font is going to be replaced by one of the other fonts.

10.1.8 tex-svg-nofont

The *tex-svg* component is the same as the *tex-svg* component, but configured without a font, with the expectation that your configuration will specify the font explicitly. This reduces the size of the initial download when the `mathjax-newcm` font is going to be replaced by one of the other fonts.

10.1.9 tex-mml-chtml-nofont

The *tex-mml-chtml* component is the same as the *tex-mml-chtml* component, but configured without a font, with the expectation that your configuration will specify the font explicitly. This reduces the size of the initial download when the `mathjax-newcm` font is going to be replaced by one of the other fonts.

10.1.10 tex-mml-svg-nofont

The *tex-mml-svg* component is the same as the *tex-mml-svg* component, but configured without a font, with the expectation that your configuration will specify the font explicitly. This reduces the size of the initial download when the `mathjax-newcm` font is going to be replaced by one of the other fonts.

10.1.11 mml-chtml-nofont

The *mml-chtml* component is the same as the *mml-chtml* component, but configured without a font, with the expectation that your configuration will specify the font explicitly. This reduces the size of the initial download when the `mathjax-newcm` font is going to be replaced by one of the other fonts.

10.1.12 mml-svg-nofont

The *mml-svg* component is the same as the *mml-svg* component, but configured without a font, with the expectation that your configuration will specify the font explicitly. This reduces the size of the initial download when the `mathjax-newcm` font is going to be replaced by one of the other fonts.

10.2 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.

10.2.1 input/tex

The TeX input format is packaged in two 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.

Note

As the number of TeX extensions available in MathJax has grown, and with the availability of third-party extensions, it is impractical to include all of them in one components. Thus the *input/tex-full* extension has been dropped from version 4.

When you include one of the TeX input components, MathJax will define functions to convert TeX strings into the output format that has been loaded and into the MathML format. 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*, *textmacros*, and *noundefined* extensions. Most of the remaining extensions 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-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 most of the extensions explicitly, as they will be loaded automatically when first used. You can also use `\require` to load an extension explicitly, if needed.

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

Note

Version 3 included a `[tex]/all-packages` components that included most of the TeX extension packages. Due to the growing number of extensions, including third-party extensions, the `all-packages` extension has been dropped from v4.

10.2.2 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.

10.2.3 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 functions to convert AsciiMath strings into the output format that has been loaded, and into the MathML format. 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 v3/v4 yet. The AsciiMath component includes legacy MathJax 2 code patched into the new MathJax framework. That makes the AsciiMath component larger than usual, and slower than the other input components.

10.3 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 above, and is unlikely to be. See the *MathML Support* section for details.

10.3.1 output/chtml

The *output/chtml* component includes the CommonHTML output processor. It does not include a default font, however, so you need to configure one explicitly in the `font` option in the `output` block of your MathJax configuration when you load this component.

- 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.
- See the *MathJax Font Support* section for information about the fonts available in MathJax.

10.3.2 output/svg

The *output/svg* component includes the SVG output processor. It does not include a default font, however, so you need to configure one explicitly in the `font` option in the `output` block of your MathJax configuration when you load this component.

- See the *SVG Support* section for details on the SVG output processor.
- See the *SVG Output Processor Options* section for information about configuring this component.
- See the *MathJax Font Support* section for information about the fonts available in MathJax.

10.4 Accessibility Components

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

- *a11y/semantic-enrich*
- *a11y/speech*
- *a11y/explorer*
- *a11y/complexity*
- *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.

The *semantic-enrich*, *speech*, and *explorer* components are part of all the *combined components*, so if you are using one of the those, you don't need to load these assistive extensions yourself.

Note

In v3, the *semantic-enrich* extension included both the semantic enrichment and the speech generation features of MathJax. In v4, these functions are now accomplished by two distinct extensions, with the speech functionality being split off into a separate *speech* extension.

If you are not using a combined component, by are using the *ui/menu* component, there are menu items that will cause these assistive extensions to be loaded dynamically, so you don't need to load them explicitly in that case.

Note

The *auto-collapse* extension from version 2 has not yet been converted to the current version of MathJax, but will be in a future release.

Note

The *assistive-menu* extension from version 2 is now part of the standard *contextual menu extension*, so doesn't have to be loaded separately.

10.4.1 a11y/semantic-enrich

The *semantic-enrich* component connects MathJax with the Speech Rule Engine, which allows MathJax to analyze the mathematics that it processes and add attributes that represent the semantic structure of the mathematics. The underlying MathML for each expression may be modified to better represent that structure, improving line-breaking and speech generation for the mathematics produced by MathJax.

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

10.4.2 a11y/speech

The *speech* 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.

The speech component uses web-workers to do the speech computations (which can be time-consuming) in a separate thread so that it doesn't interfere with the responsiveness of your web page or slow down the display of the typeset math in your page.

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

10.4.3 a11y/explorer

The *explorer* component allows readers to explore a mathematical expression interactively. When an expression is focused by tabbing to it, the expression can be explored using the arrow keys, as described below. Clicking on a typeset expression will also enter the explorer at the character that is clicked.

Once the explorer is activated, the arrow keys 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.

10.4.4 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 above.

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

10.4.5 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 in 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.

> **Note**
>
> In MathJax v2 and v3, the *assistive-mml* extension was loaded an enabled by default, but in v4, the *explorer* component has replaced it as the default assistive tool. The explorer can be disabled, however, and the assistive MathML re-enabled using the MathJax contextual menu on any typeset expression, and page author's can override the defaults in their MathJax configuration objects, if they so desire.

10.5 Miscellaneous Components

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

- *startup*
- *ui/safe*
- *ui/lazy*
- *ui/menu*

- *adaptors/liteDOM*
- *core*
- *loader*

They are described in more detail below.

10.5.1 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 in the `loader` section of your MathJax configuration to specify the components you want to load. Like you would a combined component, you load the *startup* component directly via a `<script>` tag, as in

```
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/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 Mathematics* section.

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

10.5.2 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.

10.5.3 ui/lazy

The *ui/lazy* component changes the way MathJax handles the timing for typesetting expressions in the page. Normally, MathJax will typeset all the expressions in one pass over the page. When there are many expressions in your document, that can take a significant amount of time, and cause a noticeable delay before the mathematics becomes available.

With the *lazy* extension, MathJax will only typeset expressions when they come into view in the browser. That means the user gets to see the mathematics at the top of the page (or wherever the initial link takes them on your page) right away, and MathJax won't take time to typeset expressions that are never seen. This makes even pages with a lot of mathematics appear more performant.

If you have pages with many expressions, it may be a good idea for you to use the *lazy* extension.

See the *Lazy Typesetting Options* section for more information on See *Lazy Typesetting* for additional details.

10.5.4 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, control accessibility features, and so on. The menu extension is included in all the combined components provided by MathJax.

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

10.5.5 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.

See the *The DOM Adaptor* section for more information about DOM adaptors in MathJax.

10.5.6 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.

10.5.7 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

ELEVEN

TYPESETTING 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. The current version of MathJax makes both easy to do. Typesetting math is described below, while converting math is described in the *next section*.

11.1 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 a `retry` error will be thrown. In that case, you should use the `MathJax.typesetPromise()` instead, as described below. This will make your typesetting asynchronous, however, and you may need to take that into account in the rest of your code. See also *The "MathJax Retry" Error* section for more details.

Warning

In MathJax v4, with the introduction of new fonts that include many more characters than the original MathJax TeX fonts did, the fonts have been broken into smaller pieces so that your readers don't have to download the entire font and its data for characters that may never be used. That means that typesetting mathematics may need to operate asynchronously even if the TeX *doesn't* include `\require` or any auto-loaded extensions, as the output itself could need extra font data files to be loaded. Thus in version 4, it is always best to use the promise-based commands, as described below.

The second method, `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. Note that it can be used with `await` as part of a larger `async` function. If you are getting a `retry` error while calling `MathJax.typeset()`, you should switch to using `MathJax.typesetPromise()` instead.

Each of these 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 one or more DOM elements. Supplying an array of elements will restrict the typesetting to the contents of those elements only.

`MathJax.`**`typeset`**`(`[*elements*]`)`

Arguments

- **elements** (`(string|HTMLElement)[]()`) – An optional array of DOM elements or CSS selector strings that restricts the typesetting to the contents of the specified container elements.

`MathJax.`**`typesetPromise`**`(`[*elements*]`)`

Arguments

- **elements** (`(string|HTMLElement)[]()`) – An optional array of DOM elements or CSS selector strings that restricts the typesetting to the contents of the specified container elements.

Returns Promise
A promise that resolves when the typesetting is complete.

11.2 Handling Asynchronous Typesetting

It is not recommended to perform multiple asynchronous typesetting calls simultaneously, as these can interfere with one another while they are waiting for files to load. For this reason, MathJax chains the promises returned by the *`MathJax.typesetPromise()`* function to make sure any previous typeset calls are complete before starting the new one. So if you do

```
MathJax.typesetPromise(["#container1"]);
MathJax.typesetPromise(["#container2"]);
```

the second typeset operation will wait for the first one to complete before it starts. This applies as well to the promise-based conversion functions described in the *next section*, which also use the promise chain to coordinate with other typesetting and conversion operations.

Warning

This automatic use of promises to serialize the typeset and conversion calls is new in version 4. In version 3, you were expected to handle chaining of the typeset calls yourself, but most coders failed to do this, so MathJax v4 now does that for you.

The version 3 documentation recommended using and setting *`MathJax.startup.promise`* yourself to make sure typeset calls were serialized; if you included that code pattern in your v3 work-flow, you should remove it, as it is no longer necessary.

Because *`MathJax.typesetPromise()`* returns a promise, you can use that promise to synchronize the rest of your code with the actions of MathJax. For example,

```
MathJax.typesetPromise().then(() => {
  for (const eqn of MathJax.startup.document.math) {
    console.log(eqn.math);
  }
});
```

would typeset the math on the page and then print the original TeX code to the console for each of the expressions on the page.

It is also possible to use the `await` command to wait for the promise to be resolved. For example

```
async function reportMath() {
  await MathJax.typesetPromise();
  for (const eqn of MathJax.startup.document.math) {
    console.log(eqn.math);
  }
}
```

would define an asynchronous function `reportMath()` that typesets the page and then reports the original TeX for each expression, similarly to the previous code example.

As an alternative approach, it is also possible to hook into the promise chain used by *MathJax.typesetPromise()* and the promise-based conversion functions if you have actions that need to be coordinated with MathJax's typesetting. To do this use the command

`MathJax.`**whenReady**(*action*)

Arguments

- **action** (()=>any()) – A function to be performed when MathJax has finished any pending typesetting or conversion actions.

Returns

A promise that resolves when your action has completed.

and pass it a function that does the actions you want to have synchronized with MathJax's typesetting. This will perform your function when MathJax is finished with any pending typeset or conversion actions. It also returns a promise that resolves when your action is complete, just like the promise-based typeset functions do. If your action creates a promise, be sure you function returns that promise so that *MathJax.whenReady()* will wait for it to complete before its own promise is resolved.

Using this function, the previous examples could be implemented

```
MathJax.typesetPromise();
MathJax.whenReady(() => {
  for (const eqn of MathJax.startup.document.math) {
    console.log(eqn.math);
  }
});
```

The *MathJax.whenReady()* function is analogous to the `MathJax.Hub.Queue()` command in v2.

11.3 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*])

Arguments

- **start** (number()) – An optional number at which to start the equation numbering. The default is 1.

This can be used to start the equation numbering at a particular number, or reset it to the default starting number of 1.

11.4 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 Braille output, all the math can be updated to include the Braille strings that it generates. 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 removing is no longer on the page. You do this by using the following command:

MathJax.**typesetClear**([*elements*])

Arguments

- **elements** ((string|HTMLElement)[]()) – An optional array of DOM elements or CSS selector strings that restricts the typesetting to the contents of the specified container elements.

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 settings will 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 will be removed, or CSS selector strings for them, and MathJax will forget about the math that is within those containers, while remembering the rest of the math on the page. Note that you should call this function **before** you change the contents of the containers.

For example, if you have an element with id="has-math" that you have previously 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 something like:

```
MathJax.typesetClear([node]);
node.innerHTML = new_html;
MathJax.typesetPromise([node]).then(() => {
  // the new content has been typeset
});
```

The argument passed to *MathJax.typesetClear()* 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.typesetPromise();
```

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.

11.5 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 following command:

`MathJax.startup.document.`**`getMathItemsWithin`**`(`*elements*`)`

Arguments

- **`elements`** (`(string|HTMLElement)[]()`) – An array of DOM elements or CSS selector strings that restricts the typesetting to the contents of the specified container elements.

Return MathItem[]
The list of `MathItem` objects for the expressions within the specified containers.

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 containers. 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 replacement for the v2 `ElementJax` object, and *`MathJax.startup.document.getMathItemsWithin()`* performs a similar function to the v2 function `MathJax.Hub.getAllJax()`.

11.6 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 your 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.

CHAPTER

TWELVE

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 current versions of MathJax.

(add something about inserting the math into the document's math list)

12.1 Conversion Methods

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. Based on those input and output formats, you will get the corresponding functions from the list below:

`MathJax.`**`tex2chtml`**(*math*[, *options*])

`MathJax.`**`tex2chtmlPromise`**(*math*[, *options*])

`MathJax.`**`tex2svg`**(*math*[, *options*])

`MathJax.`**`tex2svgPromise`**(*math*[, *options*])

`MathJax.`**`tex2mml`**(*math*[, *options*])

`MathJax.`**`tex2mmlPromise`**(*math*[, *options*])

`MathJax.`**`mathml2chtml`**(*math*[, *options*])

`MathJax.`**`mathml2chtmlPromise`**(*math*[, *options*])

`MathJax.`**`mathml2svg`**(*math*[, *options*])

`MathJax.`**`mathml2svgPromise`**(*math*[, *options*])

`MathJax.`**`mathml2mml`**(*math*[, *options*])

`MathJax.`**`mathml2mmlPromise`**(*math*[, *options*])

`MathJax.`**`asciimath2chtml`**(*math*[, *options*])

`MathJax.`**`asciimath2chtmlPromise`**(*math*[, *options*])

`MathJax.`**`asciimath2svg`**(*math*[, *options*])

`MathJax.`**`asciimath2svgPromise`**(*math*[, *options*])

`MathJax.`**`asciimath2mml`**(*math*[, *options*])

MathJax.**asciimath2mmlPromise**(*math*[, *options*])

Arguments

- **math** (string()) – The TeX, serialized MathML, or AsciiMath string to be converted.
- **options** (OptionList()) – Described in the section below.

Returns

The DOM tree (for CHTML or SVG output), a serialized MathML string, or a promise that returns one of these and that is resolved when the result is ready.

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 methods above that involve both mathml and svg.

The functions with names containing chtml or svg produce DOM elements as the results of the conversion, with the promise version passing that to its then() function as an argument, and the non-promise versions returning them immediately. You can insert these DOM elements into the document directly, or you can use their outerHTML property to obtain their serialized string form. Note that you may need to run

```
MathJax.startup.document.reset();
MathJax.startup.document.updateDocument();
```

in order to update the CSS needed for the output, especially for CHTML output.

Warning

If you do insert the result of a conversion function into the page, note that it will not have the MathJax contextual menu attached, and it will not become part of the list of math expressions that MathJax knows about in the page. That means that it will not be updated if a user changes menu settings that would require the math to be rerendered (e.g., if the renderer is changed, or if assistive settings like whether to use Braille output are changed). For these reasons, it is better to insert the original math string into the page and use *MathJax.typeset()* or *MathJax.typesetPromise()* to typeset the contents of the DOM element containing the math rather than convert it by hand and then inserting the result into the page.

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 that instead. Alternatively, you can use MathJax.startup.adaptor.parse() to convert the string to an HTMLdocument or XMLDocument object. For example,

```
const mml = MathJax.tex2mml('\\sqrt{x^2+1}');
const math = MathJax.startup.adaptor.parse(mml).body.firstChild;
```

will set math to be the <math> node containing the MathML DOM tree from the TeX expression \sqrt{x^2+1}.

The functions ending in Promise perform the conversion asynchronously, and return promises, while the others operate synchronously and return the converted form immediately.

Warning

In version 4, the promise-based conversion functions wait for any previously pending typeset or conversion operations to complete before performing their own conversion. The version 3 documentation recommended using and setting *MathJax.startup.promise* to make sure typeset calls were serialized; if you included that code pattern in your v3 work-flow, you should remove it, as that is now being handled by the conversion functions internally.

Note that the synchronous functions only work if the math you typeset doesn't require MathJax to load any extensions or data files (e.g., TeX input that uses `\require` or macros that are autoloaded from an extension, or output that requires additional font data to be obtained). If a file needs to be loaded, MathJax with throw a `MathJax retry` error, which will prevent the conversion from completing. In that case, you should either switch to the promise-based versions of the conversion function you are using, or preload the needed component or data files. See the *The "MathJax Retry" Error* section for more details.

Warning

In MathJax v4, with the introduction of new fonts that include many more characters than the original MathJax TeX fonts did, the fonts have been broken into smaller pieces so that your readers don't have to download the entire font and its data for characters that may never be used. That means that typesetting mathematics may need to operate asynchronously even if the TeX *doesn't* include `\require` or any auto-loaded extensions, as the output itself could need extra font data files to be loaded. Thus in version 4, it is always best to use the promise-based commands.

12.2 Conversion Options

All of the functions listed above require an argument that is the math string to be converted (e.g., the serialized MathML string, the TeX or LaTeX string, or the AsciiMath string). Note that this is not a serialized HTML string with embedded math, but only a single math expression in one of the formats that MathJax understands. Note also that you should not include math delimiters like `$$...$$` or `\(...\)` as part of the string; it is just the mathematics itself.

You may 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.
- `scale`, a number giving a scaling factor to apply to the resulting conversion. Default is 1.
- `family`, a font family name to be used for `mtext` and `merror` elements when their fonts are set to be inherited (via the `mtextInheritFont` or `merrorInheritFont` configuration options).

For example,

```
const 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,

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

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

12.3 Obtaining the Output Metrics

Since the `em`, `ex`, `containerWidth`, and `family` properties 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 element's width), MathJax provides a method for obtaining these values from a given DOM element.

`MathJax.`**`getMetricsFor`**`(`*node*, *display*`)`

Arguments

- **node** (`HTMLElement()`) – The DOM node that is the container for the mathematics.
- **display** (`boolean()`) – True if the math is in display mode, false if not.

Returns

An object containing `em`, `ex`, `containerWidth`, `scale`, and `family` values for the container, together with the `display` value.

This takes a DOM element (`node`) and a boolean (`display`), indicating if the math is in display mode or not, and returns an object containing 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);
MathJax.startup.document.reset();
MathJax.startup.document.updateDocument();
```

in order to 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.typesetPromise()`* 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.

12.4 Obtaining the Output Styles

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 CommonHTML 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.

MathJax adds rules to these stylesheets dynamically, and one side-effect of this is that those styles are not part of the stylesheet element's text content, so won't be included if you call `textContent` or any of the other methods of obtaining the text of the stylesheet. For this reason, MathJax provides the command

`MathJax.startup.adaptor.`**`cssText`**`(`*stylesheet*`)`

that will give the complete text content of a stylesheet, including the dynamically added rules.

Note that, 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 `Mathjax.startup.output.clearCache()` command just before the *`MathJax.tex2chtml()`* call.

12.5 Creating Stand-Alone SVG Images

If you are using the SVG output jax to produce stand-alone SVG files, then you should set the `fontCache` value in the `svg` section of your MathJax configuration to be `local` or `none`. If set to `global`, then there will be a common global cache created for all the character paths used in the expressions you typeset. To clear that cache, use

`MathJax.startup.output.`**`clearFontCache`**`()`

With a local font cache, the paths are stored within the SVG element itself. There will still be some dependencies on CSS, however. You can use the following to insert the needed style definitions directly into the SVG image.

```
const svgCss = [
  'svg a{fill:blue;stroke:blue}',
  '[data-mml-node="merror"]>g{fill:red;stroke:red}',
  '[data-mml-node="merror"]>rect[data-background]{fill:yellow;stroke:none}',
  '[data-frame],[data-line]{stroke-width:70px;fill:none}',
  '.mjx-dashed{stroke-dasharray:140}',
  '.mjx-dotted{stroke-linecap:round;stroke-dasharray:0,140}',
  'use[data-c]{stroke-width:3px}'
].join('');
const xmlDeclaration = '<?xml version="1.0" encoding="UTF-8" standalone="no"?>';
const SVGXMLNS = 'http://www.w3.org/2000/svg';

async function getSvgImage(math, options = {}) {
  const adaptor = MathJax.startup.adaptor;
  const result = await MathJax.tex2svgPromise(math, options);
  const svg = adaptor.tags(result, 'svg')[0];
  const defs = adaptor.tags(svg, 'defs')[0] || adaptor.append(svg, adaptor.create('defs
↪'));
  adaptor.append(defs, adaptor.node('style', {}, [adaptor.text(svgCss)], SVGXMLNS));
  adaptor.removeAttribute(svg, 'role');
  adaptor.removeAttribute(svg, 'focusable');
  adaptor.removeAttribute(svg, 'aria-hidden');
  const g = adaptor.tags(svg, 'g')[0];
  adaptor.setAttribute(g, 'stroke', 'black');
  adaptor.setAttribute(g, 'fill', 'black');
  return xmlDeclaration + '\n' + adaptor.serializeXML(svg);
}
```

This defines a function `getSvgImage()` that takes a math string and returns a self-contained serialized SVG image of the math.

Note that in version 4, the MathJax contextual menu also includes a `SVG Image` option in the `Show Math As` and `Copy Math As` submenus that you can use to obtain the SVG image directly.

CHAPTER

THIRTEEN

DETECTING TYPESET ERRORS

MathJax provides you with several ways to manage errors that occur while processing your mathematical expressions. Several of these are listed below.

13.1 Handling TeX Errors

When the TeX input jax encounters a syntax error or other problem with the TeX code that it is typesetting, it usually replaces the TeX with an error message (in red on a yellow background) to inform you of the problem. The *noundefined* and *noerrors* extensions modify that behavior. The first prevents error messages when an undefined macro is used (it displays the undefined macro name in red as an indication of the problem), while the second prevents error messages entirely, and simply displays the original TeX code inside an outline box. Note that the *noundefined* extension is included in the *combined components*.

13.1.1 Listing TeX Parse Errors

If you wish to identify the TeX expressions that don't parse properly, there are several approaches that could be taken. First, you can provide a *formatError()* function in the `tex` section of your MathJax configuration.

formatError(*jax*, *error*)

Arguments

- **jax** (`TeX()`) – The TeX input jax that is processing the math.
- **error** (`Error()`) – The error object that contains the message indicating the problem with the TeX syntax.

Returns

The `MmlNode` object that should be used as the content of the `math` tag for this expression, usually an `merror` node.

You can use this function to track the errors in your TeX code. The `jax.latex` value is the TeX string that is being typeset, while `jax.parseOptions.mathItem` is the `MathItem` object for the math on the page. This contains pointers into the DOM where the original TeX code was, along with other information. See the MathItem definition for details about this object.

For example, you could use this to print information about the errors using

```
MathJax = {
  tex: {
    formatError(jax, error) {
      console.log(`TeX error in "${jax.latex}": ${error.message}`);
      return jax.formatError(error);
    }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

which logs the message and then performs the usual action for formatting the error. Alternatively, you could use

```
MathJax = {
  tex: {
    formatError(jax, error) {
      const factory = jax.parseOptions.nodeFactory;
      const text = factory.create('token', 'mtext', {}, 'Error!');
      return factory.create('node', 'merror', [mtext], {title: error.message});
    }
  }
}
```

to have just `Error!` show up as the error message, with the actual error message stored in the `title` attribute of the `merror` node so that a tooltip will pop up when you hover over the error message.

13.1.2 Listing All Math Errors

Another approach would be to look through the list of MathItems after the typesetting is complete and filter out the ones that include `merror` nodes.

```
const errorItems = Array.from(MathJax.startup.document.math).filter((item) => {
  const node = item.root?.childNodes?.[0]?.childNodes?.[0];
  return node && node.isKind('merror') && node.attributes.get('data-mjx-error
↪');
});
for (const item of errorItems) {
  console.log(`Error in "${item.math}": ` +
              item.root.childNodes[0].childNodes[0].attributes.get('data-mjx-
↪error'));
}
```

This turns the document's math list into an array and filters by a function that looks through each MathItem's MathML tree (its `root` property) to see if it's first top-level item is an `merror` with a `data-mjx-error` attribute. Note that the first child of the top-level `math` element in the `root` is the inferred `mrow` element, which is explicit in the MathJax MathML tree, so the first `.childNodes[0]` is getting that inferred `mrow`.

13.1.3 Reporting Undefined Macros

If you are interested in obtaining a list of the macros that are undefined on a page, here is one approach to doing that.

```
MathJax = {
  startup: {
    ready() {
      const {HandlerType, ConfigurationType} = MathJax._.input.tex.HandlerTypes;
      const {Configuration} = MathJax._.input.tex.Configuration;
      Configuration.create('record-undefined', {
        [ConfigurationType.FALLBACK]: {
          [HandlerType.MACRO]: (parser, name) => {
            console.log(`\\${name} undefined in "${parser.mathItem}"`);
            parser.Push(parser.create('token', 'mtext', {mathcolor: 'red'}, `\\${name}
```

(continues on next page)

(continued from previous page)

```
↪`));
          }
        }
      });
      MathJax.startup.defaultReady();
    }
  },
  tex: {
    packages: {
      '[+]': ['record-undefined'],
      '[-]': ['noundefined']
    }
  }
}
```

Here, we create a new TeX configuration that has a fallback handler for macros, meaning that it will be called whenever a macro is not defined. That handler logs the undefined macro and the TeX in which it occurred, and then inserts the macro name into the output in red, like the *noundefined* extension does. The `tex` block's `packages` array is modified by adding the new configuration and removing the `noundefined` extension that is part of the pre-defined combined configurations.

13.2 Handling MathML Errors

MathML can contain errors, such as the wrong number of child nodes, or improper nesting of nodes. MathJax can run verification tests on the MathML to check that it is properly formed, and to report problems when they occur. By default, MathJax will replace an incorrect node by an `merror` node that lists the name of the node in red on a yellow background, leaving the rest of the math untouched. If you hover over the node name, a tooltip will pop up listing the full error.

13.2.1 Verifying MathML

There are a number of checks that MathJax can perform to verify the structure of your MathML, and these can be controlled using configuration options for the MathML input jax. The options and their defaults are given below:

```
MathJax = {
  mathml: {
    verify: {
      checkArity: true,         // check that the number of child nodes is correct
      checkAttributes: false,   // check that attribute names are valid
      checkMathvariants: true,  // check that the mathvariant value is valid
      fullErrors: false,        // show complete errors or just the name of the errant␣
↪node
      fixMmultiscripts: true,   // add missing <none> elements in <mmultiscripts>
      fixMtables: true          // add missing <mrow> and <mtd> elements in <mtable>
    }
  }
}
```

You can identify these errors in the internal MathML tree stored in a MathItem's `root` property by looking for `merror` nodes with `data-mjx-message` attributes, which hold the full error message for the node. For example,

```
for (const mitem of MathJax.startup.document.math) {
  mitem.root.walkTree((node) => {
    if (node.isKind('merror') && node.attributes.get('data-mjx-message')) {
      console.log(`Error: "${node.attributes.get('data-mjx-message')}" in`, '\n', mitem.
↪math);
    }
  });
}
```

would report the MathML verification errors in all the math in the page.

See the *MathML Input Processor Options* section for more details on the verification configuration options.

13.2.2 MathML Compilation Errors

The processing of a MathML expression can lead to compilation errors, such as errors caused by text not enclosed in a token element tag, or the presence of nodes that are not MathML nodes. Such errors cause the entire MathML tree to be replaced by an `merror` node containing the error message describing the problem.

These errors can be trapped using the *compileError()* function described in the section below.

13.3 Trapping Compile and Typeset Errors

Sometimes compiling a TeX expression into the internal MathML representation, or processing a MathML tree, can lead to an error message "Math input error". Hovering over this message should cause a tooltip with a more detailed error message to appear.

You can trap such errors by specifying a *compileError()* function in the `options` section of your MathJax configuration.

compileError(*document*, *math*, *error*)

Arguments

- **document** (`MathDocument()`) – The MathDocument containing the math.
- **math** (`MathItem()`) – The MathItem representing the math that has failed to process.
- **error** (`Error()`) – The Error object containing the error message for the problem that occurred.

The default action is to call `document.compileError(math, error)()`, which sets `math.root` to a `math` node containing an `merror` whose content is `error.message`. You can override that and do your own processing. For example

```
MathJax = {
  options: {
    compileError(document, math, error) {
      console.log(`Error: "${error.message}" in`, '\n', math.math);
      document.compileError(math, error);
    }
  }
}
```

will print the error message and offending TeX or MathML string to the console, and then call the default `compileError()` function.

Similarly, it is possible that an error can occur during the process of typesetting the math (that is, the conversion of the internal MathML to the specified output format). These produce a "Math output error" message within the page; hovering over such a message will produce a tooltip that details the cause of the problem.

As with compilation errors, there is a function that traps such typesetting errors.

typesetError(*document*, *math*, *error*)

Arguments

- **document** (`MathDocument()`) – The MathDocument containing the math.
- **math** (`MathItem()`) – The MathItem representing the math that has failed to process.
- **error** (`Error()`) – The Error object containing the error message for the problem that occurred.

For example

```
MathJax = {
  options: {
    typesetError(document, math, error) {
      console.log(`Error: "${error.message}" in`, '\n', math.math);
      document.typesetError(math, error);
    }
  }
}
```

will print the error message and offending TeX or MathML string to the console, and then call the default `typesetError()` function.

CHAPTER

FOURTEEN

THE "MATHJAX RETRY" ERROR

MathJax has a large number of optional features, and not all of them are included when you load MathJax into a web page. If one of those features is needed by your code, MathJax will suspend its operations and attempt to load the needed extension for that feature. Because this process is asynchronous, MathJax must give up the CPU, wait for the needed file to load, and restart the typesetting after it has arrived.

This process is managed internally by MathJax setting up a promise for when the file is loaded, and throwing an error so that code higher up in the typesetting process can catch that error and know that it must wait for the promise to resolve before retrying the typesetting that was being performed. This is an error with the message `MathJax retry`.

The promise-based typesetting and conversion functions handle this retry error automatically, and incorporate waiting for the asynchronous file loading to complete into their own promises. The synchronous functions, however, can't do that, since the retry promise would make them asynchronous. If a retry is requested during the running of one of the synchronous functions, the retry error will not be caught, and you will likely get an error report in the browser console indicating an uncaught `MathJax retry` error. That indicates that you may need to rewrite your code to use the promise-based functions, instead, which means your code will have to handle asynchronous typesetting, and can't work synchronously as it stands.

If there is no promise-based version of the code you are running, then you may be able to use the following function to process the retry errors for you.

`mathjax.`**`handleRetriesFor`**`(`*function*`)`

Arguments

- **function** (`()=>any()`) – A function to run with retry errors being trapped. If one occurs, the function will be called again after the promise associated with the retry error's file loading has been resolved.

Return Promise

A promise that is resolved when the function argument completes without a retry.

From within a web page, you can obtain the `mathjax` variable via

```
const {mathjax} = MathJax._.mathjax;
```

For example, you might need to do something like the following:

```
const {mathjax} = MathJax._.mathjax;
MathJax.whenReady(mathjax.handleRetriesFor(async () => {
  const doc = MathJax.startup.doc;
  const dom = doc.convert('\\color{red}{x+y}');
  await doc.actionPromises();
  doc.clearPromises();
  return dom;
```

(continues on next page)

(continued from previous page)

```
})).then((node) => {
  document.body.append(node);
  MathJax.startup.document.reset();
  MathJax.startup.document.updateDocument();
});
```

The `convert()` call would normally throw a `MathJax retry` error when loading the *color* extension the first time it is used, but the `handleRetriesFor()` call traps that and handles it, eventually typesetting the expression once the *color* extension has been loaded. In addition, we wait for any promises that were created during the `convert()` call (e.g., the speech extension will add one that resolves when the speech has been applied to the result), and the promises are cleared. Finally, when all the promises are resolved, the `.then()` call runs, where the result is appended to the document, and the document CSS is updated to include any new CSS needed for the output.

Of course, it is better to insert the TeX code (with delimiters) directly into the page and call *MathJax.typesetPromise()* instead, but this is only meant as an example of how *mathjax.handleRetriesFor()* works. MathJax v4 also includes `MathJax.startup.document.convertPromise()` command that includes the *mathjax.handleRetriesFor()* already, or you could use the *MathJax.tex2chtmlPromise()* or *MathJax.tex2svgPromise()* methods, depending on the output format that you have available.

Some things that may initiate a `MathJax retry` error include:

- Using the `\require` macro in TeX code
- Using a macro that autoloads its definition (like `\color` or `\bbox`)
- Using some named entities in MathML code in the conversion functions
- Generating output for characters whose data must be loaded dynamically.
- Loading of localization files for speech generation.

If you are trying to use synchronous calls, any of these situations may lead to the `MathJax retry` error. If you are unable to move to the promise-based calls for some reason, then your only recourse is to load any of the needed extensions before typesetting or converting the math.

To do this, be sure to include any needed TeX extensions in the `load` array of the `loader` section of your MathJax configuration. To handle the entities in MathML, add the `[mml]/entities` extension to the `load` array.

You can load all the font data up front by setting the `loadAllFontFiles` option to `true` in the `startup` section of your MathJax configuration. This can cause *many* files to be loaded, however, so should be avoided if at all possible. It is much better to move to the promise-based calls to handle this situation. If you must use `loadAllFontFiles`, then you may want to pick a font with less character coverage, such as `mathjax-tex`, the original MathJax TeX fonts that doesn't have any dynamically loaded data, rather than the newer fonts for version 4, which have much higher coverage, and so would involve loading more files.

CHAPTER

FIFTEEN

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.

15.1 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.

15.1.1 Getting MathJax via npm

To include MathJax in your project, use the command

```
npm install mathjax@4
```

This will install MathJax in `node_modules/mathjax` subdirectory of your current directory.

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

```
npm install @mathjax/src@4
```

which installs MathJax in the `node_modules/@mathjax/src` subdirectory, with the webpacked component files in the `node_modules/@mathjax/src/bundle` directory. The Typescript source code files are in `node_modules/@mathjax/src/ts`, and pre-compiled versions of this are available in two formats: as ES modules in the `node_modules/@mathjax/src/mjs` directory, and as CommonJS modules in `node_modules/@mathjax/src/cjs`. See the *Getting Started with Node* section for more details about how to use the MathJax source code in your own javascript projects.

Note

Version 4 of MathJax has moved to using scoped npm packages. The version 3 package name `mathjax-full` is now `@mathjax/src`.

15.1.2 Getting MathJax via git

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

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

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

If you need access to the source code as well as the webpacked components, then instead 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. In this case, 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
pnpm install
pnpm -s build-all
cd ..
```

This will compile the typescript source files from the `@mathjax/src/ts` directory into javascript files in the `@mathjax/src/mjs` and `@mathjax/src/cjs` directories, and then will build the component files from `@mathjax/src/components/mjs` into the `@mathjax/src/bundle` directory.

Note

MathJax version 4 has switched to using `pnpm` rather than `npm`, so you will need to install that if you don't have it installed already, as the build scripts rely on it. To do so, use

```
npm install -g pnpm
```

If you don't want to build both `cjs` and `mjs` versions, then you can use

```
pnpm -s build
```

to build just the `mjs` versions, or

```
pnpm -s build-cjs
```

to build just the `cjs` versions.

Note

The directory structure and build process for MathJax version 4 has been significantly updated. See the Release notes for 4.0.0-beta.2 for a discussion of the new dual mjs/cjs structure.

15.2 Making 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 source distribution are not needed on the server. For example, the `@mathjax/src/ts` directory is typescript source code for MathJax, and this is compiled into the javascript files found in the `@mathjax/src/mjs` or `@mathjax/src/cjs` 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/src/bundle` directory using the data in the `@mathjax/src/components/mjs` directory.

It is the contents of the `@mathjax/src/bundle` directory that you want to make available on your server, as these are the files that are served from the CDNs that provide MathJax. If you installed the plain `mathjax@4` npm package, that is the set of files you will have obtained, as the `mathjax` package is just these bundled files.

You should move those files to a convenient location on your server. This might be a top-level directory called `mathjax`, for example, or something like `assets/mathjax` in your application directory.

15.3 Linking to Your Copy of MathJax

You can include MathJax in your web page by putting

```
<script defer src="path-to-MathJax/tex-chtml.js"></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 *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` directory at the top level of you server's web site and named it `mathjax`, you could use

```
<script defer src="/mathjax/tex-chtml.js"></script>
```

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

```
<!DOCTYPE html>
<html>
    <head>
        ...
        <script defer src="/mathjax/tex-chtml.js"></script>
    </head>
    <body>
        ...
    </body>
</html>
```

15.4 Obtaining the Needed Fonts

In version 3, there was only one font (`mathjax-tex`) and it was bundled with MathJax itself, so there when you installed MathJax, you also git that font. That is no longer the case with version 4, since there is a choice of fonts, and they are made available in separate packages. Installing MathJax via `npm` or `pnpm` will get you the default `mathjax-newcm` font, but if you plan to use a different font and have that served from your server, you will need to load its font package as well. E.g.,

```
pnpm install @mathjax/mathjax-stix2-font@4
```

to install the *mathjax-stix2* font.

You will need to move the `node_modules/@mathjax/mathjax-stix2-font` directory to a suitable location on your server, as you have the MathJax files themselves.

In order to use the font you have loaded, you will need to configure MathJax to tell it the font you need, and where the font files are located on your server. For example:

```
MathJax = {
  output: {
    font: 'mathjax-stix2',
    fontPath: '<path-to-mathjax-stix2-font>',
  }
};
```

where `<path-to-mathjax-stix2-font>` is the URL for where you have places the `@mathjax/mathjax-stix2-font` folder.

In this case, your page might look like

```
<!DOCTYPE html>
<html>
    <head>
        ...
        <script>
          MathJax = {
            output: {
              font: 'mathjax-stix2',
              fontPath: '/mathjax-stix2-font',
            }
          };
        </script>
        <script defer src="/mathjax/tex-chtml.js"></script>
    </head>
    <body>
        ...
    </body>
</html>
```

15.5 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 other browsers, and it affects how fonts are loaded with the `@font-face` CSS directive. MathJax's CommonHTML output mode uses this directive to load web-based math fonts into the web page when needed. These browsers' security policies, however, may 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 "\.(woff|woff2)$">
<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 "https://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.

Note that the CDNs that host MathJax already have these settings in place, so you can load fonts from them into your own pages without having to worry about these issues.

For web servers other than Apache, you will need to consult the server's documentation to determine how to specify the needed header line for fonts on your system.

15.6 Using MathJax Locally

You can use MathJax locally without a connection to the internet by following the basic outline above, and using `file://` URLs to access your local files. Note, however, that some browsers have additional cross-origin restrictions for `file://` URLs, and that may limit where you can place the MathJax files and font files.

In that case, you may need to run a local webserver for MathJax and its files. For example, if you have placed the `mathjax` and `mathjax-newcm-font` files in a directory called `assets`, then if do

```
cd assets
node serve --cors
```

and configure your page like

```
<!DOCTYPE html>
<html>
    <head>
        ...
        <script>
          MathJax = {
            output: {
              font: 'mathjax-stix2',
              fontPath: 'http://localhost:3000/mathjax-stix2-font',
            }
          };
```

(continues on next page)

(continued from previous page)

```
        </script>
        <script defer src="http://localhost:3000/mathjax/tex-chtml.js"></script>
    </head>
    <body>
        ...
    </body>
</html>
```

then you should be able to load this file using a `file://` URL and have MathJax served from the local server without the need for any access to the internet.

CHAPTER

SIXTEEN

EXAMPLES OF MATHJAX 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 within this documentation:

- *Configuring MathJax using an external script*
- *Configuring and loading MathJax using one local file*
- *Synchronizing with MathJax using promises*
- *Resetting TeX equation numbering*
- *Updating previously typeset content*
- *Looking up the math on the page*
- *Loading MathJax only on pages with math*
- *Configuring TeX Macros at Startup*
- *Running TeX Code at Startup*
- *Automatic section numbering*
- *Better display of Unicode full-width numbers*
- *Creating stand-alone SVG images*
- *Printing all the TeX code in the page*
- *Printing all the TeX errors in the page*
- *Printing all merror messages*
- *Reporting all undefined macros*
- *Reporting all MathML structural errors*
- *Trapping compilation and typesetting errors*
- *Defining both in-line and display-mode AsciiMath delimiters*
- *A replacement for the NativeMML output jax*
- *Building a custom component*
- *Building a custom extension*
- *Building a custom MathJax file*
- *A renderAction for tooltips*

- *A renderAction to collapse complex subexpressions*
- *A renderAction to use tags for math delimiters*
- *Allowing spaces in numbers*
- *Converting Unicode full-width characters to ASCII equivalents*
- *Converting Unicode numeric superscripts to TeX ones*
- *Converting SVG size from ex to px units*
- *An Autobold extension replacement*
- *Convert mathvariant text to Unicode equivalents*
- *Backward Compatibility for TeX input*
- *Locating MathJax v2 math script tags*

CHAPTER

SEVENTEEN

GETTING STARTED WITH NODE

There are several different ways to use MathJax in node applications, as described in the links below. The first is a simple way to get started experimenting with MathJax in node, but is not designed to be used in production applications, and can't be used in a browser setting.

Another way is based on the idea of *MathJax components*, which are the mechanism MathJax uses in browsers to break up MathJax into smaller pieces that can be loaded individually as needed. MathJax components can be used in node just as they are in the browser, as described in the second link below.

Loading MathJax components generally operates asynchronously, so if you are processing expressions that need additional components to be loaded dynamically, your code must take that into account via the use of promises. This can complicate your own code. To overcome this, you can pre-load any needed components, so that MathJax doesn't need to do so dynamically while processing expressions. The third link below illustrates that.

The final method is to call on the MathJax code modules directly, outside of the components framework. This gives you the most direct control over MathJax's functionality, but at the cost of not having the convenience that MathJax components offer. This approach is described in the last link below.

Before you start, you should obtain a copy of MathJax, as described in the section on *Acquiring the MathJax Code*. Then you can follow one of the links below to find out more about how to use MathJax in that way.

17.1 The Details

CHAPTER

EIGHTEEN

EXPERIMENTING WITH MATHJAX IN NODE

If you want to experiment with MathJax in node, that is not hard to do using the approach described below. Note that this approach can not be used in a browser, but only for server-side or command-line applications. If you plan to bundle your MathJax code for use in a larger browser-based application, you will need to use one of the other methods described in the *Getting Started with Node* section.

First *get a copy of the MathJax code library*. Here, we will assume you have used `npm` or `pnpm` to install the `@mathjax/src@4` package. You will need to change the `require()` or `import` statements accordingly in the examples below if you have loaded `mathjax@4` or obtained MathJax from the `MathJax-src` GitHub repository.

18.1 Your First MathJax Program

Create a file named `test-mathjax.mjs` containing the following:

```
import MathJax from "@mathjax/src";
await MathJax.init({loader: {load: ['input/tex']}});
const mml = (await MathJax.tex2mmlPromise('x+y'));
console.log(mml);
```

then run this file from the command line using

```
node test-mathjax.mjs
```

It should produce the output

```
<math xmlns="http://www.w3.org/1998/Math/MathML" data-latex="x+y" display="block">
  <mi data-latex="x">x</mi>
  <mo data-latex="+">+</mo>
  <mi data-latex="y">y</mi>
</math>
```

This is you first MathJax node program!

Note

You could perform the same function from a CommonJS file rather than an ES6 module by creating `test-mathjax.cjs` containing

```
const MathJax = require("@mathjax/src");
MathJax.init({
  loader: {load: ['input/tex']}
```

```
}).then(() => MathJax.tex2mmlPromise('x+y'))
  .then((mml) => console.log(mml));
```

then run this file using

```
node test-mathjax.cjs
```

The `init()` function above takes as its argument a MathJax configuration object just like the ones used to configure MathJax in a browser. (See the *MathJax Configuration Options* pages for more details.) It returns a promise that is resolved when MathJax has loaded the needed components and is ready to process mathematics, at which point the global `MathJax` variable will be set up for use.

In the program above, we use the `await` command to wait for that promise to resolve, and then wait for the *`MathJax.tex2mmlPromise()`* call to convert a TeX or LaTeX expression into the corresponding MathML tree. The result is then printed.

Once you have initialized MathJax, you should be able to use MathJax in much the same way as you would in a browser. Note, however, that stand-alone node applications don't have a browser DOM, so don't have a `window` or `document` variable. Because of this, MathJax in node doesn't produce DOM elements, but rather uses its own `liteDOM` replacement for the browser DOM. See *The DOM Adaptor* section for more details about how you interact with the `liteDOM`.

18.2 Loading MathJax from Source

The examples above load MathJax from the bundled versions in the `@mathjax/src/bundle` directory, which are the files that would be used if you obtained MathJax from a server in a web page viewed by a browser. It is possible to use MathJax from the source `.js` files instead, however; for example, if you are making changes to the MathJax source code and want to check it quickly without having to repack your whole project.

To do so, use either

```
import MathJax from '@mathjax/src/source';
```

or

```
const MathJax = require('@mathjax/src/source');
```

when loading MathJax.

CHAPTER

NINETEEN

USING MATHJAX COMPONENTS IN NODE

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

First *get a copy of the MathJax code library*. Here, we will assume you have used `npm` or `pnpm` to install the `@mathjax/src@4` package. You will need to change the `require()` or `import` statements accordingly in the examples below if you have loaded `mathjax@4` or obtained MathJax from the `MathJax-src` GitHub repository.

In MathJax, the loading of components is asynchronous, and so you may need to use promises or the `await` command to mediate the flow of your program, particularly program startup. Once MathJax's components are loaded, however, you can call the non-promise-based functions, but should use the promise-based ones if you want to support autoloading of extensions, the `\require` macro in TeX input, or the v4 fonts with larger character coverage.

Warning

In MathJax v4, with the introduction of new fonts that include many more characters than the original MathJax TeX fonts did, the fonts have been broken into smaller pieces so that your readers don't have to download the entire font and its data for characters that may never be used. That means that typesetting mathematics may need to operate asynchronously even if the TeX *doesn't* include `\require` or any auto-loaded extensions, as the output itself could need extra font data files to be loaded. Thus in version 4, it is always best to use the promise-based commands.

The *synchronous examples* show how to operate synchronously from the outset, if that is required.

19.1 Configuring and Loading Components in Node

As with MathJax in a browser, using MathJax components in a node application consists of two steps: configuring MathJax, and Loading a MathJax combined component to process the configuration. Just as in a browser, the configuration is specified in the global `MathJax` variable.

19.1.1 Configuration for Node vs. the Web

If you are using MathJax components as part of a larger application that will be bundled for use in a web browser, then your MathJax configuration should be the same as the configuration you would use if you were loading MathJax into the web page directly, with one exception: you may need to provide the URL where MathJax should load any extensions that are needed once MathJax is running. Normally, MathJax determines that URL based on the `src` attribute of the `script` tag that loaded it, but since MathJax is part of your larger application, that URL is probably not appropriate, so you will need to specify the correct one yourself.

This is done using the `mathjax` property of the `paths` section in the `loader` block of your MathJax configuration. For example,

```
global.MathJax = {
  loader: {
    paths: {
      mathjax: 'https://cdn.jsdelivr.net/npm/mathjax@4'
    }
  }
}
```

would set things up so that extensions would be loaded from the `jsDelivr` CDN. You could, of course, make the extensions available on your own server and set the URL to point to that.

If you are using MathJax components as part of a server-side or command-line application, you should set the `mathjax` path to `@mathjax/src/bundle` instead:

```
global.MathJax = {
  loader: {
    paths: {
      mathjax: '@mathjax/src/bundle'
    }
  }
}
```

so that MathJax will take additional components from the `bundle` directory.

Note

In version 4, the `bundle` directory replaces the `es5` directory from version 3.

For non-browser applications, there are two additional steps you need to take. First, you must tell MathJax to use `import()` or `require()` as the mechanism for loading external files, and second, you need to load a non-browser *DOM adaptor*. MathJax provides a light-weight DOM implementation (called *liteDOM*) that is sufficient for MathJax's needs without unnecessary overhead, so you probably want to use that. If you need a more full-featured DOM implementation, you can use another one, such as `jsdom` or `linkedom` (MathJax does provide adaptors for these). It is even possible to use puppeteer with headless Chrome in order to be able to access a full DOM implementation from node.

Both of these features are set in the `loader` block of your MathJax configuration object, as illustrated in the sections below.

19.1.2 Configuring MathJax for Use with `import`

Because the configuration must be in place before the MathJax component is loaded, if you are using `import` commands rather than `require()`, that means you either need to put the MathJax configuration into a separate file to be imported before MathJax itself, or you need to use the promise-based `import()` function to load MathJax.

So you could create a file called `mathjax-config.mjs` containing

```
global.MathJax = {
  loader: {
    paths: {mathjax: '@mathjax/src/bundle'},
    load: ['adaptors/liteDOM'],
    require: (file => import(file))
  },
  // additional configuration here
};
```

and then use

```
import './mathjax-config.js';
import '@mathjax/src/bundle/tex-chtml.js';
await MathJax.startup.promise;

// your code that uses MathJax here

MathJax.done();
```

to load the `tex-chtml` combined component with that configuration, and wait for MathJax to set itself up.

Note

In MathJax v4, the speech generation is performed in web-workers (in the browser) or worker-threads (in node applications), and once these are started, they will prevent the node application from ending if they are not shut down. So v4 includes the `MathJax.done()` function that terminates the workers, thus allowing the node program to end. You should call this when your program is ready to end so that it can shut down properly.

Alternatively, you could do

```
global.MathJax = {
  loader: {
    paths: {mathjax: '@mathjax/src/bundle'},
    load: ['adaptors/liteDOM'],
    require: (file => import(file))
  },
  // additional configuration here
};
await import('@mathjax/src/bundle/tex-chtml.js');
await MathJax.startup.promise;

// your code that uses MathJax here

MathJax.done();
```

to include the configuration in-line before loading the `tex-chtml` component.

Note

ES6 modules usually use `import`, and `require()` is not available, but it is possible to define `require()` if you are making a command-line or server-side application. MathJax provides a file that does that for you, so if you add

```
import '@mathjax/src/bundle/require.mjs';
```

to your code, you can then use `require()` as described in the following section.

19.1.3 Configuring the Speech Locale

The default speech language is English, and the default Braille code is Nemeth. You can use the `sre` block of the `options` section of your MathJax configuration to specify a different locale or Braille version, as illustrated below.

```
global.MathJax = {
  loader: {
    paths: {mathjax: '@mathjax/src/bundle'},
    load: ['adaptors/liteDOM'],
    require: (file) => import(file)
  },
  options: {
    sre: {
      locale: 'de'
    }
  }
  // additional configuration here
};

await import('@mathjax/src/bundle/tex-chtml.js');
await MathJax.startup.promise;

// your code that uses MathJax here

MathJax.done();
```

which configures MathJax to produce speech strings in German rather than English.

19.1.4 Configuring MathJax for Use with `require()`

To use MathJax components in a CommonJS module, first set up the MathJax configuration, and then `require()` the combined component you want to load. So you can do

```
MathJax = {
  loader: {
    paths: {mathjax: '@mathjax/src/bundle'},
    load: ['adaptors/liteDOM'],
    require: require
  },
  // additional configuration here
};
require('@mathjax/src/bundle/tex-chtml.js');
MathJax.startup.promise
  .then(() => {
    //your MathJax code here
  })
  .catch((err) => console.error(err.message))
  .then(() => MathJax.done());
```

to configure MathJax for use with the `tex-chtml` combined component, and then wait for MathJax to start up, perform your commands (with error trapping), and then shut down MathJax.

19.1.5 Loading Individual Components

If you are using MathJax components in a server-side or command-line application, the combined components that MathJax provides may include components that you don't need (such as the menu code and expression explorer). So you may want to configure MathJax explicitly to use only the components that you need. You do this by listing the needed components in the `load` array of the `loader` section of the MathJx configuration, and then load the `startup.js` module rather than a combined component.

For example,

```
global.MathJax = {
  loader: {
    paths: {mathjax: '@mathjax/src/bundle'},
    load: ['input/tex', 'output/svg', 'adaptors/liteDOM'],
    require: (file => import(file)),
  },
  output: {font: 'mathjax-newcm'}
}
await import('@mathjax/src/bundle/startup.js');
await MathJax.startup.promise;
```

would load only the TeX input jax and the SVG output jax, along with the `liteDOM` adaptor, but without loading the menu code, the assistive tools, or any other components. Because the `input/tex` component includes the *require* and *autoload* extensions, the TeX that you process could still load TeX extensions that are needed.

Because the `output/svg` component does not include a font, you need to configure that separately in the `output` section of the configuration, as shown.

19.1.6 Loading MathJax Components from Source

The examples above all load the webpacked versions of MathJax's components. It is possible to load the files from the source `.js` files in the `mjs` or `cjs` directories, which may be useful if you are modifying the MathJax source files and want to test your changes without having to repack all the components.

To do this, you should set the `source` mapping in the `loader` section of the MathJax configuration, and then load the combined component from its source file in the `components` directory rather than the `bundle` directory. The `source.js` file in `components/mjs` or `components/cjs` directory contains the mapping of component names to their source definitions, and you can use that to set the `source` field of your MathJax configuration.

You can obtain the `source.js` file using `@mathjax/src/components/js/source.js`, and it will select the `mjs` or `cjs` directory depending on whether you use `import` or `require()` to load it. So for use in ES6 modules, you can do

```
import {source} from '@mathjax/src/components/js/source.js';
import '@mathjax/src/bundle/require.mjs';  // needed by speech-rule engine

global.MathJax = {
  loader: {
    paths: {mathjax: '@mathjax/src/bundle'},
    load: ['adaptors/liteDOM'],
    require: (file => import(file)),
    source: source
  }
  // additional configuration here
}
await import(source['tex-chtml']);
await MathJax.startup.promise;

// your code that uses MathJax here

MathJax.done();
```

while for CommonJS modules, you can do

```
const {source} = require('@mathjax/src/components/js/source.js');

MathJax = {
  loader: {
    paths: {mathjax: '@mathjax/src/bundle'},
    load: ['adaptors/liteDOM'],
    require: require,
    source: source
  }
  // additional configuration here
}
require(source['tex-chtml']);
MathJax.startup.promise
  .then(() => {
    //your MathJax code here
  })
  .catch((err) => console.error(err.message))
  .then(() => MathJax.done());
```

19.2 Calling MathJax from Components

Once you have loaded a combined component file (or the `startup` component), you can use the normal MathJax commands to typeset mathematics. For example, in a browser application, you can call *`MathJax.typesetPromise()`* to typeset the page.

For a command-line application, you could do

```
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking

function typeset(math, display = true) {
  return MathJax.tex2svgPromise(math, {
    display: display,
    em: EM,
    ex: EX,
    containerWidth: WIDTH
  }).then((node) => {
     const adaptor = MathJax.startup.adaptor;
     return(adaptor.serializeXML(adaptor.tags(node, 'svg')[0]));
  }).catch(err => console.error(err));
}
```

to define a `typeset()` command that takes a TeX string and an optional boolean that specifies whether the typesetting should be in display mode or in-line mode and returns a promise that is resolved when the typesetting is complete (while handling any waiting that had to be done to load extensions, fonts, etc.).

The `typeset()` promise returns the serialized SVG output, so that you could do

```
const svg = await typeset('\\sqrt{1+x^2}');
```

to get the SVG output. See the *Creating Stand-Alone SVG Images* section for an example of generating SVG images that handles the CSS needed by some expressions in MathJax.

19.3 Examples of Components in Node

The following combines some of the ideas described above into a single, complete example of a command-line tool that takes three arguments: a TeX string to typeset, the language locale to use, and the Braille format to use. The last two are optional, and default to `en` and `nemeth`.

```
global.MathJax = {
  loader: {
    paths: {mathjax: '@mathjax/src/bundle'},
    load: ['adaptors/liteDOM'],
    require: (file) => import(file)
  },
  options: {
    sre: {
      locale: process.argv[3] || 'en',
      braille: process.argv[4] || 'nemeth'
    }
  },
  output: {
    linebreaks: {
      inline: false,
    }
  },
  // additional configuration here
};

await import('@mathjax/src/bundle/tex-svg.js');
await MathJax.startup.promise;

const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking

function typeset(math, display = true) {
  return MathJax.tex2svgPromise(math, {
    display: display,
    em: EM,
    ex: EX,
    containerWidth: WIDTH
  }).then((node) => {
    const adaptor = MathJax.startup.adaptor;
    return(adaptor.serializeXML(adaptor.tags(node, 'svg')[0]));
  }).catch(err => console.error(err));
}

const math = process.argv[2] || '';
const svg = await typeset(math);
console.log(svg);

MathJax.done();
```

See the *Creating Stand-Alone SVG Images* section for an example of generating SVG images that handles the CSS needed by some expressions in MathJax.

See the MathJax node demos for more examples of how to use MathJax from a node application. In particular, see the component-based examples for illustrations of how to configure and load MathJax components.

CHAPTER

TWENTY

USING COMPONENTS SYNCHRONOUSLY

MathJax components are designed to operate in a browser environment, where loading components is inherently asynchronous as the browser has to wait for files to be downloaded over the network. This means that using MathJax's promise-based typesetting and conversion functions is generally required, and so the rest of your code needs to be designed to handle the promises that are part of MathJax. That can complicate your code, and your work with other frameworks may require you to operate synchronously.

In a *node* application, you can load components individually yourself via node's `import` or `require()` commands, rather than relying on MathJax's *loader*, which operates asynchronously. With a little care, this can give 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 handle loading dependencies yourself, and 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.

Note, however, that this means you will not be able to use the `\require` macro in TeX expressions, and that TeX extensions will not be able to be auto-loaded, as those actions are asynchronous.

Warning

In MathJax v4, with the introduction of new fonts that include many more characters than the original MathJax TeX fonts did, the fonts have been broken into smaller pieces so that your readers don't have to download the entire font and its data for characters that may never be used. That means that typesetting mathematics may need to operate asynchronously even if the TeX *doesn't* include `\require` or any auto-loaded extensions, as the output itself could need extra font data files to be loaded. Thus in version 4, it is always best to use the promise-based commands, when possible.

The examples below show how to pre-load the needed font data so that you can still work synchronously even with the larger v4 fonts.

20.1 The Basics of Pre-Loading Extensions

First, *get a copy of the MathJax code library*. Here, we will assume you have used `npm` or `pnpm` to install the `@mathjax/src@4` package. You will need to use `@mathjax/src`, not just `mathjax`, since the examples given here rely on access to the component definitions and other source code that is not part of the `mathjax` package, which includes only the bundled files.

The examples linked below load the needed MathJax components my hand, using the source files in MathJax's `components` directories (the files used to create the bundled components found in the `bundle` directory). The examples

use the path `@mathjax/src/components/js` to access the component definitions, as MathJax's `package.json` file is set up to map this to the `components/mjs` directory when used in an `import` command, and to `components/cjs` when used in a `require()` command, so the same address will give you the proper ES or CommonJS module for the mechanism you are using to load it.

All the examples begin with the following lines:

```
import {MathJax} from '@mathjax/src/js/components/global.js';
import {insert} from '@mathjax/src/js/util/Options.js';
import '@mathjax/src/js/components/startup.js';
import '@mathjax/src/components/js/core/core.js';
import '@mathjax/src/components/js/adaptors/liteDOM/liteDOM.js';
```

(or the equivalent using `require()`). The first line sets up the global `MathJax` variable, handling any existing `MathJax` value as a MathJax configuration (in case the examples below are included in a larger application that sets up its own configuration). The second line obtains a utility for inserting values into the MathJax configuration, overriding existing values, if any. The third line loads the *startup* component, which will be called later to instantiate the needed MathJax objects and create the various typeset and conversion functions. The fourth line loads the core component, and the fifth loads the *adaptors/liteDOM* component, which implements a limited DOM for use in node. (You would not load this component if you were using this technique to make a browser-based application, and there are other alternative adaptors that could be used if you need a more robust DOM implementation.)

Next we load the TeX components that we plan to use:

```
import '@mathjax/src/components/js/input/tex-base/tex-base.js';
import '@mathjax/src/components/js/input/tex/extensions/ams/ams.js';
import '@mathjax/src/components/js/input/tex/extensions/newcommand/newcommand.js';
import '@mathjax/src/components/js/input/tex/extensions/color/color.js';
```

In this case, we use the *tex-base* component, which only includes the *base* configuration, whereas the *input/tex* component includes the *require* and the *autoload* components, which we can't support synchronously.

We also load the *ams*, *newcommand*, and *color* components. These are just for illustration; you can include whatever components you need for the expressions you will be processing.

Next, we load the CommonHTML output jax

```
import '@mathjax/src/components/js/output/chtml/chtml.js';
```

though you could use the SVG output jax if you prefer.

Now that everything is loaded (though some of the examples load additional items), we configure the TeX input jax to use the pre-loaded extensions:

```
insert(MathJax.config, {
  tex: {
    packages: {'[+]': ['ams', 'newcommand', 'color']}
  }
}, false);
```

This uses the `insert()` function that we loaded earlier.

Then we start up MathJax:

```
MathJax.config.startup.ready();
```

and finally process some math and print the results:

```
const math = process.argv[2] || '';
const adaptor = MathJax.startup.adaptor;
console.log(adaptor.outerHTML(MathJax.tex2chtml(math)));
```

Here, we take the math from the command line arguments, but you will likely obtain the math to be processed from elsewhere in your code. You may want to provide a `typeset()` function that encapsulates the code to do the typesetting.

This is the outline that is illustrated in the examples below. They include variations on this theme that show how to handle fonts synchronously in several different ways. There is also an example that handles speech generation, though that requires one asynchronous step, and only creates speech for the top-level element of the resulting expression.

20.2 Using the MathJax-TeX font

This example uses the `mathjax-tex` font, which is the original font-set used by MathJax v2 and v3. Because this font has limited character coverage it is *not* broken into multiple pieces, so you don't have to worry about dynamically loaded font data, so preloading the TeX extensions is all you have to worry about in order to be able to process math synchronously.

The lines that are required for this that differ from the outline given above are highlighted below.

Note that you will need to use `npm` or `pnpm` to install the `@mathjax/mathjax-tex-font` package in order to use the `mathjax-tex` font.

```
import {MathJax} from '@mathjax/src/js/components/global.js';
import {insert} from '@mathjax/src/js/util/Options.js';
import '@mathjax/src/js/components/startup.js';
import '@mathjax/src/components/js/core/core.js';
import '@mathjax/src/components/js/adaptors/liteDOM/liteDOM.js';

//
// Load the TeX components that we want to use
//
import '@mathjax/src/components/js/input/tex-base/tex-base.js';
import '@mathjax/src/components/js/input/tex/extensions/ams/ams.js';
import '@mathjax/src/components/js/input/tex/extensions/newcommand/newcommand.js';
import '@mathjax/src/components/js/input/tex/extensions/color/color.js';

//
// Load the output jax
//
import '@mathjax/src/components/js/output/chtml/chtml.js';

//
// Load the mathjax-tex font
//
import {MathJaxTexFont} from '@mathjax/mathjax-tex-font/js/chtml.js';

//
// Add the pre-loaded TeX extensions here, and specify the font
//
insert(MathJax.config, {
  tex: {
```

(continues on next page)

(continued from previous page)

```
    packages: {'[+]': ['ams', 'newcommand', 'color']}
  },
  chtml: {
    fontData: MathJaxTexFont
  }
}, false);

//
// Start up MathJax
//
MathJax.config.startup.ready();

//
// Convert some math synchronously
//
const math = process.argv[2] || '';
const adaptor = MathJax.startup.adaptor;
console.log(adaptor.outerHTML(MathJax.tex2chtml(math)));
```

20.3 Using the MathJax-NewCM font

This example uses the default font for MathJax v4, the `mathjax-newcm` font, which is based on the New Computer Modern font. Because `mathjax-newcm` has extensive character coverage, it is broken into a number of separate files that are loaded dynamically when needed. That means producing output that uses `mathjax-newcm` can be asynchronous, as some character data may need to be loaded from additional files.

To use this font synchronously, you need to pre-load the font data files for the characters that you expect to need. The example below loads the calligraphic characters, but you could include additional `import` commands to load other ranges of characters. These files can be found in the `node_modules/@mathjax/mathjax-tex-font/mjs/chtml/dynamic` directory.

The `mathjax-tex-font` package should be installed automatically when you install the `@mathjax/src` npm package, so you shouldn't need to install it by hand, as you did for the `mathjax-tex` font in the previous example.

The lines that are required for this that differ from the outline given above are highlighted below.

```
import {MathJax} from '@mathjax/src/js/components/global.js';
import {insert} from '@mathjax/src/js/util/Options.js';
import '@mathjax/src/js/components/startup.js';
import '@mathjax/src/components/js/core/core.js';
import '@mathjax/src/components/js/adaptors/liteDOM/liteDOM.js';

//
// Load the TeX components that we want to use
//
import '@mathjax/src/components/js/input/tex-base/tex-base.js';
import '@mathjax/src/components/js/input/tex/extensions/ams/ams.js';
import '@mathjax/src/components/js/input/tex/extensions/newcommand/newcommand.js';
import '@mathjax/src/components/js/input/tex/extensions/color/color.js';

//
```

(continues on next page)

(continued from previous page)

```
16 // Load the output jax
17 //
18 import '@mathjax/src/components/js/output/chtml/chtml.js';
19
20 //
21 // Load the font to use, and any dynamic font files
22 //
23 import {MathJaxNewcmFont} from '@mathjax/mathjax-newcm-font/js/chtml.js';
24 import '@mathjax/mathjax-newcm-font/js/chtml/dynamic/calligraphic.js';
25   //
26   // ... load any additional ones here, and add them to the array below.
27   //
28 const fontPreloads = ['calligraphic'];
29
30 //
31 // Add the pre-loaded TeX extensions here
32 //
33 insert(MathJax.config, {
34   tex: {
35     packages: {'[+]': ['ams', 'newcommand', 'color']}
36   },
37   chtml: {
38     fontData: MathJaxNewcmFont
39   }
40 }, false);
41
42 //
43 // Start up MathJax
44 //
45 MathJax.config.startup.ready();
46
47 //
48 // Activate the dynamic font files
49 //
50 const font = MathJax.startup.document.outputJax.font;
51 const dynamic = MathJaxNewcmFont.dynamicFiles;
52 fontPreloads.forEach(name => dynamic[name].setup(font));
53
54 //
55 // Convert some math synchronously
56 //
57 const math = process.argv[2] || '';
58 const adaptor = MathJax.startup.adaptor;
59 console.log(adaptor.outerHTML(MathJax.tex2chtml(math)));
```

This approach could also be used to handle any of the MathJax fonts available for v4 by first installing the needed font, and changing lines 23 to 28, line 38, and line 51 to refer to the correct font.

Technically, lines 37 to 39 are not needed, since the `mathjax-newcm` font is the default for MathJax v4, but these lines show how to configure MathJax for any font.

20.4 A CommonJS Example

This example illustrates using `require()` rather than `import` in a CommonJS module. Because `require()` is synchronous, this makes it possible to load the dynamic font files in a loop rather than having to list them individually as we do above. MathJax provides a `loadDynamicFilesSync()` method for doing so, but it requires that you specify a mechanism for loading the files. This is usually an asynchronous method, but since we are using `require()`, we indicate that it is actually synchronous.

The changes needed for this are highlighted below. Of course, all the `import` commands have been changed to equivalent `require()` commands throughout.

```
const {MathJax, combineDefaults} = require('@mathjax/src/js/components/global.js');
const {insert} = require('@mathjax/src/js/util/Options.js');
require('@mathjax/src/js/components/startup.js');
require('@mathjax/src/components/js/core/core.js');
require('@mathjax/src/components/js/adaptors/liteDOM/liteDOM.js');

//
// Needed for asyncLoad below
//
const {mathjax} = require('@mathjax/src/js/mathjax.js');
const {Package} = require('@mathjax/src/js/components/package.js');

//
// Load the TeX components that we plan to use
//
require('@mathjax/src/components/js/input/tex-base/tex-base.js');
require('@mathjax/src/components/js/input/tex/extensions/ams/ams.js');
require('@mathjax/src/components/js/input/tex/extensions/newcommand/newcommand.js');
require('@mathjax/src/components/js/input/tex/extensions/color/color.js');

//
// Load the output jax
//
require('@mathjax/src/components/js/output/chtml/chtml.js');

//
// Set the font path and add the pre-loaded TeX extensions here
//
insert(MathJax.config, {
  loader: {
    paths: {
      'mathjax-newcm': '@mathjax/mathjax-newcm-font/js'
    }
  },
  tex: {
    packages: {'[+]': ['ams', 'newcommand', 'color']}
  }
}, false);

//
// Start up MathJax
//
MathJax.config.startup.ready();
```

(continues on next page)

(continued from previous page)

```
44
45 //
46 // Load the font dynamic files
47 //
48 mathjax.asyncLoad = (file) => require(Package.resolvePath(file));
49 mathjax.asyncIsSynchronous = true;
50 MathJax.startup.document.outputJax.font.loadDynamicFilesSync();
51
52 //
53 // Convert some math synchronously
54 //
55 const math = process.argv[2] || ''
56 const adaptor = MathJax.startup.adaptor;
57 console.log(adaptor.outerHTML(MathJax.tex2chtml(math)));
```

This example loads **all** the dynamic font files, so you don't have to know which ones you will need. Note, however, that that can be a large number of files, with a large amount of data, much of which likely will never be used. This can increase the startup time for your application, so you may want to use the technique of individually loading only the files you actually need.

In an ES module, one could use

```
mathjax.asyncLoad = (file) => import(Package.resolvePath(file));
await MathJax.startup.document.outputJax.font.loadDynamicFiles();
```

to load all the font files, but that is asynchronous, so you need to use `await` to wait for the command to complete. Be aware that that means other parts of your code could run before that is completed, so you will need to be careful not to call MathJax commands until after the loading is complete.

20.5 Loading All Font Data Synchronously

In the previous example, we took advantage of `require()` to make the `loadDynamicFilesSync()` method available to load all the font data before processing any math. Although ES modules (using `import` and `export`) don't have a `require()` command, it is possible to define one that can be used to load CommonJS modules. MathJax provides a file that does that for you, so you don't need to know the details of how that works. This is done in line 12 below, which is the only new line added to the CommonJS example above.

In order for this to work, you need to use the CommonJS versions of all the MathJax modules. That is done by changing the `js` directory name to `cjs` everywhere it occurs in the `import` commands and the loader path definition.

```
1  import {MathJax} from '@mathjax/src/cjs/components/global.js';
2  import {insert} from '@mathjax/src/cjs/util/Options.js';
3  import '@mathjax/src/cjs/components/startup.js';
4  import '@mathjax/src/components/cjs/core/core.js';
5  import '@mathjax/src/components/cjs/adaptors/liteDOM/liteDOM.js';
6
7  //
8  // Needed for asyncLoad below
9  //
10 import {mathjax} from '@mathjax/src/cjs/mathjax.js';
11 import {Package} from '@mathjax/src/cjs/components/package.js';
```

(continues on next page)

(continued from previous page)

```
import '@mathjax/src/components/require.mjs';

//
// Load the TeX components that we want to use
//
import '@mathjax/src/components/cjs/input/tex-base/tex-base.js';
import '@mathjax/src/components/cjs/input/tex/extensions/ams/ams.js';
import '@mathjax/src/components/cjs/input/tex/extensions/newcommand/newcommand.js';
import '@mathjax/src/components/cjs/input/tex/extensions/color/color.js';

//
// Load the output jax
//
import '@mathjax/src/components/cjs/output/chtml/chtml.js';

//
// Set the font path and add the pre-loaded TeX extensions here
//
insert(MathJax.config, {
  loader: {
    paths: {
      'mathjax-newcm': '@mathjax/mathjax-newcm-font/cjs'
    }
  },
  tex: {
    packages: {'[+]': ['ams', 'newcommand', 'color']}
  }
}, false);

//
// Start up MathJax
//
MathJax.config.startup.ready();

//
// Load the font dynamic files
//
mathjax.asyncLoad = (file) => require(Package.resolvePath(file));
mathjax.asyncIsSynchronous = true;
MathJax.startup.document.outputJax.font.loadDynamicFilesSync();

//
// Convert some math synchronously
//
const math = process.argv[2] || '';
const adaptor = MathJax.startup.adaptor;
console.log(adaptor.outerHTML(MathJax.tex2chtml(math)));
```

If you don't use the `cjs` paths explicitly, the `import` commands will load the ES Modules, while the `asyncLoad` command, using `require()`, will load the CommonJS modules. That will cause all of MathJax to be loaded again from the `cjs` directories, and you will have two copies of everything. Then `loadDynamicFilesSync()` will load the dynamic files into the `cjs` copies, while your typesetting will be performed by the `mjs` versions, which don't have the dynamic files loaded, and that will lead to a `MathJax retry` error when any dynamic file is needed. That is, your

loading of the dynamic files will have no effect, because they went into the wrong copy of MathJax.

20.6 Synchronous Typesetting with Speech

This example builds on the *Using the MathJax-NewCM font* example by adding the needed code for generating speech output for the resulting expressions. This is accomplished by include the speech-rule-engine (SRE) code. Unfortunately, the startup code for SRE is inherently asynchronous, so you do need to handle one promise during initialization and wait for that before performing any typesetting operations, but once that one promise is resolved, you can work synchronously from there.

Since SRE uses `require()` to load its dependencies when it is used in node, we include the `components/require.mjs` file to make that available from our ES module.

Lines 7 through 13 are the import commands needed to load SRE and other needed modules.

Lines 47 through 68 define a function that adds a speech string to the root node of the internal MathML tree, and then adds a `renderAction` to the document options that performs the `addSpeech()` action. The `aria-label` attribute will be included in the DOM elements generated by MathJax later in the conversion step.

Lines 70 through 75 give the asynchronous code that must be used to get SRE up and running before typesetting can be done synchronously. This waits for SRE to set itself up, passing it the locale and modality needed for the language specified as the second command-line argument to this script, then waits for SRE to be ready before startup up MathJax.

You may need to use `npm` or `pnpm` to install the `speech-rule-engine` npm module, if it isn't already installed.

```
1  import {MathJax} from '@mathjax/src/js/components/global.js';
2  import {insert} from '@mathjax/src/js/util/Options.js';
3  import '@mathjax/src/js/components/startup.js';
4  import '@mathjax/src/components/js/core/core.js';
5  import '@mathjax/src/components/js/adaptors/liteDOM/liteDOM.js';
6
7  //
8  // Load code for SRE
9  //
10 import '@mathjax/src/components/require.mjs';
11 import '@mathjax/src/components/js/a11y/semantic-enrich/semantic-enrich.js';
12 import {setupEngine, engineReady, toSpeech} from 'speech-rule-engine/js/common/system.js
   ↪';
13 import {STATE} from '@mathjax/src/js/core/MathItem.js';
14
15 //
16 // Load the TeX components that we want to use
17 //
18 import '@mathjax/src/components/js/input/tex-base/tex-base.js';
19 import '@mathjax/src/components/js/input/tex/extensions/ams/ams.js';
20 import '@mathjax/src/components/js/input/tex/extensions/newcommand/newcommand.js';
21 import '@mathjax/src/components/js/input/tex/extensions/color/color.js';
22
23 //
24 // Load the output jax
25 //
26 import '@mathjax/src/components/js/output/chtml/chtml.js';
27
28 //
```

(continues on next page)

(continued from previous page)

```
// Load the font to use, and any dynamic font files
//
import {MathJaxNewcmFont} from '@mathjax/mathjax-newcm-font/js/chtml.js';
import '@mathjax/mathjax-newcm-font/js/chtml/dynamic/calligraphic.js';
  //
  // ... load any additional ones here, and add them to the array below.
  //
const fontPreloads = ['calligraphic'];

//
// Add the pre-loaded TeX extensions here
//
insert(MathJax.config, {
  tex: {
    packages: {'[+]': ['ams', 'newcommand', 'color']}
  }
}, false);

//
// Add a speech string to the root math element
//
function addSpeech(item) {
  const speech = toSpeech(MathJax.startup.toMML(item.root));
  item.root.attributes.set('aria-label', speech);
}

//
// Add a render action that computes the speech
//
insert(MathJax.config, {
  options: {
    renderActions: {
      addSpeech: [
        STATE.COMPILED + 10,
        (doc) => {for (const item of doc.math) addSpeech(item)},
        (item, doc) => addSpeech(item)
      ]
    }
  }
}, false);

//
// Start up SRE
//
const locale = process.argv[3] || 'en';
const modality = locale === 'nemeth' || locale === 'euro' ? 'braille' : 'speech';
await setupEngine({modality, locale}).then(() => engineReady());

//
// Start up MathJax
//
MathJax.config.startup.ready();
```

(continues on next page)

(continued from previous page)

```
81
82  //
83  // Activate the dynamic font files
84  //
85  const font = MathJax.startup.document.outputJax.font;
86  const dynamic = MathJaxNewcmFont.dynamicFiles;
87  fontPreloads.forEach(name => dynamic[name].setup(font));
88
89  //
90  // Convert some math synchronously
91  //
92  const math = process.argv[2] || '';
93  const adaptor = MathJax.startup.adaptor;
94  console.log(adaptor.outerHTML(MathJax.tex2chtml(math)));
```

If you don't want to, or can't, use `await`, you can add one more `then()` clause whose function starts up the main code for your application instead.

More examples are available in the MathJax node demos for using 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.

CHAPTER

TWENTYONE

LINKING TO MATHJAX DIRECTLY IN NODE

The previous sections use the *MathJax components* framework to manage most of the details of setting up and using MathJax. That framework uses a global `MathJax` variable to configure MathJax, and to store the functions for typesetting and converting math in web pages.

It is possible, however, to bypass the components layer and link to the MathJax modules directly. This provides the lowest-level access to the MathJax code, and while it is more complicated than using components, it gives you the greatest control over the details of MathJax. In this approach, you trade off ease of configuration and use for more direct and granular command over MathJax's operations.

When you import MathJax code directly, the MathJax loader and startup modules are not used, and since those underlie the dynamic loading of MathJax code on the fly, that ability is more restricted in this setting. Some modules rely on that ability, however; in particular the *require* and *autoload* TeX extensions, and the MathJax menu code, all depend on the component infrastructure, and so can not be used when importing the MathJax modules directly.

With the direct approach, you must load all the MathJax code that you will be using explicitly, and you will need to instantiate and configure the MathJax objects (like the input and output jax, and the MathDocument object) by hand, as the *startup* module that performs those duties within the components framework, along with the `MathJax` configuration variable, are not used when you load MathJax modules directly.

Note

With a little care, it is possible to mix components and direct loading of modules. This is illustrated in the *Using Components Synchronously* section. Although that shows how to use MathJax synchronously, those techniques can be used for asynchronous processing as well. This is also illustrated in the *Mixing Components and Direct Linking* example below.

Finally, MathJax v4 introduced new fonts that include many more characters than the original MathJax TeX fonts, and these have been broken into smaller pieces so that, in web pages, your readers don't have to download the entire font and its data for characters that may never be used. Font ranges are downloaded dynamically when needed, but when you use direct access to MathJax, rather than the components framework, that dynamic loading takes a bit more work. You either have to preload the ranges that you will need, or make provisions for loading the ranges yourself when they are needed. Both these approaches are illustrated in the examples below.

21.1 The Basics of Linking Directly to MathJax

First, *get a copy of the MathJax code library*. Here, we will assume you have used `npm` or `pnpm` to install the `@mathjax/src@4` package. You will need to use `@mathjax/src`, not just `mathjax`, since the latter only includes the bundled component files, not the individual MathJax modules that you will be importing directly.

The MathJax source code for v3 and earlier consisted of ES5 javascript in CommonJS modules. As of version 4, MathJax is compiled into both ES5 CommonJS modules and the more modern ES6 Modules. These are stored in the `cjs` and `mjs` directories, respectively, of the `@mathjax/src` node package. The MathJax `package.json` file is set up so that references to `@mathjax/src/js` will access the `cjs` directory when used in a `require()` command, and the `mjs` directory when used in an `import` command. The examples below will use `import` commands, but you can change them to the corresponding `require()` commands without altering the file paths.

The original source code for MathJax is in Typescript, which is a form of javascript that has additional information about the types of data stored in variables, used for function arguments and return values, and so on. Those Typescript files are compiled into the `cjs` and `mjs` directories when MathJax is built. The `ts` directory holds the Typescript files, and those contain comments describing the functions and objects they include. You can refer to them to see what can be imported from each of the compiled files in the `cjs` and `mjs` directories.

Most of the examples below begin with `import` commands like the following:

```
import {mathjax} from '@mathjax/src/js/mathjax.js';
import {TeX} from '@mathjax/src/js/input/tex.js';
import {CHTML} from '@mathjax/src/js/output/chtml.js';
import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';
import '@mathjax/src/js/util/asyncLoad/esm.js';
```

These load the objects and classes needed to use MathJax's internal structures.

The first obtains the `mathjax` object, which contains the version number, a function for creating a `MathDocument()` instance that handles the typesetting and conversion for a document, a function for handling asynchronous actions by MathJax, and a function for loading external files dynamically, among other things.

The next two lines load the class constructors for the input and output jax. The fourth loads the LiteDOM adaptor that implements a simple DOM replacement for use within node applications (since node doesn't have a built-in DOM like browsers do). See the *The DOM Adaptor* section for more details about the DOM adaptors available in MathJax.

The fifth line loads a function that registers the code for handling HTML documents, which is currently the only format MathJax understands (but we hope to extend this to other formats like Markdown in the future).

The last line tells MathJax to use `import()` commands to load external files, when needed. In a CommonJS module, you would use `require()` to load `js/util/asyncLoad/node.js` rather than `js/util/asyncLoad/esm.js`, and would replace all the `import` commands by corresponding `require()` calls.

Most of the examples also load a number of TeX extensions:

```
import '@mathjax/src/js/input/tex/base/BaseConfiguration.js';
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';
import '@mathjax/src/js/input/tex/noundefined/NoUndefinedConfiguration.js';
```

Here, we load the *base*, *ams*, *newcommand*, and *noundefined* extensions. The names of these packages are then added to the `packages` array of the options passed to the TeX input jax constructor when it is instantiated later on. The MathJax TeX extensions are in subdirectories of the `ts/input/tex` directory, so you can look there for the configuration files that you can load. See *The TeX/LaTeX Extension List* for more about the TeX extensions. Remember, you must load all the extensions explicitly that you plan to use, and the *autoload* and *require* extensions can't be used, as they rely on the component framework.

Since node applications don't have a fully functional DOM (the LiteDOM is very minimal), MathJax can't determine the font metrics like the em- and ex-sizes, or the font in use, or the width of container elements, as it can in a browser with a full DOM. Thus the next lines define default values for these:

```
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking
```

Then the examples create a DOM adaptor and inform MathJax that is should recognize HTML documents:

```
const adaptor = liteAdaptor({fontSize: EM});
RegisterHTMLHandler(adaptor);
```

Next, the examples create the input and output jax:

```
const tex = new TeX({
  packages: ['base', 'ams', 'newcommand', 'noundefined'],
  formatError(jax, err) {console.error(err.message); process.exit(1)},
  //
  // Other TeX configuration goes here
  //
});
const chtml = new CHTML({
  fontURL: 'https://cdn.jsdelivr.net/npm/@mathjax/mathjax-newcm-font/chtml/woff2',
  //
  // Any output options go here
  //
});
```

Here, we create a TeX input jax instance and configure the `packages` array to include the packages that we loaded above. We also include a *`formatError()`* function that will report the error and then stop the program from running. Since we are only going to process one equation in these examples, that makes it easy to tell when the TeX input is faulty.

Next, we create a CommonHTML (CHTML) output jax, and configure the URL where the font files will be found. If you are hosting your own copy of MathJax, you should replace this URL with the actual URL of where you have placed the font files on your server.

You can include any additional configuration options for the input and output jax, as well, in the indicated locations. For example, if you wanted to predefine some TeX macros, you could load the *configmacros* extension, add it to the `packages` list, and include a `macros` block to the options for the TeX input jax that defines the needed macros.

Similar commands could be used to create an MathML or AsciiMath input jax, or an SVG output jax.

Once we have the input and output jax, we create the `MathDocument()` instance:

```
const html = mathjax.document('', {
  InputJax: tex,
  OutputJax: chtml,
  //
  // Other document options go here
  //
});
```

This specifies the input and output jax, and any other document options that you need to set. The content of the document is blank due to the empty string as the first argument to `mathjax.document()`. Alternatively, you can pass a serialized HTML string, or an actual DOM object or document fragment to create a MathDocument to handle the given content.

After the document is created, we can use it to convert TeX expressions into CHTML output. The heart of this process is a command like the following:

```
const node = html.convert(process.argv[2] || '', {
  display: true,
  em: EM,
  ex: EX,
  containerWidth: WIDTH
});
```

This takes the command-line argument from `process.argv[2]` and treats it as a TeX expression, converting it to CHTML output. The `display` property indicates that it should be typeset as a displayed equation rather than in-line (though you could make that a command-line argument as well), and uses the em- and ex-sizes and container width values defined earlier.

`mathDocument.`**`convert`**`(`*math*`[`, *options* `])`

Arguments

- **math** (`string()`) – The TeX, MathML, or AsciiMath expression to be converted.
- **options** (`OptionList()`) – The options for the conversion. These can include:
  - **format**: the format of the input being passed (`'TeX'`, `'MathML'`, or `'AsciiMath'`). The default is the name of the first input jax of the MathDocument.
  - **display**: Whether this should be typeset in display style or in-line style. The default is `true`. For MathML input, this option is ignored, as the `<math>` tag's `display` attribute is used to specify the display style.
  - **end**: The process state at which the conversion should end. The default is `STATE.LAST`, meaning all render actions should be performed. The initial list of states is in the `ts/core/Mathitem.ts` file, though other files can augment that list.
  - **em**: The em-size of the surrounding font in pixels. The default is 16.
  - **ex**: The ex-size of the surrounding font in pixels. The default is 8.
  - **containerWidth**: The width of the surrounding container element in pixels. The default is `null`, meaning we consider the container to be infinitely wide.
  - **scale**: The scaling factor to be applied to the output. The default is 1.
  - **family**: The name of the surrounding font (for when `mtext` or `merror` use the surrounding font). The default is an empty string.

Returns

The DOM node containing the typeset version of the mathematics.

You could provide the `display`, `ex`, `em`, and other values from command-line arguments, for example, though we use the defaults in these examples.

Note that there is also

`mathDocument.`**`convertPromise`**`(`*math*`[`, *options* `])`

taking the same arguments as *mathDocument.convert()* above, and returning a promise that resolves when the conversion is complete, passing the generated node as the argument to its `then()` method. This function handles any asynchronous file loads, like those needed for dynamic font ranges. Some of the examples below use this function for that purpose.

Finally, we output a JSON object that contains the serialized HTML output along with the CSS stylesheet contents for the expression (this will not be a minimal stylesheet, as, for example, it includes all the web-font definitions, even if those fonts aren't used).

```
//
// Generate a JSON object with the CHTML output and needed CSS
//
console.log(JSON.stringify({
  math: adaptor.outerHTML(node),
  css: adaptor.cssText(chtml.styleSheet(html))
}));
```

Some examples generate other output (for instance, MathML code, or a complete HTML page).

21.2 The Examples

In the examples below, the highlighted lines are the ones that differ from the explanations above, or from previous examples.

21.3 Converting TeX to MathML

This example combines the ideas from the previous sections into a complete example. In this case, it converts a LaTeX expression into a corresponding MathML one.

Listing 1: tex2mml.mjs

```
1  //
2  //  Load the modules needed for MathJax
3  //
4  import {mathjax} from '@mathjax/src/js/mathjax.js';
5  import {TeX} from '@mathjax/src/js/input/tex.js';
6  import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
7  import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';
8  import {SerializedMmlVisitor} from '@mathjax/src/js/core/MmlTree/SerializedMmlVisitor.js
   ↪';
9  import {STATE} from '@mathjax/src/js/core/MathItem.js';
10
11 //
```

(continues on next page)

(continued from previous page)

```
// Import the needed TeX packages
//
import '@mathjax/src/js/input/tex/base/BaseConfiguration.js';
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';
import '@mathjax/src/js/input/tex/noundefined/NoUndefinedConfiguration.js';

//
// The em and ex sizes and container width to use during the conversion
//
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking

//
// Create DOM adaptor and register it for HTML documents
//
const adaptor = liteAdaptor({fontSize: EM});
RegisterHTMLHandler(adaptor);

//
// Create input jax and a (blank) document using it
//
const tex = new TeX({
  packages: ['base', 'ams', 'newcommand', 'noundefined'],
  formatError(jax, err) {console.error(err.message); process.exit(1)},
  //
  // Other TeX configuration goes here
  //
});
const html = mathjax.document('', {
  InputJax: tex,
  //
  // Other document options go here
  //
});

//
//  Create a MathML serializer
//
const visitor = new SerializedMmlVisitor();
const toMathML = (node => visitor.visitTree(node, html));

//
// Convert the math from the command line
//
const mml = html.convert(process.argv[2] || '', {
  display: true,
  em: EM,
  ex: EX,
  containerWidth: WIDTH,
  end: STATE.CONVERT         // stop after conversion to MathML
```

(continues on next page)

(continued from previous page)

```
64 });
65
66 //
67 // Output the resulting MathML
68 //
69 console.log(toMathML(mml));
```

Here, line 9 loads the `STATE` variable that is used in line 63 to stop the conversion process after the LaTeX is compiled into the internal MathML format.

Lines 49 through 53 create a MathML serializer using the `SerializedMmlVisitor()` loaded in line 8. This is used in line 69 to convert the internal MathML to a string form for output.

Running this command as

```
node tex2mml.mjs '\sqrt{1-x^2}'
```

produces

```
<math xmlns="http://www.w3.org/1998/Math/MathML" data-latex="\sqrt{1-x^2}" display="block
↪">
  <msqrt data-latex="\sqrt{1-x^2}">
    <mn data-latex="1">1</mn>
    <mo data-latex="-">&#x2212;</mo>
    <msup data-latex="x^2">
      <mi data-latex="x">x</mi>
      <mn data-latex="2">2</mn>
    </msup>
  </msqrt>
</math>
```

Note that the MathML includes `data-latex` attributes indicating the LaTeX that produced each node. If you don't want those attributes, you can add

```
66 //
67 // Remove data-latex and data-latex-item attributes, if any.
68 //
69 mml.walkTree((node) => {
70   const attributes = node.attributes;
71   attributes.unset('data-latex');
72   attributes.unset('data-latex-item');
73 });
```

at line 66 just before the final output is produced. With this change, the output above becomes

```
<math xmlns="http://www.w3.org/1998/Math/MathML" display="block">
  <msqrt>
    <mn>1</mn>
    <mo>&#x2212;</mo>
    <msup>
      <mi>x</mi>
      <mn>2</mn>
    </msup>
```

(continues on next page)

(continued from previous page)

```
  </msqrt>
</math>
```

21.4 Converting TeX to CHTML

This example puts together all the code blocks from *The Basics of Linking Directly to MathJax* section in order to give an illustration of the complete process. The only new lines are 56 through 59, which cause all the font data to be loaded before the conversion is performed, thus avoiding the need to have to handle dynamically loaded font ranges.

Listing 2: tex2chtml.mjs

```
//
//  Load the modules needed for MathJax
//
import {mathjax} from '@mathjax/src/js/mathjax.js';
import {TeX} from '@mathjax/src/js/input/tex.js';
import {CHTML} from '@mathjax/src/js/output/chtml.js';
import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';
import '@mathjax/src/js/util/asyncLoad/esm.js';

//
// Import the needed TeX packages
//
import '@mathjax/src/js/input/tex/base/BaseConfiguration.js';
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';
import '@mathjax/src/js/input/tex/noundefined/NoUndefinedConfiguration.js';

//
// The em and ex sizes and container width to use during the conversion
//
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking

//
//  Create DOM adaptor and register it for HTML documents
//
const adaptor = liteAdaptor({fontSize: EM});
RegisterHTMLHandler(adaptor);

//
//  Create input and output jax and a (blank) document using them
//
const tex = new TeX({
  packages: ['base', 'ams', 'newcommand', 'noundefined'],
  formatError(jax, err) {console.error(err.message); process.exit(1)},
  //
  // Other TeX configuration goes here
```

(continues on next page)

(continued from previous page)

```
  //
});
const chtml = new CHTML({
  fontURL: 'https://cdn.jsdelivr.net/npm/@mathjax/mathjax-newcm-font/chtml/woff2',
  //
  // Any output options go here
  //
});
const html = mathjax.document('', {
  InputJax: tex,
  OutputJax: chtml,
  //
  // Other document options go here
  //
});

//
// Load all the font data
//
await chtml.font.loadDynamicFiles();

//
// Typeset the math from the command line
//
const node = html.convert(process.argv[2] || '', {
  display: true,
  em: EM,
  ex: EX,
  containerWidth: WIDTH
});

//
// Generate a JSON object with the CHTML output and needed CSS
//
console.log(JSON.stringify({
  math: adaptor.outerHTML(node),
  css: adaptor.cssText(chtml.styleSheet(html))
}));
```

21.4.1 Removing LaTeX Attributes from CHTML

In the *tex2mml* example above, we saw that the internal MathML contains `data-latex` attributes that indicate the LaTeX commands that produce each MathML node. Those attributes are retained in the CHTML output. If you want to remove them, we can use a similar idea to the one above, but since we don't have direct access to the MathML representation in this case, we need to hook into the MathJax rendering pipeline in order to remove the attributes before the CHTML output is created. That can be done by configuring a `renderAction` in the document options when the `html` document is created. This is illustrated below, with changes from the previous example highlighted.

Listing 3: tex2chtml-remove.mjs

```
//
//  Load the modules needed for MathJax
```

(continues on next page)

(continued from previous page)

```
//
import {mathjax} from '@mathjax/src/js/mathjax.js';
import {TeX} from '@mathjax/src/js/input/tex.js';
import {CHTML} from '@mathjax/src/js/output/chtml.js';
import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';
import {STATE} from '@mathjax/src/js/core/MathItem.js';
import '@mathjax/src/js/util/asyncLoad/esm.js';

//
// Import the needed TeX packages
//
import '@mathjax/src/js/input/tex/base/BaseConfiguration.js';
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';
import '@mathjax/src/js/input/tex/noundefined/NoUndefinedConfiguration.js';

//
// The em and ex sizes and container width to use during the conversion
//
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking

//
//  Create DOM adaptor and register it for HTML documents
//
const adaptor = liteAdaptor({fontSize: EM});
RegisterHTMLHandler(adaptor);

//
//  Create input and output jax and a (blank) document using them
//
const tex = new TeX({
  packages: ['base', 'ams', 'newcommand', 'noundefined'],
  formatError(jax, err) {console.error(err.message); process.exit(1)},
});
const chtml = new CHTML({
  fontURL: 'https://cdn.jsdelivr.net/npm/@mathjax/mathjax-newcm-font/chtml/woff2',
});
const html = mathjax.document('', {
  InputJax: tex,
  OutputJax: chtml,
  renderActions: {
    removeLatex: [
      STATE.CONVERT + 1,
      () => {},
      (math, doc) => {
        math.root.walkTree(node => {
          const attributes = node.attributes;
          attributes.unset('data-latex');
          attributes.unset('data-latex-item');
```

(continues on next page)

(continued from previous page)

```
        });
      }
    ]
  },
});

//
// Load all the font data
//
await chtml.font.loadDynamicFiles();

//
// Typeset the math from the command line
//
const node = html.convert(process.argv[2] || '', {
  display: true,
  em: EM,
  ex: EX,
  containerWidth: WIDTH
});

//
// Generate a JSON object with the CHTML output and needed CSS
//
console.log(JSON.stringify({
  math: adaptor.outerHTML(node),
  css: adaptor.cssText(chtml.styleSheet(html))
}));
```

Here, lines 46 through 58 define a render action called `removeLatex` that occurs right after the conversion to MathML (indicated by the `STATE.CONVERT + 1`), and that performs the tree-walking on `math.root`, which is the internal MathML representation of the expression. Since we are only calling `html.convert()` rather than rendering an entire page with `html.render()`, we don't need to provide a document-level function for this action, and so use `() => {}` for that.

21.4.2 Loading Font Ranges Dynamically

In these past two examples, we used `chtml.font.loadDynamicFiles()` to load all the font data, so that dynamic loading would not need to occur during the conversion process. The font data in MathJax version 4 is much more extensive than in v3, due to the more expansive character coverage of the v4 fonts, so loading *all* the data can be time-consuming, especially when most of the data will never be used.

Instead, we can let MathJax load the data as needed. Because loading the data is asynchronous, this requires that we handle the asynchronous nature of those file loads during the conversion process. This is done via the *mathDocument.convertPromise()* function, which returns a promise that is resolved when the conversion process completes, after handling any asynchronous font file loading. The example below shows how to accomplish that.

Listing 4: tex2chtml-promise.mjs

```
//
//  Load the modules needed for MathJax
//
import {mathjax} from '@mathjax/src/js/mathjax.js';
```

(continues on next page)

(continued from previous page)

```
import {TeX} from '@mathjax/src/js/input/tex.js';
import {CHTML} from '@mathjax/src/js/output/chtml.js';
import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';
import '@mathjax/src/js/util/asyncLoad/esm.js';

//
// Import the needed TeX packages
//
import '@mathjax/src/js/input/tex/base/BaseConfiguration.js';
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';
import '@mathjax/src/js/input/tex/noundefined/NoUndefinedConfiguration.js';

//
// The em and ex sizes and container width to use during the conversion
//
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking

//
//  Create DOM adaptor and register it for HTML documents
//
const adaptor = liteAdaptor({fontSize: EM});
RegisterHTMLHandler(adaptor);

//
// Create input and output jax and a (blank) document using them
//
const tex = new TeX({
  packages: ['base', 'ams', 'newcommand', 'noundefined'],
  formatError(jax, err) {throw err},
});
const chtml = new CHTML({
  fontURL: 'https://cdn.jsdelivr.net/npm/@mathjax/mathjax-newcm-font/chtml/woff2',
});
const html = mathjax.document('', {
  InputJax: tex,
  OutputJax: chtml,
});

//
// Typeset the math from the command line
//
html.convertPromise(process.argv[2] || '', {
  display: true,
  em: EM,
  ex: EX,
  containerWidth: WIDTH
}).then((node) => {
  //
```

(continues on next page)

(continued from previous page)

```
  // Generate a JSON object with the CHTML output and needed CSS
  //
  console.log(JSON.stringify({
    math: adaptor.outerHTML(node),
    css: adaptor.cssText(chtml.styleSheet(html))
  }));
}).catch((err) => console.error(err.message));
```

Here, we remove the `chtml.font.loadDynamicFiles()` call, and replace `html.convert()` by `html.convertPromise()`, so that if a font file needs to be loaded, that will be properly handled, putting the rest of the code in its `then()` call. Without this, the `html.convert()` call could throw a *MathJax retry* error; it is the `html.convertPromise()` function that traps and processes those errors as part of the handling of asynchronous file loads.

The other change is that the *formatError()* function now throws the error it receives, which is then trapped by the `catch()` call following the `html.convertPromise()` function and reported there. One could have used the original *formatError()*, but this shows another approach to handling TeX errors.

21.4.3 Specifying The Font to Use

The examples so far, other than the first one, have all used the default font, which is `mathjax-newcm`, based on the New Computer Modern font. MathJax v4 provides a number of other fonts, however (see the *MathJax Font Support* section for details), and you can use any of these to replace the default font.

In the example below, we use the `mathjax-fira` font, which is a sans-serif font. First, install the font using

```
pnpm install @mathjax/mathjax-fira-font
```

(or use `npm` instead of `pnpm`), and then modify the previous example as indicated in the highlighted lines below.

Listing 5: tex2chtml-font.mjs

```
//
//  Load the modules needed for MathJax
//
import {mathjax} from '@mathjax/src/js/mathjax.js';
import {TeX} from '@mathjax/src/js/input/tex.js';
import {CHTML} from '@mathjax/src/js/output/chtml.js';
import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';
import '@mathjax/src/js/util/asyncLoad/esm.js';

//
// Import the needed TeX packages
//
import '@mathjax/src/js/input/tex/base/BaseConfiguration.js';
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';
import '@mathjax/src/js/input/tex/noundefined/NoUndefinedConfiguration.js';

//
// Import the desired font
//
import {MathJaxFiraFont} from '@mathjax/mathjax-fira-font/js/chtml.js';

```

(continues on next page)

(continued from previous page)

```
24 //
25 // The em and ex sizes and container width to use during the conversion
26 //
27 const EM = 16;          // size of an em in pixels
28 const EX = 8;           // size of an ex in pixels
29 const WIDTH = 80 * EM;  // width of container for linebreaking
30
31 //
32 //  Create DOM adaptor and register it for HTML documents
33 //
34 const adaptor = liteAdaptor({fontSize: EM});
35 RegisterHTMLHandler(adaptor);
36
37 //
38 //  Create input and output jax and a (blank) document using them
39 //
40 const tex = new TeX({
41   packages: ['base', 'ams', 'newcommand', 'noundefined'],
42   formatError(jax, err) {throw err},
43 });
44 const chtml = new CHTML({
45   fontData: MathJaxFiraFont,
46   fontURL: 'https://cdn.jsdelivr.net/npm/@mathjax/mathjax-fira-font/chtml/woff2',
47 });
48 const html = mathjax.document('', {
49   InputJax: tex,
50   OutputJax: chtml,
51 });
52
53 //
54 // Typeset the math from the command line
55 //
56 html.convertPromise(process.argv[2] || '', {
57   display: true,
58   em: EM,
59   ex: EX,
60   containerWidth: WIDTH
61 }).then((node) => {
62   //
63   // Generate a JSON object with the CHTML output and needed CSS
64   //
65   console.log(JSON.stringify({
66     math: adaptor.outerHTML(node),
67     css: adaptor.cssText(chtml.styleSheet(html))
68   }));
69 }).catch((err) => console.error(err.message));
```

Here, line 22 imports the Fira font class, which is passed to the CHTML output jax at line 45. Line 46 now needs to point to the `mathjax-fira-font` directory on the CDN.

The earlier examples could be modified in a similar way, as well.

21.5 Mixing Components and Direct Linking

One of the drawbacks to using direct loading of MathJax modules is that you don't have the MathJax component framework to work with, which means you can't use `\require{}` or autoloaded TeX components, for example. It is possible to use both together, however, by importing the needed component definitions. This is done in the *Using Components Synchronously* section to show how to handle synchronous typesetting, but this technique also can be used more generally with the promise-based commands, as illustrated in the example below.

Listing 6: tex2chtml-mixed.mjs

```
//
//  Load the modules needed for MathJax
//
import {mathjax} from '@mathjax/src/js/mathjax.js';
import {TeX} from '@mathjax/src/js/input/tex.js';
import {CHTML} from '@mathjax/src/js/output/chtml.js';
import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';

//
// Load the component definitions
//
import {Loader} from '@mathjax/src/js/components/loader.js';
import {Package} from '@mathjax/src/js/components/package.js';
import '@mathjax/src/components/js/startup/init.js';
import '@mathjax/src/components/js/core/lib/core.js';
import '@mathjax/src/components/js/input/tex/tex.js';
import '@mathjax/src/components/js/output/chtml/chtml.js';

//
// Record the pre-loaded component files
//
Loader.preLoaded(
  'loader', 'startup',
  'core',
  'input/tex',
  'output/chtml',
);

//
// The em and ex sizes and container width to use during the conversion
//
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking

//
// Set up methods for loading dynamic files
//
MathJax.config.loader.require = (file) => import(file);
mathjax.asyncLoad = (file) => import(Package.resolvePath(file));

//
```

(continues on next page)

(continued from previous page)

```
44 //  Create DOM adaptor and register it for HTML documents
45 //
46 const adaptor = liteAdaptor({fontSize: EM});
47 RegisterHTMLHandler(adaptor);
48
49 //
50 // Create input and output jax and a (blank) document using them
51 //
52 const tex = new TeX({
53   formatError(jax, err) {throw err},
54   ...(MathJax.config.tex || {})
55 });
56 const chtml = new CHTML({
57   ...(MathJax.config.output || {}),
58   ...(MathJax.config.chtml || {}),
59   fontURL: 'https://cdn.jsdelivr.net/npm/@mathjax/mathjax-newcm-font/chtml/woff2',
60 });
61 const html = mathjax.document('', {
62   InputJax: tex,
63   OutputJax: chtml,
64   ...(MathJax.config.options || {})
65 });
66
67 //
68 // Typeset the math from the command line
69 //
70 html.convertPromise(process.argv[2] || '', {
71   display: true,
72   em: EM,
73   ex: EX,
74   containerWidth: WIDTH
75 }).then((node) => {
76   //
77   // Generate a JSON object with the CHTML output and needed CSS
78   //
79   console.log(JSON.stringify({
80     math: adaptor.outerHTML(node),
81     css: adaptor.cssText(chtml.styleSheet(html))
82   }));
83 }).catch((err) => console.error(err.message));
```

Here, lines 10 through 18 load the component framework and definition files. The first two lines obtain the `Loader()` and `Package()` class definitions, which are the heart of the component framework. The next line initializes the *startup* component, which sets up some of the needed component configuration. The next line loads the core component definition, and the final two lines load the input and output jax component definitions for the ones we will be using.

Lines 20 through 28 register the pre-loaded components with the loader so that it won't try to load them again.

Lines 37 through 41 define the methods needed for dynamic loading of files. The first tells MathJax to use `import()` to load files, and the second tells MathJax to use `Package.resolvePath()` to process file names before importing them. That allows references like `[tex]/cancel` or `[mathjax-fira]/chtml/dynamic/calligraphic` to be resolved to their full URLs before they are loaded.

Lines 54, 57, 58, and 64 incorporate the appropriate MathJax configuration blocks into the options used for creating

the input and output jax and the math document. The component files initialize some of these values, and if the `tex2chtml-mixed.mjs` file is imported into another application, that would allow that application to provide its own `MathJax` configuration object, just like in a web page, and its configuration would be incorporated into the creation of the MathJax objects here.

Note that the `import` commands that loaded the TeX packages (*base*, *ams*, *newcommand*, and *noundefined*) have been removed, as the `input/tex` component loads those (and several others) itself. Note also that the `packages` array has been removed, as the `input/tex` component defines that itself, and already includes the packages that it loads.

With these changes, the LaTeX being processed can now use `\require`, and macros that autoload extensions will work as well. So this gives you the best of both worlds: the convenience of MathJax components, and the control of direct imports.

If you want to preload additional TeX packages, you can import them and then push their names onto the `tex.packages` array prior to instantiating the TeX input jax. For example

```
import '@mathjax/src/components/js/input/tex/extensions/mathtools/mathtools.js';
import '@mathjax/src/components/js/input/tex/extensions/physics/physics.js';
MathJax.config.tex.packages.push('mathtools', 'physics');
Loader.preLoaded('[tex]/mathtools', '[tex]/physics');
```

could be added at line 19 in order to include the *mathtools* and *physics* TeX packages.

21.6 Generating Speech Strings without Typesetting

One of MathJax's most important features is the ability to generate speech strings from mathematical notation. MathJax uses the Speech Rule Engine (SRE) to perform this function.

The example below is based on the *Converting TeX to MathML* code described above, with the changes highlighted.

Listing 7: tex2speech.mjs

```
1  //
2  //  Load the modules needed for MathJax
3  //
4  import {mathjax} from '@mathjax/src/js/mathjax.js';
5  import {TeX} from '@mathjax/src/js/input/tex.js';
6  import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
7  import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';
8  import {SerializedMmlVisitor} from '@mathjax/src/js/core/MmlTree/SerializedMmlVisitor.js
   ↪';
9  import {STATE} from '@mathjax/src/js/core/MathItem.js';
10
11 //
12 // Import the speech-rule-engine
13 //
14 import '@mathjax/src/components/require.mjs';
15 import {setupEngine, engineReady, toSpeech} from 'speech-rule-engine/js/common/system.js
   ↪';
16
17 //
18 // Import the needed TeX packages
19 //
```

(continues on next page)

(continued from previous page)

```
import '@mathjax/src/js/input/tex/base/BaseConfiguration.js';
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';
import '@mathjax/src/js/input/tex/noundefined/NoUndefinedConfiguration.js';

//
// The em and ex sizes and container width to use during the conversion
//
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking

//
// Create DOM adaptor and register it for HTML documents
//
const adaptor = liteAdaptor({fontSize: EM});
RegisterHTMLHandler(adaptor);

//
// Create input jax and a (blank) document using it
//
const tex = new TeX({
  packages: ['base', 'ams', 'newcommand', 'noundefined'],
  formatError(jax, err) {console.error(err.message); process.exit(1)},
  //
  // Other TeX configuration goes here
  //
});
const html = mathjax.document('', {
  InputJax: tex,
  //
  // Other document options go here
  //
});

//
//  Create a MathML serializer
//
const visitor = new SerializedMmlVisitor();
const toMathML = (node => visitor.visitTree(node, html));

//
// Convert the math from the command line
//
const mml = html.convert(process.argv[2] || '', {
  display: true,
  em: EM,
  ex: EX,
  containerWidth: WIDTH,
  end: STATE.CONVERT         // stop after conversion to MathML
});

```

(continues on next page)

(continued from previous page)

```
72 //
73 // Set up the speech engine to use English
74 //
75 const locale = process.argv[3] || 'en';
76 const modality = locale === 'nemeth' || locale === 'euro' ? 'braille' : 'speech';
77 await setupEngine({locale, modality}).then(() => engineReady());
78
79 //
80 // Produce the speech for the converted MathML
81 //
82 console.log(toSpeech(toMathML(mml)));
```

Here, line 15 loads the functions needed for speech generation. Because SRE uses `require()` to load its dependencies when used from node, line 14 makes that available before loading SRE.

Line 75 gets the locale (i.e., the language) to use for the speech, and line 76 determines whether we are generating speech or Braille strings. Line 77 sets up SRE for the proper locale and modality, and waits for the needed files to be loaded.

Finally, line 82 uses the `toSpeech()` function from SRE to convert the MathML generated from the TeX into the needed speech or Braille string.

When this node application is run from the command line, the first command line argument is the LaTeX string to convert, while the output is the corresponding speech string. For example,

```
node tex2speech.mjs '\frac{a}{b}'
```

would generate the phrase

```
StartFraction a Over b EndFraction
```

SRE can generate speech in several languages; here, the default language is English, but the second command-line argument can be used to specify a different language locale. For example,

```
node tex2speech.mjs '\frac{a}{b}' de
```

would generate the German phrase

```
Anfang Bruch a durch b Ende Bruch
```

while

```
node tex2speech.mjs '\frac{a}{b}' nemeth
```

would generate the Braille code

```

```

The available locales can be found in the `bundle/sre/mathmaps` directory.

There are several different speech rulesets that SRE can use, including *clearspeak*, *mathspeak*, and *chromvox* rules. You can add a `domain` option to the list passed to `setupEngine()` in order to specify which of these rulesets to use. The default is *mathspeak*.

21.7 Pre-processing a Complete Page

All of the previous examples convert a single mathematical expression at a time, but you may wish to preprocess an entire web page, rather than individual expressions. In that case, you would load the page's text and use that when creating the math document, and then call `html.renderPromise()` rather than `html.convertPromise()`.

This is illustrated with the example below, this time using SVG output rather than CHTML output.

Listing 8: tex2svg-page.mjs

```
import fs from 'fs';

//
//  Load the modules needed for MathJax
//
import {mathjax} from '@mathjax/src/js/mathjax.js';
import {TeX} from '@mathjax/src/js/input/tex.js';
import {SVG} from '@mathjax/src/js/output/svg.js';
import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';
import '@mathjax/src/js/util/asyncLoad/esm.js';

//
// Import the needed TeX packages
//
import '@mathjax/src/js/input/tex/base/BaseConfiguration.js';
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';
import '@mathjax/src/js/input/tex/noundefined/NoUndefinedConfiguration.js';

//
// The em and ex sizes to use during the conversion
//
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels

//
//  Create DOM adaptor and register it for HTML documents
//
const adaptor = liteAdaptor({fontSize: EM});
RegisterHTMLHandler(adaptor);

//
//  Read the HTML file
//
const htmlfile = fs.readFileSync(process.argv[2] || 0, 'utf8');

//
//  Create input and output jax and a document using them on the HTML file
//
const tex = new TeX({
  packages: ['base', 'ams', 'newcommand', 'noundefined'],
  formatError(jax, err) {console.error(err.message); return jax.formatError(err)},
  //
```

(continues on next page)

(continued from previous page)

```
45    // Other TeX configuration goes here
46    //
47  });
48  const svg = new SVG({
49    fontCache: 'global',
50    exFactor: EX / EM,
51    //
52    // Any output options go here
53    //
54  });
55  const html = mathjax.document(htmlfile, {
56    InputJax: tex,
57    OutputJax: svg,
58    //
59    // Other document options go here
60    //
61  });
62
63  //
64  // Wait for the typesetting to finish
65  //
66  await html.renderPromise();
67
68  //
69  // If no math was found on the page, remove the stylesheet and font cache (if any)
70  //
71  if (Array.from(html.math).length === 0) {
72    adaptor.remove(svg.svgStyles);
73    const cache = adaptor.elementById(adaptor.body(html.document), 'MJX-SVG-global-cache');
74    if (cache) adaptor.remove(cache);
75  }
76
77  //
78  // Output the resulting HTML
79  //
80  console.log(adaptor.doctype(html.document));
81  console.log(adaptor.outerHTML(adaptor.root(html.document)));
```

Here, line 1 loads the node `fs` library that is used in line 36 to read the HTML file that is to be processed (either the one given as the first command-line argument, or from standard input when the script is run as a filter).

Line 8 loads the SVG output jax rather than the CHTML one, and lines 48 through 54 instantiate the output jax. We set the `fontCache` to `global` so that all the SVG path data will be stored in one place and can be shared among all the expressions on the page rather than having individual copies for each expression. We also set up the ex-to-em factor, since we can't measure that directory using the LiteDOM in node. Line 57 uses the SVG output jax that we just created rather than the CHTML one from previous examples.

Line 66 is the key change from the previous examples, which now uses `html.renderPromise()` to process the entire page, rather than just process a single expression. We use the promise-based function so that we handle any font data files that need to be loaded.

Lines 72 to 75 check to see that there is actually math that was processed on the page, and if not, it removes the stylesheet and font-cache <svg> element that it added to the page, leaving the page essentially untouched.

Lines 80 and 81 produce the final output for the pre-processed page.

Finally, the *formatError()* function on line 43 now logs the error and then calls the default *formatError()* function, so that the error will appear within the final HTML document as it would in a web page.

Note that when pre-processing a page, there are a number of limitations:

- All the expressions will use the same ex-to-em factor, since MathJax can't measure the font metrics of the surrounding font in node.
- The handling of characters that are not in the MathJax fonts will be problematic, as MathJax can't measure their sizes, so will have to make assumptions that probably aren't right. The output will try to use a fixed-width font in order to try to reduce its error, but that can produce ugly results. Fortunately, the fonts in MathJax v4 have much greater coverage than in v3, so the problem should occur much less often.
- The menu code relies on MathJax actually being active in the page, and when the page is pre-processed in this way, that will not be the case. So the menu is not available in pre-processed pages. That means your readers won't be able to view or copy the math notation, change the renderer, or use any of the other features available in the MathJax contextual menu.
- The expression explorer also relies on MathJax being present, and so that feature will not be available in pre-processed pages. You may wish to include the *assistive-mml* extension in order to help support users who require screen readers, as described below.
- Finally, since MathJax doesn't know the size of the screen that your user will be using when viewing your page, it is not able to do automatic line breaking of displayed equations, unless you configure the line-breaking width explicitly. In-line breaks will still be possible, since they are determined by the browser at run time.

To include the *assistive-mml* extension, add the following `import` command

```
import {AssistiveMmlHandler} from '@mathjax/src/js/a11y/assistive-mml.js';
```

and change line 31 to be

```
AssistiveHandler(RegisterHTMLHandler(adaptor));
```

That will cause MathJax to include visually hidden MathML that can be read by screen readers.

21.8 More Examples

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

CHAPTER

TWENTYTWO

THE DOM ADAPTOR

Because node applications don't have access to a built-in Document Object Model (DOM), like browsers do, node applications must use an alternative to a native DOM. There are a number of node packages that offer DOM substitutes, such as *jsDOM* and *linkedom*. These libraries generally provide implementations of the browser DOM in javascript that include more functionality than is required by MathJax in a node application, and so MathJax provides its own LiteDOM for this purpose. This is a minimal implementation of a DOM that includes just enough for MathJax to do its job, so is light-weight and fast, but MathJax can be made to work with most DOM implementations. For example, if your node application is already using a DOM library, you may want to have MathJax work with that rather than its own LiteDOM.

The key to this is MathJax's concept of a *DOM adaptor*. This is an interface that standardizes the functionality that MathJax needs, and that converts those actions into the corresponding calls appropriate for the DOM implementation being used. This makes it possible to use MathJax with DOM-like data structures that don't implement the HTMLElement interface (such as the LiteDOM), for example.

The interaction that MathJax does with DOM elements is mediated through the DOM adaptor. For example, creation of new DOM nodes is handled through the `adaptor.node()` method, setting node attributes is done through `adaptor.setAttribute()`, and measuring the size of an element is accomplished via `adaptor.nodeSize()`. All MathJax modules use the DOM adaptor whenever they create or manipulate DOM elements, expect those modules that can run only in a browser setting, like the expression explorer and the MathJax contextual menu.

When the initial `MathDocument()` is created, it gets a DOM adaptor to handle interactions with the DOM that is part of the document that the MathDocument controls. This adaptor is passed on to the output jax, and any other component that needs to handle the document DOM. If you are writing a MathJax component, you will want to use the DOM adaptor rather than manipulate DOM elements directly if your extension is meant to work within node applications and not just the browser. The examples in the MathJax Node Demos repository all use the DOM adaptor to handle interaction with DOM elements.

The interface for the DOM adaptor is given in the ts/core/DOMadaptor.ts file, where you can see that methods that are available for you to call from your own code.

MathJax includes DOM adaptors for the browser DOM and its own LiteDOM, as well as for *jsDOM* and *linkedom*. If you wish to create an adaptor for another DOM implementation, you should look at the ones found in the ts/adaptors directory for examples.

CHAPTER

TWENTYTHREE

EXAMPLES OF MATHJAX IN NODE

There are a number of examples of using MathJax from within node documents available within this documentation.

In addition, the MathJax Node Demos repository includes numerous examples of how to use MathJax from a *node* application. There are examples using modern ESM modules, and others using the older CommonJS format. These are each grouped into several categories that illustrate four different ways to access the MathJax code. The main examples use MathJax's LiteDOM implementation, but there are also some examples using other DOM replacements.

- Using MathJax components in ESM modules
- Using MathJax components in CJS modules
- Using MathJax with alternative DOM implementations

CHAPTER

TWENTYFOUR

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 and above, the `tex2jax` functions have been folded into the TeX input jax itself.

The TeX input jax can be configured to look for whatever textual 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. You can not configure the TeX input jax to use HTML tags as TeX math delimiters, though it is possible to use a custom render action to look for such tags. The *Changes in the MathJax API* section includes an example of how to do this for the v2-style `<script type="math/tex">` tags, for example.

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 that MathJax's TeX input processor is *not* actual LaTeX, it is an implementation of a subset of the LaTeX macros in javascript in a browser. Not all LaTeX macros and control sequences are available, and there are some differences in how MathJax interprets some expressions, as described in the first link below. Not all LaTeX packages are available in MathJax; see the sections on extension below.

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

24.1 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 jax.

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`, `\begin{enumerate}...\end{enumerate}`, `\begin{tabular}...\end{tabular}` 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 a few exception to this rule. First, MathJax supports the `\ref` and `\eqref` macros outside of math mode. Also, MathJax supports some macros that add text within math mode (such as the `\text{}` macro), and allows `$...$` and `\(...\)` to switch back into math mode from within text mode. In v3 and below, MathJax did not process macros within text mode, other than escaping special characters like `\$`, `\{`, and `\}`. So, for example, `\text{some \textbf{bold} text}` would produce the output "some \textbf{bold} text", not "some **bold** text" in v3.

Version 3.1 introduced a new `[tex]/textmacros` extension that implemented a number of text-mode macros within `\text{}` and other macros that produce text-mode material. See the *textmacros* documentation for details. In version 4, this extension is included by default in all the combined configuration files, so `\text{some \textbf{bold} text}` *does* produce "some **bold** text" in v4.

Second, because MathJax concentrates in math-mode, not text-mode, some macros that initiate text-mode in actual TeX may stay in math-mode in MathJax. For example, the argument `\llap{}` and `\rlap{}` in actual TeX is typeset in text-mode, but in MathJax, these remain in math-mode so that you don't have to use dollar signs to go back into math-mode as you would in actual TeX.

Third, TeX's rules about how arguments to macros are subtle and not always consistent. For example, `\sqrt \frac a b` will produce the square root of the fraction *a*/*b*, but `\hat \frac a b` will produce an error. Both of these expressions represent poor practice, as the arguments should be within braces to make the grouping clear: `\sqrt{\frac{a}{b}}` and `\hat{\frac{a}{b}}`. MathJax standardizes the handling of braces more than TeX does, and sometimes requires the braces when TeX may not. So some expression that TeX or LaTeX will accept will cause errors in MathJax if they don't include the usual braces.

Finally, some features in MathJax might be more limited than in actual LaTeX. For example, MathJax doesn't implement the `\multicolumn`, `\multirow`, `\omit`, `\intertext`, or `\vadjust` macros, so table layout is more restricted in MathJax, currently. Similarly, `\hfil` and related macros are only allowed at the left and right-hand ends of the contents of a cell within a table.

24.2 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: {'[+]': [['$', '$']]}
  }
};
```

This adds the single dollar signs as additional in-line math delimiters.

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.

Alternatively, putting the dollar sign inside a tag would also prevent it from being treated as a math delimiter. That is,

```
... the cost is <span>$</span>2.50 for the first one, and
<span>$</span>2.00 for each additional one ...
```

would also prevent the dollar signs from being treated as math delimiters. This is because MathJax only matches delimiters that appear within the same parent element of the HTML page, so isolating the dollar sign inside a tag prevents it from matching with another dollar sign to delimit a mathematical expression.

In version 3 and below, TeX expressions can not contain HTML elements (other than <br>, <wbr>, and comments), but in version 4, a new *texhtml* extension provides a means of including HTML notation within TeX expressions. See the *texhtml* page for details. Placing a dollar sign with a would still prevent it from being treated as a delimiter, even with the *texhtml* extension.

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.

24.3 TeX and LaTeX in HTML documents

24.3.1 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.

New in version 4, however, is a *texhtml* extension that allows you to embed HTML tags within a TeX expression. This allows you to include form input elements or images in your mathematics, for example. See the *texhtml* page for more details.

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.

24.3.2 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). If expressions with two or more subscripts aren't being displayed properly, and especially if you see the intervening text appearing in italics, that is a good sign that Markdown is processing the underscores before MathJax runs.

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. Note, however, that this may put the TeX expression and its delimiters inside a <code> tag in the page, and MathJax usually ignores the contents of those. In that case, you will need to configure MathJax to not skip <code> tags. For example, use

```
window.MathJax = {
  options: {
    skipHtmlTags: {'[-]': ['code']}
  }
};
```

to remove the <code> tag from the list of those that are skipped when looking for math delimiters.

Alternatively, some content-management systems, including those based on Markdown, 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.

24.4 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.

24.4.1 Configuring Macros at Startup

You can include macro definitions in the `macros` section of the `tex` block 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, or use the javascript `String.raw` method for specifying the string literal.

Similarly, you can create new environments with the `environments` section of the `tex` block of your configuration, and active characters (single characters that act as a macro without the need for a backslash) using the `active` section of the `tex` block of your configuration.

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

24.4.2 Running TeX Code During Startup

Alternatively, you can use the *ready()* function in the `startup` block of your configuration to process a TeX expression that defines the macros, as in the following example:

```
window.MathJax = {
  startup: {
    ready() {
      MathJax.startup.defaultReady();
      const {STATE} = MathJax._.core.MathItem;
      MathJax.tex2mml(String.raw`
        \newcommand{\RR}{\mathbf{R}}
        \newcommand{\bold}[1]{\mathbf{#1}}
        \let\star=\ast
      `);
    }
  }
};
```

This allows you to use actual TeX commands, but without having to have an actual TeX block within the page to make the definitions.

If you need to have definitions that are different for each page, then you could include them in a script that specifies the definitions, and then include that in the *ready()* call. For example,

```
<script type="text/x-tex-macros">
  \newcommand{\RR}{\mathbf{R}}
  \newcommand{\bold}[1]{\mathbf{#1}}
  \let\star=\ast
</script>
<script>
  window.MathJax = {
    startup: {
      ready() {
        MathJax.startup.defaultReady();
        const {STATE} = MathJax._.core.MathItem;
        const defs = document.querySelector('script[type="text/x-tex-macros"]');
        MathJax.tex2mml(defs?.textContent || '');
      }
    }
  };
</script>
```

will take the definitions from the first script, if any, and process them during its startup phase. You can have the first script be part of the page, and the second be part of the scripts loaded into every page, and that will allow you to have page-specific definitions. It would also be possible to use a regular `<script>` tag to set a javascript variable (rather than using `type="text/x-tex-macros"`) and use that value in the `convert()` call that rather than looking up the contents of a script.

24.5 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 using

```
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 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 are displayed 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.

24.6 TeX and LaTeX extensions

While MathJax includes nearly all of the Plain TeX math macros, and many of the LaTeX macros and environments, not all of these are 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 many 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.

24.6.1 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 the `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`, include the `ams`, `newcommand`, `noundefined`, `textmacros`, `require`, `autoload`, and `configmacros` extensions, with most of the other extensions being autoloaded as needed. The `input/tex-base` component has no extensions loaded, so contains only the base macros.

Note

In version 3, there were combined configurations that ended with `-full` that included most of the TeX extensions. Because the number of extensions is growing, and some are third-party extensions, it is not longer feasible to include all of them, so these `-full` combined components have been dropped in v4.

If you load a combined 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 loading the `tex-chtml.js` combined component file or another one that includes those extensions.

24.6.2 Loading Extensions at Run Time

You can load 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).

24.6.3 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 *MathJax Configuration Options* 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), and will complain that your configuration includes options with no default values. 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'
  }
};
```

If the `color` macro is autoloaded from within the page, that configuration will be used to initialize the extension.

Finally, the *setoptions* extension provides a `\setOptions` command that can be used to change the options on they fly as the web page is being processed. See the *setoptions* section for details.

24.7 The TeX/LaTeX Extension List

The main extensions are described below:

24.7.1 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* extension is loaded.

---

action Commands

The *action* extension implements the following macros: `\mathtip`, `\texttip`, `\toggle`

24.7.2 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']}}
};
```

---

ams Options

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

```
MathJax = {
  tex: {
    ams: {
      operatornamePattern:  /^[-*a-zA-Z]+/,
      multlineWidth: '100%',
      multlineIndent: '1em'
    }
  }
};
```

operatornamePattern: /^[-*a-zA-Z]+/

The `multiLetterIdentifier` pattern to use for the contents of `\operatorname{}` arguments to identify what should be used as the contents of a single <mo> element.

multlineWidth: '100%'

The width to use for multline environments.

multlineIndent: '1em'

The margin to use on both sides of multline environments.

Note

The `mutlineWidth` option used to be in the main `tex` block, but as of version 3.2, it is now in the `ams` sub-block of the `tex` block. Version 3.2 includes code to move the configuration from its old location to its new one, but that backward-compatibility code has been removed in version 4.

---

ams Commands

The *ams* extension implements the following macros: \approxeq, \backepsilon, \backprime, \backsim, \backsimeq, \barwedge, \Bbbk, \because, \beth, \between, \bigstar, \binom, \blacklozenge, \blacksquare, \blacktriangle, \blacktriangledown, \blacktriangleleft, \blacktriangleright, \Box, \boxdot, \boxed, \boxminus, \boxplus, \boxtimes, \bumpeq, \Bumpeq, \Cap, \centerdot, \cfrac, \checkmark, \circeq, \circlearrowleft, \circlearrowright, \circledast, \circledcirc, \circleddash, \circledR, \circledS, \complement, \Cup, \curlyeqprec, \curlyeqsucc, \curlyvee, \curlywedge, \curvearrowleft, \curvearrowright, \daleth, \dashleftarrow, \dashrightarrow, \dbinom, \ddddot, \dddot, \DeclareMathOperator, \dfrac, \diagdown, \diagup, \Diamond, \digamma, \divideontimes, \Doteq, \doteqdot, \dotplus, \doublebarwedge, \doublecap, \doublecup, \downdownarrows, \downharpoonleft, \downharpoonright, \eqcirc, \eqref, \eqsim, \eqslantgtr, \eqslantless, \eth, \fallingdotseq, \Finv, \frac, \Game, \genfrac, \geqq, \geqslant, \ggg, \gggtr, \gimel, \gnapprox, \gneq, \gneqq, \gnsim, \gtrapprox, \gtrdot, \gtreqless, \gtreqqless, \gtrless, \gtrsim, \gvertneqq, \hslash, \idotsint, \iiiint, \impliedby, \implies, \injlim, \intercal, \Join, \leadsto, \leftarrowtail, \leftleftarrows, \leftrightarrows, \leftrightharpoons, \leftrightsquigarrow, \leftthreetimes, \leqq, \leqslant, \lessapprox, \lessdot, \lesseqgtr, \lesseqqgtr, \lessgtr, \lesssim, \lhd, \llcorner, \Lleftarrow, \lll, \llless, \lnapprox, \lneq, \lneqq, \lnsim, \looparrowleft, \looparrowright, \lozenge, \lrcorner, \Lsh, \ltimes, \lvert, \lVert, \lvertneqq, \maltese, \mathring, \measuredangle, \mho, \multimap, \ncong, \negmedspace, \negthickspace, \nexists, \ngeq, \ngeqq, \ngeqslant, \ngtr, \nleftarrow, \nLeftarrow, \nleftrightarrow, \nLeftrightarrow, \nleq, \nleqq, \nleqslant, \nless, \nmid, \nobreakspace, \notag, \nparallel, \nprec, \npreceq, \nrightarrow, \nRightarrow, \nshortmid, \nshortparallel, \nsim, \nsubseteq, \nsubseteqq, \nsucc, \nsucceq, \nsupseteq, \nsupseteqq, \ntriangleleft, \ntrianglelefteq,

\ntriangleright, \ntrianglerighteq, \nvdash, \nvDash, \nVdash, \nVDash, \operatorname, \pitchfork, \precapprox, \preccurlyeq, \precnapprox, \precneqq, \precnsim, \precsim, \projlim, \restriction, \rhd, \rightarrowtail, \rightleftarrows, \rightleftharpoons, \rightrightarrows, \rightsquigarrow, \rightthreetimes, \risingdotseq, \Rrightarrow, \Rsh, \rtimes, \rvert, \rVert, \shortmid, \shortparallel, \shoveleft, \shoveright, \sideset, \smallfrown, \smallsetminus, \smallsmile, \sphericalangle, \sqsubset, \sqsupset, \square, \Subset, \subseteqq, \subsetneq, \subsetneqq, \substack, \succapprox, \succcurlyeq, \succnapprox, \succneqq, \succnsim, \succsim, \Supset, \supseteqq, \supsetneq, \supsetneqq, \tag, \tbinom, \tfrac, \therefore, \thickapprox, \thicksim, \triangledown, \trianglelefteq, \triangleq, \trianglerighteq, \twoheadleftarrow, \twoheadrightarrow, \ulcorner, \unlhd, \unrhd, \upharpoonleft, \upharpoonright, \upuparrows, \urcorner, \varDelta, \varGamma, \varinjlim, \varkappa, \varLambda, \varliminf, \varlimsup, \varnothing, \varOmega, \varPhi, \varPi, \varprojlim, \varpropto, \varPsi, \varSigma, \varsubsetneq, \varsubsetneqq, \varsupsetneq, \varsupsetneqq, \varTheta, \vartriangle, \vartriangleleft, \vartriangleright, \varUpsilon, \varXi, \vDash, \Vdash, \veebar, \Vvdash, \xleftarrow, \xrightarrow, \yen

And the following environments: `align*`, `align`, `alignat*`, `alignat`, `aligned`, `alignedat`, `bmatrix`, `Bmatrix`, `cases`, `eqnarray*`, `equation*`, `flalign*`, `flalign`, `gather*`, `gather`, `gathered`, `matrix`, `multline*`, `multline`, `pmatrix`, `smallmatrix`, `split`, `subarray`, `vmatrix`, `Vmatrix`, `xalignat*`, `xalignat`, `xxalignat`

24.7.3 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.)

---

amscd Commands

The *amscd* extension implements the following macros: `@`, `\minCDarrowheight`, `\minCDarrowwidth`

And the following environments: `CD`

24.7.4 autoload

The *autoload* extension predefines macros from some of the extensions that haven't been loaded already so that they automatically load the needed extension when the macro is first used. Some exceptions are the *physics* package, since it redefines standard macros, and the *ams* package, due to the large number of macros it contains, and the fact that it is already included in most of the combined components.

> **Warning**
>
> Because the number of available macros is growing, new TeX extensions are not being added to *autoload* as they are developed, and the *autoload* extension may be phased out in the future. Extensions that are **not** auto-loaded include: *ams*, *bbm*, *bboldx*, *begingroup*, *cases*, *centernot*, *colortbl*, *dsfont*, *empheq*, *mathtools*, *physics*, *require*, *setoptions*, *textcomp*, *textmacros*, *units*, and *upgreek*.

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 has access to many of the TeX extensions, even if they aren't loaded initially, and you should rarely have to use `\require` or load other extensions 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: {
```

(continues on next page)

(continued from previous page)

```
      verb: ['verb']
    }
  }
};
```

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 v2 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']
    }
  }
};
```

The *autoload* 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 for those components. 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: {
      action: ['toggle', 'mathtip', 'texttip'],
      amscd: [[], ['CD']],
      bbox: ['bbox'],
      boldsymbol: ['boldsymbol'],
      braket: [
        'bra',
        'ket',
        'braket',
        'set',
        'Bra',
        'Ket',
        'Braket',
        'Set',
        'ketbra',
        'Ketbra',
      ],
      bussproofs: [[], ['prooftree']],
      cancel: ['cancel', 'bcancel', 'xcancel', 'cancelto'],
      color: ['color', 'definecolor', 'textcolor', 'colorbox', 'fcolorbox'],
      enclose: ['enclose'],
      extpfeil: [
        'xtwoheadrightarrow',
        'xtwoheadleftarrow',
        'xmapsto',
        'xlongequal',
        'xtofrom',
        'Newextarrow',
      ],
      html: ['data', 'href', 'class', 'style', 'cssId'],
      mhchem: ['ce', 'pu'],
      newcommand: [
        'newcommand',
        'renewcommand',
        'newenvironment',
        'renewenvironment',
        'def',
        'let',
      ],
      unicode: ['unicode', 'U', 'char'],
      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: {
      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: {
      '[extensions]/myExtension' : ['myMacro', 'myOtherMacro']
    }
  }
};
```

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

24.7.5 bbm

The *bbm* extension implements the bbm style package from LaTeX, which makes alternative blackboard-bold characters available in TeX. See the bbm CTAN page for more information and documentation.

This extension defines the following commands:

\mathbbm{math}

Typesets `math` using the bbm blackboard-bold fonts.

\mathbbmss{math}

Typesets `math` using the bbm blackboard-bold sans-serif fonts.

\mathbbmtt{math}

Typesets `math` using the bbm blackboard-bold typewriter fonts.

\mathversion{name}

Specifies whether the use the bold or normal versions of the bbm fonts. If `name` is `bold`, the bold versions are used; anything else indicates the normal versions. This is a global setting, whose default is given by the `bbm.bold` setting described *below*.

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

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

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

---

bbm Options

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

```
MathJax = {
  tex: {
    bbm: {
      bold: false
    }
  }
};
```

`bold: false`
: Specifies whether the bold versions of the bbm fonts are to be used. The default is to use the normal versions.

---

bbm Commands

The *bbm* extension implements the following macros: `\mathbbm`, `\mathbbmss`, `\mathbbmtt`, `\mathversion`

24.7.6 bboldx

The *bboldx* extension implements the `bboldx` style package from LaTeX, which makes alternative black-board bold characters available in TeX. This includes not just the alphabetic and numeric characters, but also for some punctuation, and macros for Greek upper- and lower-case letters, along with several delimiters. See the bboldx CTAN page for more information and documentation.

This package redefines the `\mathbb` macro to use one of the bboldx fonts, based on the options described *below*. This will typeset letters, numbers, and some punctuation using the bboldx fonts. To get the Greek letters, however, you must use macros like `\bbGamma` and `\bbalpha`. The complete set is:

Uppercase Greek:

- `\bbGamma`
- `\bbDelta`
- `\bbTheta`
- `\bbLambda`
- `\bbXi`
- `\bbPi`
- `\bbSigma`
- `\bbUpsilon`
- `\bbPhi`
- `\bbPsi`

- \bbOmega

Lower Case Greek

- \bbalpha
- \bbbeta
- \bbgamma
- \bbdelta
- \bbepsilon
- \bbzeta
- \bbeta
- \bbtheta
- \bbiota
- \bbkappa
- \bblambda
- \bbmu
- \bbnu
- \bbxi
- \bbpi
- \bbrho
- \bbsigma
- \bbtau
- \bbupsilon
- \bbphi
- \bbchi
- \bbpsi
- \bbomega

Dotless i and j

- \bbdotlessi
- \bbdotlessj

If the *textmacros* extension is loaded, then the *bboldx* extension will define text-based versions of the Greek commands as well, prefixed by `txt` and `txtbf`. E.g., `\txtbbalpha` and `\txtbbfbbalpha` can be used in `\text{}` to get bboldx versions of the alpha in normal and bold styles. The dotless i and j can be obtained from `\itextbb`, `\itexstbfbb`, `\jtextbb`, and `\jtextbbf`.

The bboldx extension also creates the delimiters `\bbLparen`, `\bbRparen`, `\bbLbrack`, `\bbRbrack`, `\bbLangle`, and `\bbRangle`, but since the fonts don't include any of the larger versions or the pieces needed to make stretchy assemblies, these don't work with `\left` and `\right`, and will just produce the normal stretchy parentheses, brackets, and angle brackets.

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

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

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

---

bboldx Options

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

```
MathJax = {
  tex: {
    bboldx: {
      bfbb: false,
      light: false
    }
  }
};
```

`bfbb: false`

Specifies whether to use the bold-weight versions of the bboldx fonts.

`light: false`

Specifies whether to use the light-weight versions of the bboldx fonts.

---

bboldx Commands

The *bboldx* extension implements the following macros: `\bbalpha`, `\bbbeta`, `\bbchi`, `\bbDelta`, `\bbdelta`, `\bbdotlessi`, `\bbdotlessj`, `\bbepsilon`, `\bbeta`, `\bbGamma`, `\bbgamma`, `\bbiota`, `\bbkappa`, `\bbLambda`, `\bblambda`, `\bbLangle`, `\bbLbrack`, `\bbLparen`, `\bbmu`, `\bbnu`, `\bbOmega`, `\bbomega`, `\bbPhi`, `\bbphi`, `\bbPi`, `\bbpi`, `\bbPsi`, `\bbpsi`, `\bbRangle`, `\bbRbrack`, `\bbrho`, `\bbRparen`, `\bbSigma`, `\bbsigma`, `\bbtau`, `\bbTheta`, `\bbtheta`, `\bbUpsilon`, `\bbupsilon`, `\bbXi`, `\bbxi`, `\bbzeta`, `\bfbbalpha`, `\bfbbbeta`, `\bfbbchi`, `\bfbbDelta`, `\bfbbdelta`, `\bfbbdotlessi`, `\bfbbdotlessj`, `\bfbbepsilon`, `\bfbbeta`, `\bfbbGamma`, `\bfbbgamma`, `\bfbbiota`, `\bfbbkappa`, `\bfbbLambda`, `\bfbblambda`, `\bfbbLangle`, `\bfbbLbrack`, `\bfbbLparen`, `\bfbbmu`, `\bfbbnu`, `\bfbbOmega`, `\bfbbomega`, `\bfbbPhi`, `\bfbbphi`, `\bfbbPi`, `\bfbbpi`, `\bfbbPsi`, `\bfbbpsi`, `\bfbbRangle`, `\bfbbRbrack`, `\bfbbrho`, `\bfbbRparen`, `\bfbbSigma`, `\bfbbsigma`, `\bfbbtau`, `\bfbbTheta`, `\bfbbtheta`, `\bfbbUpsilon`, `\bfbbupsilon`, `\bfbbXi`, `\bfbbxi`, `\bfbbzeta`, `\imathbb`, `\imathbfbb`, `\jmathbb`, `\jmathbfbb`, `\mathbb`, `\mathbfbb`

The *bboldx* extension implements the following text-mode macros when the *textmacros* extension is loaded (these are only available inside `\text{}` or other text-mode macros): `\itextbb`, `\itextbfbb`, `\jtextbb`, `\jtextbfbb`, `\textbb`, `\textbfbb`, `\txtbbalpha`, `\txtbbbeta`, `\txtbbchi`, `\txtbbDelta`, `\txtbbdelta`, `\txtbbdotlessi`, `\txtbbdotlessj`, `\txtbbepsilon`, `\txtbbeta`, `\txtbbGamma`, `\txtbbgamma`, `\txtbbiota`, `\txtbbkappa`, `\txtbbLambda`, `\txtbblambda`, `\txtbbLangle`, `\txtbbLbrack`, `\txtbbLparen`, `\txtbbmu`, `\txtbbnu`, `\txtbbOmega`, `\txtbbomega`, `\txtbbPhi`, `\txtbbphi`, `\txtbbPi`, `\txtbbpi`, `\txtbbPsi`, `\txtbbpsi`, `\txtbbRangle`, `\txtbbRbrack`, `\txtbbrho`, `\txtbbRparen`, `\txtbbSigma`, `\txtbbsigma`, `\txtbbtau`, `\txtbbTheta`, `\txtbbtheta`, `\txtbbUpsilon`, `\txtbbupsilon`, `\txtbbXi`, `\txtbbxi`, `\txtbbzeta`,

\txtbfbbalpha, \txtbfbbbeta, \txtbfbbchi, \txtbfbbDelta, \txtbfbbdelta, \txtbfbbdotlessi, \txtbfbbdotlessj, \txtbfbbepsilon, \txtbfbbeta, \txtbfbbGamma, \txtbfbbgamma, \txtbfbbiota, \txtbfbbkappa, \txtbfbbLambda, \txtbfbblambda, \txtbfbbLangle, \txtbfbbLbrack, \txtbfbbLparen, \txtbfbbmu, \txtbfbbnu, \txtbfbbOmega, \txtbfbbomega, \txtbfbbPhi, \txtbfbbphi, \txtbfbbPi, \txtbfbbpi, \txtbfbbPsi, \txtbfbbpsi, \txtbfbbRangle, \txtbfbbRbrack, \txtbfbbrho, \txtbfbbRparen, \txtbfbbSigma, \txtbfbbsigma, \txtbfbbtau, \txtbfbbTheta, \txtbfbbtheta, \txtbfbbUpsilon, \txtbfbbupsilon, \txtbfbbXi, \txtbfbbxi, \txtbfbbzeta

24.7.7 bbox

The *bbox* extension defines a non-standard 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]{x+1}
                     % 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.

---

bbox Commands

The *bbox* extension implements the following macros: `\bbox`

24.7.8 begingroup

The *begingroup* extension implements commands that provide a mechanism for localizing macro definitions so that they are not permanent. This is useful if you have a blog site, for example, and want to isolate changes that your readers make in their comments so that they don't affect later comments.

It defines two new non-standard macros, `\begingroup` and `\endgroup`, that are used to start and stop a local namespace for macros. Any macros that are defined between the `\begingroup` and `\endgroup` will be removed after the `\endgroup` is executed. For example, if you put `\(\begingroup\)` at the top of each reader's comments and `\(\endgroup\)` at the end, then any macros they define within their response will be removed after it is processed. Note that `\begingroup` and `\endgroup` do not have to appear within the same expression.

In addition to these two macros, the *begingroup* extension defines the standard `\global` and `\gdef` control sequences from TeX. (The `\let`, `\def`, `\newcommand`, and `\newenvironment` control sequences are already defined in the *newcommand* package.)

The begingroup package also defines two non-standard macros: `\begingroupReset` and `\begingroupSandbox`. The first effectively applies `\endgroup` to any open `\begingroup` macros, returning the definitions to the ones that were in effect before the first `\begingroup`, plus any new `\global` definitions.

The `\begingroupSandbox` macro can be used to isolate the definitions in one section of the page from those in another, so that sites, such as question-and-anser sites, can use this between user posts to make sure that one user's definitions won't affect any other user's expressions. The `\begingroupSandbox` macro can't be redefined, and its action is to remove all the open `\begingroup` calls and then start a new local greoup (effectively doing `\begingroupReset\begingroup`). This removes any previous user definitions and makes a new group for the next user's definition. Furthermore, `\global` definitions are put into the new `\begingroup` so that even if the math includes `\global` definitions, they will be removed when the next `\begingroupSandbox` command is performed, and will not pollute the next user's macro namespace.

Any definitions made before the first `\begingroupSandbox` invocation is made will be available within all the sandboxes, but once `\begingroupSandbox` is performed, there is no going back; the original macro namespace is no longer accessible to new definitions.

The *begingroup* package is not autoloaded, so you must request it explicitly if you want to use it. To load the *begingroup* extension explicitly, add `'[tex]/begingroup'` to the `load` array of the `loader` block of your MathJax configuration, and add `'begingroup'` to the `packages` array of the `tex` block.

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

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

---

begingroup Options

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

```
MathJax = {
  tex: {
    begingroup: {
      allowGlobal: ["let", "def", "newcommand", "DeclareMathOperator", "Newextarrow"]
    }
  }
};
```

allowGlobal: ["let", "def", "newcommand", "DeclareMathOperator", "Newextarrow"]

This lists the commands that can follow the `\global` command. Extensions may add to this if they include new macros that can define global values. Alternatively, you can remove commands from the list to prevent users from making global definitions.

---

begingroup Commands

The *begingroup* extension implements the following macros: `\begingroup`, `\begingroupReset`, `\begingroupSandbox`, `\endgroup`, `\gdef`, `\global`

24.7.9 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.

---

boldsymbol Commands

The *boldsymbol* extension implements the following macros: `\boldsymbol`

24.7.10 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.

---

braket Commands

The *braket* extension implements the following macros: `\|`, `\bra`, `\Bra`, `\braket`, `\Braket`, `\ket`, `\Ket`, `\ketbra`, `\Ketbra`, `\set`, `\Set`, `|`

24.7.11 bussproofs

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

Note that there are several 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.

---

bussproofs Commands

The *bussproofs* extension implements the following macros: `\alwaysDashedLine`, `\alwaysNoLine`, `\alwaysRootAtBottom`, `\alwaysRootAtTop`, `\alwaysSingleLine`, `\alwaysSolidLine`, `\AXC`, `\Axiom`, `\AxiomC`, `\BIC`, `\BinaryInf`, `\BinaryInfC`, `\dashedLine`, `\fCenter`, `\LeftLabel`, `\LL`, `\noLine`, `\QuaternaryInf`, `\QuaternaryInfC`, `\QuinaryInf`, `\QuinaryInfC`, `\RightLabel`, `\RL`, `\rootAtBottom`, `\rootAtTop`, `\singleLine`, `\solidLine`, `\TIC`, `\TrinaryInf`, `\TrinaryInfC`, `\UIC`, `\UnaryInf`, `\UnaryInfC`

And the following environments: `prooftree`

24.7.12 cancel

The *cancel* extension implements the `cancel` package from LaTeX, which provides a means of "crossing out" sub-expressions in various ways. See the cancel CTAN page for more information and documentation. See also the *enclose* extension for some alternative approaches to crossing out sub-expressions.

This package 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.

---

cancel Commands

The *cancel* extension implements the following macros: `\bcancel`, `\cancel`, `\cancelto`, `\xcancel`

24.7.13 cases

The *cases* extension implements the `cases` style package from LaTeX. It provides environments `numcases` and `subnumcases` for formulas with separately enumerated cases. See the cases CTAN page for more information and documentation.

An example of the `numcases` environment is:

```
\begin{numcases} {|x|=}
  x, & for $x \geq 0$\\
  -x, & for $x < 0$
\end{numcases}
```

This package is not autoloaded, so you must request it explicitly if you want to use it. To load the *cases* extension, add `'[tex]/cases', '[tex]/empheq'` to the `load` array of the `loader` block of your MathJax configuration, and add `'cases', 'empheq'` to the `packages` array of the `tex` block. Note that the *empheq* package is required when you load *cases*.

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

You can configure the *autoload* extension to load *cases* via

```
window.MathJax = {
  tex: {
    autoload: {
      cases: [[], ['numcases', 'subnumcases']]
    }
  }
};
```

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

---

cases Commands

The *cases* extension implements the following macros: `&`

And the following environments: `numcases`, `subnumcases`

24.7.14 centernot

The *centernot* extension implements the `centernot` style package from LaTeX. It provides the `\centernot` command which can be used as a replacement of the standard `\not` command and generally leads to a better alignment of the slash with the operator it negates. This can be observed with the following two examples:

```
\begin{array}{c}
  A \not\longrightarrow B\\
  A \centernot\longrightarrow B
\end{array}
```

```
\begin{array}{c}
  A \nparallel B\\
  A \not\parallel B\\
  A \centernot\parallel B
\end{array}
```

See also the centernot CTAN page for more information and documentation.

In addition to `\centernot` the package also implements the non-standard `\centerOver`.

`\centerOver{symbol1}{symbol2}`

Overlays `symbol2` centered on top of `symbol1`. The result has the width and TeX class of `symbol1`.

For example, the following produces a circle around a triangle:

```
\centerOver{\bigcirc}{\small\triangle}
```

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

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

You can configure the *autoload* extension to load *centernot* via

```
window.MathJax = {
  tex: {
    autoload: {
      centernot: ['centernot', 'centerOver']
    }
  }
};
```

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

centernot Commands

The *centernot* extension implements the following macros: `\centernot`, `\centerOver`

24.7.15 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`.

---

color Commands

The *color* extension implements the following macros: `\color`, `\colorbox`, `\definecolor`, `\fcolorbox`, `\textcolor`

24.7.16 colortbl

The *colortbl* extension partially implements the `colortbl` style package from LaTeX. It allows coloring of rows, columns, and individual cell of tables. See the colortbl CTAN page for more information and documentation. Note that MathJax currently does not implement any commands for styling or coloring table boundaries. In addition, MathJax implement some of the *colortbl* commands differently:

\rowcolor[model]{color}

Specifies the color for a single row in a table. It needs to be used in the first cell of a row to color. If used elsewhere, it will produce an error.

\columncolor[model]{color}

Specifies the color for a single column. It needs to be used in the first **cell** of the column to color, or in the table or array preamble. If used elsewhere, it will produce an error.

In addition overhang arguments are currently not handled. That is MathJax ignores up to two optional bracketed arguments after the mandatory color argument.

\cellcolor[model]{color}

Allows to color a single cell. It can be used anywhere in the cell to color.

> **Note**
>
> In version 3, the `\columncolor` macro could not be used in the preamble for the table or array, but had to be in the cell of the column in the first row of the table. In v4, you can use `\columncolor` in the preamble, but for backward compatibility, you can use it in the first row as well.

The order of precedence of the color commands is as follows: `\cellcolor` > `\rowcolor` > `\columncolor`. See the example below for all three commands in action.

```
\begin{array}{|>{\columncolor{red}}c|c|}
  \rowcolor[gray]{.5} one & two\\
  \rowcolor{lightblue} three & four\\\hline
  five & six \\
  \rowcolor{magenta} \cellcolor{green} \color{white}seven &eight
\end{array}
```

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

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

You can configure the *autoload* extension to load *colortbl* via

```
window.MathJax = {
  tex: {
    autoload: {
      colortbl: ['cellcolor', 'columncolor', 'rowcolor']
    }
  }
};
```

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

---

colortbl Commands

The *colortbl* extension implements the following macros: `\cellcolor`, `\columncolor`, `\rowcolor`

24.7.17 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 *colorv2* extension explicitly, add `'[tex]/colorv2'` to the `load` array of the `loader` block of your MathJax configuration, and add `'colorv2'` to the `packages` array of the `tex` block.

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

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* extension 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
    }
  }
};
```

colorv2 Commands

The *colorv2* extension implements the following macros: `\color`

24.7.18 configmacros

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

```
window.MathJax = {
  tex: {
    macros: {
      RR: "{\\bf R}",
      bold: ["{\\bf #1}", 1]
    },
    environments: {
      braced: ["\\left\\{", "\\right\\}"]
    },
    active: {
      '*': ["#1 \times #2", 2]
    }
  }
};
```

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 a `braced` environment that puts `\left\{` and `\right\}` around its contents. Finally, it makes `*` an active character that acts like a macro (without the need for a backslash), which takes two arguments and puts a "times" symbol between them.

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, a `environments` option that lets you pre-define your own environments, and an `active` option that lets you define active characters.

`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. Alternatively, you can use the `String.raw` syntax to create the string literals with single backslashes.

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. Alternatively, you can use the `String.raw` syntax to create the string literals with single backslashes.

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)`.

active: {}

This lists active characters to define before the TeX input processor begins. These are *name: value* pairs where the *name* gives the (single) character to be defined, and *value* gives the replacement text for the active character. 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. This works the same as for the `macros` assignments above.

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. Alternatively, you can use the `String.raw` syntax to create the string literals with single backslashes.

For example,

```
active: {
  '*': ['#1 \\times #2', 2],
  '+': '\\boldsymbol{\\char`+}'
}
```

makes * be a macro that typesets a times symbol between the two arguments that follow it, while + will produce a bold plus sign. Note that you don't want to use the symbol you are defining as part of the definition, as that would cause an infinite loop. Instead, we use the `\char` macro to insert the plus sign rather than using + directly.

24.7.19 dsfont

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

This extension defines the following commands:

\mathds{math}

Typesets `math` using the dsfont blackboard-bold fonts.

The dsfonts come in two forms: serifed and san-serif; which one is used is determined by an option described :ref`below <tex-dsfont-options>`.

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

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

(continues on next page)

(continued from previous page)

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

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

---

dsfont Options

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

```
MathJax = {
  tex: {
    dsfont: {
      sans: false
    }
  }
};
```

`sans: false`
: Determines whether the sans-serif font is used rather than the serifed version.

---

dsfont Commands

The *dsfont* extension implements the following macros: `\mathds`

24.7.20 empheq

The *empheq* extension partially implements the `empheq` style package from LaTeX. The package provides macros and environments for emphasising equations. See the *list of control sequences* for details about what commands are implemented in this extension. Note, that the current implementation of the `empheq` environment supports only the `left` and `right` options. Also see the empheq CTAN page for more information and documentation.

As an example, you could do:

```
\begin{empheq}[left=\empheqlbrace, right=\empheqrbrace]{align}
E&=mc^2 \\
Y&= \sum_{n=1}^\infty \frac{1}{n^2}
\end{empheq}
```

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

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

You can configure the *autoload* extension to load *empheq* when the `empheq` environment is first used via

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

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

---

empheq Commands

The *empheq* extension implements the following macros: `\empheqbigl`, `\empheqbiglangle`, `\empheqbiglbrace`, `\empheqbiglbrack`, `\empheqbiglceil`, `\empheqbiglfloor`, `\empheqbiglparen`, `\empheqbiglvert`, `\empheqbiglVert`, `\empheqbigr`, `\empheqbigrangle`, `\empheqbigrbrace`, `\empheqbigrbrack`, `\empheqbigrceil`, `\empheqbigrfloor`, `\empheqbigrparen`, `\empheqbigrvert`, `\empheqbigrVert`, `\empheql`, `\empheqlangle`, `\empheqlbrace`, `\empheqlbrack`, `\empheqlceil`, `\empheqlfloor`, `\empheqlparen`, `\empheqlvert`, `\empheqlVert`, `\empheqr`, `\empheqrangle`, `\empheqrbrace`, `\empheqrbrack`, `\empheqrceil`, `\empheqrfloor`, `\empheqrparen`, `\empheqrvert`, `\empheqrVert`

And the following environments: `empheq`

24.7.21 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 optional 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.

enclose Commands

The *enclose* extension implements the following macros: `\enclose`

24.7.22 extpfeil

The *extpfeil* extension adds 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.

extpfeil Commands

The *extpfeil* extension implements the following macros: `\Newextarrow`, `\xlongequal`, `\xmapsto`, `\xtofrom`, `\xtwoheadleftarrow`, `\xtwoheadrightarrow`

24.7.23 fontsizev3

The *fontsizev3* extension (new in v4.1.2) re-defines the various font-sizing commands (like `\tiny`, `\small`, `\large`, etc.) to be the sizes that they were in MathJax v3 and in v4 prior to v4.1.2, when they were corrected to be consistent with the sizes used in actual LaTeX. This extension is intended for those sites that have legacy content where the old sizing values are critical to the layout.

This extension is **not** loaded automatically when the *autoload* extension is used. To load the *fontsizev3* extension explicitly, add `'[tex]/fontsizev3'` to the `load` array of the `loader` block of your MathJax configuration, and add `'fontsizev3'` to the `packages` array of the `tex` block.

```
window.MathJax = {
  loader: {load: ['[tex]/fontsizesv3']},
  tex: {packages: {'[+]': ['fontsizesv3']}}
};
```

or, use `\require{fontsizesv3}` in a TeX expression to load it dynamically from within the math on the page, if the *require* extension is loaded.

---

fontsizesv3 Commands

The *fontsizesv3* extension implements the following macros: `\Tiny`, `\tiny`, `\scriptsize`, `\small`, `\normalsize`, `\large`, `\Large`, `\LARGE`, `\huge`, `\Huge`

24.7.24 gensymb

The *gensymb* extension implements the `gensymb` style package from LaTeX. It provides a number of macros for unit notation. See the CTAN page for more information and documentation for *gensymb*.

Note that the `mathjax-tex` font (the original default font in v2 and v3) does not include several of these characters, so the output will depend on the system fonts available to your reader, and will vary from reader to reader.

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

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

You can configure the *autoload* extension to load *gensymb* via

```
window.MathJax = {
  tex: {
    autoload: {
      gensymb: ['celsius', 'degree', 'micro', 'ohm', 'perthousand']
    }
  }
};
```

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

---

gensymb Commands

The *gensymb* extension implements the following macros: `\celsius`, `\degree`, `\micro`, `\ohm`, `\perthousand`

24.7.25 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 given `css` declarations to the element associated with `math`.

\data{dataset}{math}

Treats the `dataset` as a comma-separated list of *name*=*value* pairs and adds `data-name` attributes with the given values to the typeset `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)}}

\data{special=true}{x}  % output will have attribute data-special="true"
```

Note

For the \href macro, the *url* parameter is not processed further, 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.

Warning

You should not add styles or other values that change the size of the typeset math, other than via explicit borders or padding values. MathJax needs to know the dimensions of the math it typesets in order to properly lay out any surrounding math (like fraction bars or square root symbols), and so it will not be able to take into account changes that occur due to inherited CSS, or by CSS values other than border and padding.

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.

---

html Commands

The *html* extension implements the following macros: `\class`, `\cssId`, `\data`, `\href`, `\style`

24.7.26 mathtools

The *mathtools* extension implements the `mathtools` style package from LaTeX. The package provides a number of tools for advanced mathematical typesetting. See the mathtools CTAN page for more information and documentation.

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

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

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

---

mathtools Options

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

```
MathJax = {
  tex: {
    mathtools: {
      multline-gap: '1em',
      multlined-pos: 'c',
      multlined-width: '',
      firstline-afterskip: '',
      lastline-preskip: '',
      smallmatrix-align: 'c',
      shortvdotsadjustabove: '.2em',
      shortvdotsadjustbelow: '.2em',
      centercolon: false,
      centercolon-offset: '.04em',
      thincolon-dx: '-.04em',
      thincolon-dw: '-.08em',
```

(continues on next page)

(continued from previous page)

```
      use-unicode: false,
      legacycolonsymbols: false,
      prescript-sub-format: '',
      prescript-sup-format: '',
      prescript-arg-format: '',
      allow-mathtoolsset: true,
      pairedDelimiters: {},
      tagforms: {}
    }
  }
};
```

multline-gap: '1em'

Horizontal space for multlined environments. Note that in version 3, this was `multlinegap` without the dash.

multlined-pos: 'c'

Default alignment for multlined environments.

multlined-width: ''

The default width for miltlined environments.

firstline-afterskip: ''

Space for first line of multlined (overrides multlinegap).

lastline-preskip: ''

Space for last line of multlined (overrides multlinegap).

smallmatrix-align: 'c'

Default alignment for smallmatrix environments.

shortvdotsadjustabove: '.2em'

Space to remove above `\shortvdots`.

shortvdotsadjustbelow: '.2em'

Space to remove below `\shortvdots`.

centercolon: false

True to have colon automatically centered.

centercolon-offset: '.04em'

Vertical adjustment for centered colons.

thincolon-dx: '-.04em'

Horizontal adjustment for thin colons (e.g., `\coloneqq`).

thincolon-dw: '-.08em'

Width adjustment for thin colons.

use-unicode: false

True to use unicode characters rather than multi-character version for `\coloneqq`, etc., when possible.

legacycolonsymbols: false

The 2022 update to the LaTeX mathtools package changed the definitions of `\coloneq` and three other related macros. Setting this option to `true` will cause MathJax to use the older definitions rather than the ones from 2022 and later.

prescript-sub-format: ''

Format for \prescript subscript.

prescript-sup-format: ''

Format for \prescript superscript.

prescript-arg-format: ''

Format for \prescript base.

allow-mathtoolsset: true

True to allow \mathtoolsset to change settings.

pairedDelimiters: {}

Predefined paired delimiters of the form `name: [left, right, body, argcount, pre, post]`.

tagforms: {}

Tag form definitions of the form `name: [left, right, format]`.

---

mathtools Commands

The *mathtools* extension implements the following macros: \:, \Aboxed, \adjustlimits, \approxcolon, \Approxcolon, \ArrowBetweenLines, \bigtimes, \clap, \colonapprox, \Colonapprox, \colondash, \Colondash, \coloneq, \Coloneq, \coloneqq, \Coloneqq, \colonsim, \Colonsim, \cramped, \crampedclap, \crampedllap, \crampedrlap, \crampedsubstack, \dashcolon, \Dashcolon, \dblcolon, \DeclarePairedDelimiter, \DeclarePairedDelimiters, \DeclarePairedDelimitersX, \DeclarePairedDelimitersXPP, \DeclarePairedDelimiterX, \DeclarePairedDelimiterXPP, \eqcolon, \Eqcolon, \eqqcolon, \Eqqcolon, \lparen, \MakeAboxedCommand, \mathclap, \mathllap, \mathmakebox, \mathmbox, \mathrlap, \mathtoolsset, \MoveEqLeft, \MTFlushSpaceAbove, \MTFlushSpaceBelow, \MTThinColon, \ndownarrow, \newtagform, \nuparrow, \ordinarycolon, \overbracket, \prescript, \refeq, \renewtagform, \rparen, \shortvdotswithin, \shoveleft, \shoveright, \simcolon, \Simcolon, \splitdfrac, \splitfrac, \textclap, \textllap, \textrlap, \underbracket, \usetagform, \vcentercolon, \vdotswithin, \xhookleftarrow, \xhookrightarrow, \xLeftarrow, \xleftharpoondown, \xleftharpoonup, \xleftrightarrow, \xLeftrightarrow, \xleftrightharpoons, \xlongleftarrow, \xLongleftarrow, \xlongrightarrow, \xLongrightarrow, \xmapsto, \xmathstrut, \xRightarrow, \xrightharpoondown, \xrightharpoonup, \xrightleftharpoons

And the following environments: bmatrix*, Bmatrix*, bsmallmatrix*, Bsmallmatrix*, bsmallmatrix, Bsmallmatrix, cases*, crampedsubarray, dcases*, dcases, drcases*, drcases, lgathered, matrix*, multlined, pmatrix*, psmallmatrix*, psmallmatrix, rcases*, rcases, rgathered, smallmatrix*, spreadlines, vmatrix*, Vmatrix*, vsmallmatrix*, Vsmallmatrix*, vsmallmatrix, Vsmallmatrix

24.7.27 mhchem

The *mhchem* extension 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{[AgCl2]-}
\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 written for MathJax by the author of the original LaTeX package. An older version was available MathJax version 2.7, but it is no longer part of MathJax version 3 and above. Only the newer version of *mhchem* is available.

---

mhchem Commands

The *mhchem* extension implements the following macros: `\ce`, `\mhchemBondDTD`, `\mhchemBondTD`, `\mhchemBondTDD`, `\mhchemleftarrow`, `\mhchemleftrightarrow`, `\mhchemlongleftarrow`, `\mhchemlongleftrightarrow`, `\mhchemlongleftrightarrows`, `\mhchemlongLeftrightharpoons`, `\mhchemlongrightarrow`, `\mhchemlongrightleftharpoons`, `\mhchemlongRightleftharpoons`, `\mhchemrightarrow`, `\mhchemxleftarrow`, `\mhchemxleftrightarrow`, `\mhchemxleftrightarrows`, `\mhchemxLeftrightharpoons`, `\mhchemxrightarrow`, `\mhchemxrightleftharpoons`, `\mhchemxRightleftharpoons`, `\pu`, `\tripledash`

24.7.28 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, and for mechanisms for pre-defining macros at startup.

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* extension is loaded.

Since the *newcommand* 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']}}
};
```

---

newcommand Options

When the *newcommand* extension is added to the `packages` array for the `tex` block of your MathJax configuration (as it is in all the combined components), two new options are made available in the `tex` block:

```
MathJax = {
  tex: {
    maxMacros: 10000,                       // maximum number of macro substitutions␣
↪per expression
    protectedMacros: ['begingroupSandbox'],  // macros that can't be redefined
  }
};
```

`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.

`protectedMacros: ['begingroupSandbox']`

This array lists the macro names that can't be redefined by `\let`, `\def`, `\newcommand`, or other commands the define TeX control sequences. For example, in a question-and-answer website where users can enter mathemtical expressions, this protects the listed macro from being overwritten by a user, possibly interfering with another user. See the *begingroup* for more on isolating users from one another.

---

newcommand Commands

The *newcommand* extension implements the following macros: `\def`, `\let`, `\newcommand`, `\newenvironment`, `\renewcommand`, `\renewenvironment`

24.7.29 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 and above, 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']}}
};
```

24.7.30 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.

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']},
  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 string to have no background color.

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

24.7.31 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* extension is loaded.

---

physics Options

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

```
MathJax = {
  tex: {
    physics: {
      italicdiff: false,
      arrowdel: false
    }
  }
};
```

italicdiff: false

This corresponds to the `italicdiff` option of the *physics* LaTeX package to use italic form for the *d* in the `\differential` and `\derivative` commands.

arrowdel: false

This corresponds to the `arrowdel` option of the *physics* LaTeX package to use vector notation over the nabla symbol.

Note, that the *physics* extension does not implement the *notrig* option.

---

physics Commands

The *physics* extension implements the following macros: `)`, `]`, `\abs`, `\absolutevalue`, `\acomm`, `\acos`, `\acosecant`, `\acosine`, `\acot`, `\acotangent`, `\acsc`, `\admat`, `\anticommutator`, `\antidiagonalmatrix`, `\arccos`, `\arccosecant`, `\arccosine`, `\arccot`, `\arccotangent`, `\arccsc`, `\arcsec`, `\arcsecant`, `\arcsin`, `\arcsine`, `\arctan`, `\arctangent`, `\asec`, `\asecant`, `\asin`, `\asine`, `\atan`, `\atangent`, `\bmqty`, `\bqty`, `\Bqty`, `\bra`, `\braket`, `\comm`, `\commutator`, `\cos`, `\cosecant`, `\cosh`, `\cosine`, `\cot`, `\cotangent`, `\coth`, `\cp`, `\cross`, `\crossproduct`, `\csc`, `\csch`, `\curl`, `\dd`, `\derivative`, `\det`, `\determinant`, `\diagonalmatrix`, `\diffd`, `\differential`, `\div`, `\divergence`, `\divisionsymbol`, `\divsymbol`, `\dmat`, `\dotproduct`, `\dv`, `\dyad`, `\erf`, `\ev`, `\eval`, `\evaluated`, `\exp`, `\expectationvalue`, `\exponential`, `\expval`, `\fderivative`, `\fdv`, `\flatfrac`, `\functionalderivative`, `\grad`, `\gradient`, `\gradientnabla`, `\hypcosecant`, `\hypcosine`, `\hypcotangent`, `\hypsecant`, `\hypsine`, `\hyptangent`, `\identitymatrix`, `\Im`, `\imaginary`, `\imat`, `\innerproduct`, `\ip`, `\ket`, `\ketbra`, `\laplacian`, `\ln`, `\log`, `\logarithm`, `\matrixdeterminant`, `\matrixel`, `\matrixelement`, `\matrixquantity`, `\mdet`, `\mel`, `\mqty`, `\naturallogarithm`, `\norm`, `\op`, `\order`, `\outerproduct`, `\partialderivative`, `\paulimatrix`, `\pb`, `\pderivative`, `\pdv`, `\pmat`, `\pmqty`, `\Pmqty`, `\poissonbracket`, `\pqty`, `\Pr`, `\principalvalue`, `\Probability`, `\pv`, `\PV`, `\qall`, `\qand`, `\qas`, `\qassume`, `\qc`, `\qcc`, `\qcomma`, `\qelse`, `\qeven`, `\qfor`, `\qgiven`, `\qif`, `\qin`, `\qinteger`, `\qlet`, `\qodd`, `\qor`, `\qotherwise`, `\qq`, `\qqtext`, `\qsince`, `\qthen`, `\qty`, `\quantity`, `\qunless`, `\qusing`, `\rank`, `\Re`, `\real`, `\Res`, `\Residue`, `\sbmqty`, `\sec`, `\secant`, `\sech`, `\sin`, `\sine`, `\sinh`, `\smallmatrixquantity`, `\smdet`, `\smqty`, `\spmqty`, `\sPmqty`, `\svmqty`, `\tan`, `\tangent`, `\tanh`, `\tr`, `\Tr`, `\trace`, `\Trace`, `\va`, `\var`, `\variation`, `\vb`, `\vdot`, `\vectorarrow`, `\vectorbold`, `\vectorunit`, `\vmqty`, `\vnabla`, `\vqty`, `\vu`, `\xmat`, `\xmatrix`, `\zeromatrix`, `\zmat`, `|`

And the following environments: `smallmatrix`

24.7.32 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,
        autoload: false,
        configmacros: false,
        tagformat: false,
        setoptions: false,
        texhtml: false,
      },
      defaultAllow: true,
      prefix: 'tex'
    }
  };
```

`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.

`prefix: 'tex'`

The prefix to use when creating the component name for the extension. By default, the `prefix` is `tex`, so `\require{bbox}` will load `[tex]/bbox`.

require Commands

The *require* extension implements the following macros: `\require`

24.7.33 setoptions

The *setoptions* extension implements a non-standard `\setOptions` macro that allows you to change the options for a TeX package, or for the TeX input jax itself, from within a TeX expression.

`\setOptions[package]{options}`

> Sets the options for *package* to the ones given in *options*. Here, *options* is a collection of space- or comma-separated option names (to be set to `true`) or *option*=*value* declarations, where the given option will get the specified value. If the value contains spaces or commas, it can be enclosed in braces, which will not become part of the value.

For example:

```
\[
  \setOptions{tagSide=left}
  E = mc^2 \tag{1}
\]

\[
  \setOptions{tagSide=right}
  e^{\pi i} + 1 = 0 \tag{2}
\]
```

would typeset the first expression with its tag on the left, and the second (and subsequent) expressions with tags on the right.

To change a package setting, use the package name as an optional bracket argument:

```
\[
  \setOptions[physics]{arrowdel=true}
  \grad
  \setOptions[physics]{arrowdel=false}
\]
```

Here the gradient symbol will have an arrow, but subsequent ones will not.

Note that any changes made by `\setOptions` are global, so will affect all the following expressions. If you want a local change, you will need to set the value back to its original one explicitly, as in the example above.

Because changing the option settings can cause adverse consequences, and so could be misused in a setting where users are providing the TeX content for your site, the *setoptions* package is not autoloaded, and it can not be loaded with `\require{}`. You must include it in the package list explicitly if you want to allow its use.

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

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

The require command with setoptions

If the *require* package is enabled, *setoptions* modifies `\require` to allow passing of options for the required package (and makes the original `\require` macro available as `\Require`). So the new syntax is:

`\require[options]{package}`

where *options* is a list of options in the same format as used by `\setOptions`, and `package` is the name of the extension to load. This command is equivalent to:

```
\Require{package}\setOptions[package]{options}
```

meaning that the package is loaded and then its options are set.

For example:

```
\require[harrowsize=3em]{amscd}
```

would load the *amscd* extension and then set its `harrowsize` option to `3em`.

Note that the same rules apply to which options can be set for which package as those that govern `\setOptions` itself.

setoptions Options

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

```
MathJax = {
  tex: {
    setoptions: {
      filterPackage: SetOptionsUtil.filterPackage,  // filter for whether a package can␣
↪be configured
      filterOption: SetOptionsUtil.filterOption,    // filter for whether an option can␣
↪be set
      filterValue: SetOptionsUtil.filterValue,      // filter for the value to assign to␣
↪an option
      allowPackageDefault: true,       // default for allowing packages when not␣
↪explicitly set in allowOptions
      allowOptionsDefault: true,       // default for allowing option that isn't␣
↪explicitly set in allowOptions
      allowOptions: {                  // list of packages to allow/disallow, and their␣
↪options to allow/disallow
        //
        //  top-level tex items can be set, but not these ones
        //    (that leaves digits and the tagging options that can be set)
        //
        tex: {
          FindTeX: false,
          formatError: false,
          package: false,
          baseURL: false,
          tags: false,          // would require a new TeX input jax instance
```

(continues on next page)

(continued from previous page)

```
        maxBuffer: false,
        maxMaxros: false,
        macros: false,
        environments: false
      },
      //
      // These packages can't be configured at all
      //
      setoptions: false,
      autoload: false,
      require: false,
      configmacros: false,
      tagformat: false
    }
  }
}
};
```

filterPackage: SetOptionsUtil.filterPackage

This is a function that is called to determine if a package can have its options set or not. It is passed the TeX parser and the name of the extension as its arguments, and returns true if the package allows its options to be configured and false otherwise. The default is to first check that the named package exists, then check if the package is explicitly allowed by its entry in the `allowOptions` configuration option. That entry can either be `true`, allowing all options of the package to be set, or a list of the options that are allowed to be set, or `false` to mean that no options can be set for that package. If the package is not in the `allowOptions` list, then the value of the `allowPackageDefault` option is used. If that value is not `false`, an error is issued. You can supply your own function to process the package names in another way if you wish.

filterOption: SetOptionsUtil.filterOption

This is a function that is called to determine whether an option can be set for a given package. It is passed the TeX parser, the package name, and the option name as its arguments, and returns true if that option can be set for that package, and false otherwise. The default is to check if the option is listed explicitly in the list of options for the given package in the `allowOptions` list. If the value is explicitly `false`, or if it is not listed and the `allowOptionDefault` is `false`, then produce an error. Otherwise check that the option actually exists for the package, and report an error if not, otherwise allow the option to be set. You can supply your own function to process the option names in another way if you wish.

filterValue: SetOptionsUtil.filterValue

This is a function that is called to check whether the value provided for a given option is allowed. It is passed the TeX parser, the package name, the option name, and the new option value as its arguments, and it returns the value to be used for the option. The default is simply to return the value it is given, but you can use this to alter the value, or to produce an error if the value is not valid.

allowPackageDefault: true

This indicates how to handle packages that are not listed explicitly in the `allowOptions` list. If `true`, packages that are not listed are allowed to have their options set. If the value is `false`, only the packages that are listed as `true` or have explicit option lists can have their options set.

allowOptionsDefault: true

This indicates how to handle options that are not listed explicitly in the `allowOptions` list for a given package. If `true`, options that are not listed are allowed to be set, and if `false`, only the options that are listed explicitly as `true` for the given package can have their options set.

allowOptions: {...}

This is a list of the packages that indicates whether their options can be set or not, and which options can be set. If a package name appears and is explicitly set to `false`, that package's options can't be set. If it is `true` and `allowOptionsDefault` is true, then any of its options can be set. If it is an explicit list of options, then if the option is listed as `true`, it can be set, and if `false` it can not. If an option is not listed, then the value of `allowOptionsDefault` is used to determine whether it can be set or not. If a package does not appear explicitly in the list, then the value of `allowPackageDefault` is used to determine if the package's options can be set or not.

You can include additional package names and their options in this list. The defaults are set to allow reasonable security without having to list every single option that can be set.

---

setoptions Commands

The *setoptions* extension implements the following macros: `\setOptions`

24.7.34 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 below.

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 + ')',
  ref: '', // means use the tag function
  id: (id) => 'mjx-eqn:' + id.replace(/\s/g, '_'),
  url: (id, base) => base + '#' + encodeURIComponent(id),
}
```

number: (n) => 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, for example.

tag: (tag) => '(' + tag + ')'

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.

ref: ''

A function that tells MathJax how to format a reference (e.g., from `\ref`) to an equation number. If set to an empty string, MathJax will call the `tag` function to get this value.

id: (id) => 'mjx-eqn:' + id.replace(/\s/g, '_')

A function that tells MathJax what *id* attribute to use as an anchor for the equation (so that it can be used in URL references).

url: (id, base) => 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: {
    tags: 'ams',
    packages: {'[+]': ['tagformat']},
    tagformat: {
      number: (n) => MathJax.config.section + '.' + n,
      id: (tag) => 'eqn-id:' + tag
    },
    preFilters: [
      ({math}) => {
        if (math.inputData.recompile) {
          MathJax.config.section = math.inputData.recompile.section;
        }
      }
    ],
    postFilters: [
      ({math}) => {
        if (math.inputData.recompile) {
          math.inputData.recompile.section = MathJax.config.section;
        }
      }
    ]
  },
  loader: {load: ['[tex]/tagformat']},
};
```

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.

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 = {
  tex: {
    packages: {'[+]': ['sections']},
  },
  startup: {
    ready() {
      const Configuration = MathJax._.input.tex.Configuration.Configuration;
      const CommandMap = MathJax._.input.tex.TokenMap.CommandMap;
      new CommandMap('sections', {
        nextSection: 'NextSection',
        setSection: 'SetSection',
      }, {
        NextSection(parser, _name) {
          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 configuration in with the rest of your configuration options. You also will need to load the new `sections` configuration by adding it to your package list loaded into the TeX input jax.

```
{packages: {'[+]': ['tagformat', 'sections']}}
```

Warning

In v4 the `SymbolMap` has been renamed `TokenMap`, and that has been changed in the example above from the v3 version.

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: none">\(\nextSection\)</span>
```

or something similar.

Here is a complete example HTML document:

```
<!DOCTYPE html>
<html>
```

(continues on next page)

(continued from previous page)

```
<head>
<title>Section numbering example</title>
<script>
MathJax = {
  section: 1,
  tex: {
    tags: 'ams',
    packages: {'[+]': ['tagformat', 'sections']},
    tagformat: {
      number: (n) => MathJax.config.section + '.' + n,
      id: (tag) => 'eqn-id:' + tag
    },
    preFilters: [
      ({math}) => {
        if (math.inputData.recompile) {
          MathJax.config.section = math.inputData.recompile.section;
        }
      }
    ],
    postFilters: [
      ({math}) => {
        if (math.inputData.recompile) {
          math.inputData.recompile.section = MathJax.config.section;
        }
      }
    ]
  },
  loader: {load: ['[tex]/tagformat']},
  startup: {
    ready() {
      const Configuration = MathJax._.input.tex.Configuration.Configuration;
      const CommandMap = MathJax._.input.tex.TokenMap.CommandMap;
      new CommandMap('sections', {
        nextSection(parser, name) {
          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();
    }
  }
};</script>
<script src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-chtml.js"></script>
</head>
<body>
```

(continues on next page)

(continued from previous page)

```
<h1>Section 1</h1>

<p>
Equations in section 1:

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

and

\begin{equation}\label{complex}
e^{\pi i} + 1 = 0
\end{equation}
</p>
<p>
That is the end of section 1.
</p>

<hr/>

<h1>Section 2</h1>
<span style="display: none">\(\nextSection\)</span>

<p>
Equations in section 2:

\begin{equation}
y = \sqrt{1-x^2}
\end{equation}

and

\begin{equation}
\sum_{i=1}^n i = \frac{n(n+1)}{2}
\end{equation}
</p>
<p>
References to equations include section numbers: \ref{complex}.
</p>

</body>
</html>
```

24.7.35 texhtml

Normally, MathJax will not process math that includes HTML tags, other than a few exceptions like <br> and HTML comments. The *texhtml* extension provides a method of including HTML tags within TeX expressions. For example, this can allow you to include form inputs (like answer blanks or drop-down menus) in your expressions.

Warning

Because the HTML within TeX expressions is not filtered in any way by MathJax, you should **not** use this extension in settings where your readers can enter expressions that are shown to other users. That would allow them to enter maliceous code (including script tags) that could compromise the security of your site.

For this reason, you must not only load the *texhtml* extension, but must set the `allowTexHTML` option in the `tex` block of your MathJax configuration, as described in the configuration example below.

To include HTML tags within your expressions, enclose the HTML within the special `<tex-html>...</tex-html>` tags. For example,

```
when \(x=3\), \(x^2 + 3x + 1 =
<tex-html><input type="text" id="answer" size="5"/></tex-html>\).
```

You could then use javascript to retrieve the value of the input element with `id="answer"` and test if the solution is correct.

See the *Specifying the size of HTML in Expressions* section for information about how MathJax determines the size of HTML embedded in TeX expressions.

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

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

Note that the *texhtml* extension is not allowed to be loaded with `\require{texhtml}`.

---

texhtml Commands

The *texhtml* extension implements the following macros: <

24.7.36 textcomp

The *textcomp* extension implements the old `textcomp` style package from LaTeX. The macros of the package are now part of LaTeX's base package, but because MathJax concentrates on mathematical typesetting, not text, they remain in a separate package in MathJax. The *textcomp* extension provides a number of text macros that can be used in math mode as well. See the textcomp CTAN page for more information and documentation.

The macros provided in *textcomp* can be used equally in math and text mode. In order to make them available in text mode, you need to load the *textmacros* extension, and add the `'textcomp'` package to the `packages` array of the `textmacros` options of the `tex` block of you configuration. Note that *textmacros* is already included in the components that include `input/tex`.

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

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

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

---

textcomp Commands

The *textcomp* extension implements the following macros: \textacutedbl, \textasciiacute, \textasciibreve, \textasciicaron, \textasciicircum, \textasciidieresis, \textasciimacron, \textasciitilde, \textasteriskcentered, \textbackslash, \textbaht, \textbar, \textbardbl, \textbigcircle, \textblank, \textborn, \textbraceleft, \textbraceright, \textbrokenbar, \textbullet, \textcelsius, \textcent, \textcentoldstyle, \textcircledP, \textcolonmonetary, \textcompwordmark, \textcopyleft, \textcopyright, \textcurrency, \textdagger, \textdaggerdbl, \textdegree, \textdied, \textdiscount, \textdiv, \textdivorced, \textdollar, \textdollaroldstyle, \textdong, \textdownarrow, \texteightoldstyle, \textellipsis, \textemdash, \textendash, \textestimated, \texteuro, \textexclamdown, \textfiveoldstyle, \textflorin, \textfouroldstyle, \textfractionsolidus, \textgravedbl, \textgreater, \textguarani, \textinterrobang, \textinterrobangdown, \textlangle, \textlbrackdbl, \textleftarrow, \textless, \textlira, \textlnot, \textlquill, \textmarried, \textmho, \textminus, \textmu, \textmusicalnote, \textnaira, \textnineoldstyle, \textnumero, \textohm, \textonehalf, \textoneoldstyle, \textonequarter, \textonesuperior, \textopenbullet, \textordfeminine, \textordmasculine, \textparagraph, \textperiodcentered, \textpertenthousand, \textperthousand, \textpeso, \textpm, \textquestiondown, \textquotedblleft, \textquotedblright, \textquoteleft, \textquoteright, \textrangle, \textrbrackdbl, \textrecipe, \textreferencemark, \textregistered, \textrightarrow, \textrquill, \textsection, \textservicemark, \textsevenoldstyle, \textsixoldstyle, \textsterling, \textsurd, \textthreeoldstyle, \textthreequarters, \textthreesuperior, \texttildelow, \texttimes, \texttrademark, \texttwooldstyle, \texttwosuperior, \textunderscore, \textuparrow, \textvisiblespace, \textwon, \textyen, \textzerooldstyle

24.7.37 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 already loaded in all the components that include the TeX input jax, other than `input/tex-base`. To load the *textmacros* extension explicitly (when using `input/tex-base` for example), 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* extension is loaded.

Since the *textmacros* 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: {'[-]': ['textmacros']}}
};
```

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. Some extensions (e.g., the *bboldx* and *textcomp* packages) add more macros to this list when they are loaded.

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
\’	acute accent
\`	grave accent
\‘	grave accent
\^	circumflex accent
\"	umlaut accent
\~	tilde accent
\=	macron accent
\.	over dot accent
\u	breve accent
\v	caron accent
\underline	underlined text

Font Control

\emph	emphasized text
\rm	roman text
\mit	math italic text
\oldstyle	oldstyle numerals
\cal	calligraphic text
\it	italic text
\bf	bold text
\sf	sans-serif text
\tt	typewriter text
\frak	Fraktur text
\Bbb	blackboard-bold text
\textnormal	normal text
\textup	upright text
\textrm	roman text
\textit	italic text
\textbf	bold text
\textsf	sans-serif text
\texttt	typewriter text

Size Control

\Tiny	tiny size
\tiny	very tiny size
\scriptsize	size of super- and subscripts
\SMALL	same as script size
\footnotesize	size of footnotes
\Small	same as footnote size
\small	small size
\normalsize	standard size
\large	large size
\Large	larger sizse
\LARGE	very large size
\huge	even larger size
\Huge	larger size yet
\HUGE	largest size (non-standard)

Special Characters

\dagger	†
\ddagger	‡
\S	§
\AA	
\ldots	ellipses
\vdots	three vertical dots

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)
\mkern	kern (by a given size)
\mskip	horizontal space (of a given size)
\mspace	horizontal space (of a given size)
\rule	line of a given width and height
\Rule	box with given dimensions (non-standard)
\Space	space with given dimensions (non-standard)

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
\data	specify data attribute
\cssId	specify CSS id

Character Creation

\char	character from unicode value
\U	character from unicode value
\unicode	character from unicode value
\mmlToken	create MathML token element

Equation Numbers

\ref	cite a labeled equation
\eqref	cite a labeled equation with parentheses

Bounding Box Macros

\llap	overlap on the left
\rlap	overlap on the right
\phantom	empty space of the size of given math
\hphantom	0-height box with width of given math
\vphantom	0-width box with height of given math
\smash	math with zero height and depth

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 packages 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']},
```

(continues on next page)

(continued from previous page)

```
  tex: {
    packages: {'[+]': ['textmacros', 'bbox']},
    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 appropriate for textmode, but some can be usefully employed in text mode.

24.7.38 unicode

The *unicode* extension implements several macros for generating character from their unicode code points: the standard `\char` control sequence, plus the non-standard `\U{}` and `\unicode{}` macros.

Note

The `\U` and `\char` commands are new in version 4.

The `\char` command must be followed by a number giving the unicode code point for the desired character. As in actual TeX, that number can be in hexadecimal if preceded by a quotation mark ("), or octal if preceded by a single quote ('), or by a base-10 number, or by a backtick (`) followed by a character or single-character control sequence. For example,

```
\char65
\char"41
\char'101
\char`A
\char`\A
```

all produce the letter A.

The `\unicode` command takes an argument that is either a base-10 number or a hexadecimal number preceded by an `x`. 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, using optional bracketed arguments.

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.

Finally, the `\U` command takes an argument that is a hexadecimal number giving the unicode code point for the desired character. Unlike `\char` and `\unicode`, the resulting character is inserted back into the TeX input string and processed by the TeX interpreter, so `\U{5C}sum` would be equivalent to `\sum` since the backslash is U+005C.

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.

---

unicode Commands

The *unicode* extension implements the following macros: `\char`, `\U`, `\unicode`

24.7.39 units

The *units* extension implements the `units` style package from LaTeX, which defines the `\nicefrac`, `\unitfrac` and `\units` macros. See the units CTAN page for more information and documentation.

Note

An implementation for MathJax of the `siunitx` LaTeX package is under development by a third party, and we hope to make it available soon.

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

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

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

---

units Options

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

```
MathJax = {
  tex: {
    units: {
      loose: false,
      ugly: false
    }
  }
};
```

`loose: false`
: Determines whether fractions are loose (true) or tight (false), as described in the units documentation.

`ugly: false`
: Determines whether fractions are ugly (true) or nice (false), as described in the units documentation.

---

units Commands

The *units* extension implements the following macros: `\nicefrac`, `\unitfrac`, `\units`

24.7.40 upgreek

The *upgreek* extension implements the `upgreek` style package from LaTeX. It provides upright Greek characters for both lower and upper cases. See the upgreek CTAN page for more information and documentation.

Note, that the extension does not implement the font selection options from the LaTeX package.

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

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

You can configure the *autoload* extension to load *upgreek* via

```
window.MathJax = {
  tex: {
    autoload: {
      upgreek: ['upalpha', 'upbeta', 'upchi', 'updelta', 'Updelta', 'upepsilon',
                'upeta', 'upgamma', 'Upgamma', 'upiota', 'upkappa', 'uplambda',
```

(continues on next page)

(continued from previous page)

```
                'Uplambda', 'upmu', 'upnu', 'upomega', 'Upomega', 'upomicron',
                'upphi', 'Upphi', 'uppi', 'Uppi', 'uppsi', 'Uppsi', 'uprho',
                'upsigma', 'Upsigma', 'uptau', 'uptheta', 'Uptheta', 'upupsilon',
                'Upupsilon', 'upvarepsilon', 'upvarphi', 'upvarpi', 'upvarrho',
                'upvarsigma', 'upvartheta', 'upxi', 'Upxi', 'upzeta']
    }
  }
};
```

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

---

upgreek Commands

The *upgreek* extension implements the following macros: `\upalpha`, `\upbeta`, `\upchi`, `\updelta`, `\Updelta`, `\upepsilon`, `\upeta`, `\upgamma`, `\Upgamma`, `\upiota`, `\upkappa`, `\uplambda`, `\Uplambda`, `\upmu`, `\upnu`, `\upomega`, `\Upomega`, `\upomicron`, `\upphi`, `\Upphi`, `\uppi`, `\Uppi`, `\uppsi`, `\Uppsi`, `\uprho`, `\upsigma`, `\Upsigma`, `\uptau`, `\uptheta`, `\Uptheta`, `\upupsilon`, `\Upupsilon`, `\upvarepsilon`, `\upvarphi`, `\upvarpi`, `\upvarrho`, `\upvarsigma`, `\upvartheta`, `\upxi`, `\Upxi`, `\upzeta`

24.7.41 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.

---

verb Commands

The *verb* extension implements the following macros: `\verb`

These extensions have not been ported to the current version of MathJax:

24.7.42 autobold

The *autobold* extension is no longer available in MathJax version 3 or later.

It is possible to use a TeX input jax pre-filter to get the same effect, however. See *An Autobold Filter* for an example of how to do this.

24.7.43 autoload-all

The *autoload-all* extension from v2 has been replaced in v3 and above by the *autoload* extension, which is more easily configurable.

24.7.44 mediawiki-texvc

The *mediawiki-texvc* extension predefines macros that match the behavior of the MediaWiki Math Extension.

This extension has not been translated to version 3 or above, so currently it is not available. It *may* be included in a future release of MathJax.

See the section on *A Custom Extension* for how to create your own TeX extension.

24.8 Supported TeX/LaTeX commands

This is a long list of the TeX macros supported by MathJax.

Information about how to use LaTeX macros can be found on a variety of websites, including:

- A basic tutorial is available on the Mathematics StackExchange. This is for v2, but the information mostly still applies to v3 and above, though it won't include the features that are new in later versions.
- More complete details, with examples and explanations, are 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 and above.
- The LaTeX Wikibook sections in Mathematics and Advanced Mathematics also have good information about using LaTeX, but remember that MathJax mostly deals with the math-mode macros, not text-mode layout, and this wikibook is about LaTeX in general, not about MathJax specifically.

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 autoloaded by the *autoload* extension, otherwise the extension must be loaded explicitly in your configuration. If a macro from the base package is redefined by an extension, then **base** is included in the second column along with the package name that redefines it.

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 extension. These are marked here as being in the *text-base* package.

24.8.1 Symbols

_	**base**, text-base
.	
'	**base**, text-base
’	
(	**base**, physics
)	**base**, physics
[	**base**, physics
]	**base**, physics
{	**base**, text-base
}	**base**, text-base
@	*amscd*
/	
\ (backslash-space)	**base**, text-base
_	**base**, text-base
\,	**base**, text-base
\;	**base**, text-base
\:	**base**, mathtools, text-base
\!	**base**, text-base
\.	text-base
\'	text-base
\‘	text-base
\’	text-base
\"	text-base
\(	text-base
\{	**base**, text-base
\}	**base**, text-base
*	
\\	**base**, text-base
\&	**base**, text-base
\#	**base**, text-base
\%	**base**, text-base
\`	text-base
\^	text-base
\=	text-base
\>	**base**, text-base
|	**base**, *braket*
\~	text-base
\$	**base**, text-base
&	**base**, cases, text-base
#	**base**, text-base
%	**base**, text-base
`	text-base
^	**base**, text-base
<	**base**, texhtml
>	
|	**base**, *braket*, physics

continues on next page

Table 1 – continued from previous page

~	**base**, text-base
$	text-base

24.8.2 A

\AA	**base**, text-base
\above	
\abovewithdelims	
\Aboxed	mathtools
\abs	physics
\absolutevalue	physics
\acomm	physics
\acos	physics
\acosecant	physics
\acosine	physics
\acot	physics
\acotangent	physics
\acsc	physics
\acute	
\adjustlimits	mathtools
\admat	physics
\aleph	
\allowbreak	
\alpha	
\alwaysDashedLine	*bussproofs*
\alwaysNoLine	*bussproofs*
\alwaysRootAtBottom	*bussproofs*
\alwaysRootAtTop	*bussproofs*
\alwaysSingleLine	*bussproofs*
\alwaysSolidLine	*bussproofs*
\amalg	
\And	
\angle	
\anticommutator	physics
\antidiagonalmatrix	physics
\approx	
\approxcolon	mathtools
\Approxcolon	mathtools
\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

continues on next page

Table 2 – continued from previous page

\arctangent	physics
\arg	
\array	
\ArrowBetweenLines	mathtools
\arrowvert	
\Arrowvert	
\asec	physics
\asecant	physics
\asin	physics
\asine	physics
\ast	
\asymp	
\atan	physics
\atangent	physics
\atop	
\atopwithdelims	
\AXC	*bussproofs*
\Axiom	*bussproofs*
\AxiomC	*bussproofs*

24.8.3 B

\backepsilon	**ams**
\backprime	**ams**
\backsim	**ams**
\backsimeq	**ams**
\backslash	
\badbreak	
\bar	
\barwedge	**ams**
\bbalpha	bboldx
\Bbb	**base**, text-base
\bbbeta	bboldx
\Bbbk	**ams**
\bbchi	bboldx
\bbDelta	bboldx
\bbdelta	bboldx
\bbdotlessi	bboldx
\bbdotlessj	bboldx
\bbepsilon	bboldx
\bbeta	bboldx
\bbGamma	bboldx
\bbgamma	bboldx
\bbiota	bboldx
\bbkappa	bboldx
\bbLambda	bboldx
\bblambda	bboldx
\bbLangle	bboldx
\bbLbrack	bboldx
\bbLparen	bboldx

continues on next page

Table 3 – continued from previous page

\bbmu	bboldx
\bbnu	bboldx
\bbOmega	bboldx
\bbomega	bboldx
\bbox	*bbox*
\bbPhi	bboldx
\bbphi	bboldx
\bbPi	bboldx
\bbpi	bboldx
\bbPsi	bboldx
\bbpsi	bboldx
\bbRangle	bboldx
\bbRbrack	bboldx
\bbrho	bboldx
\bbRparen	bboldx
\bbSigma	bboldx
\bbsigma	bboldx
\bbtau	bboldx
\bbTheta	bboldx
\bbtheta	bboldx
\bbUpsilon	bboldx
\bbupsilon	bboldx
\bbXi	bboldx
\bbxi	bboldx
\bbzeta	bboldx
\bcancel	*cancel*
\because	**ams**
\begin	
\begingroup	begingroup
\begingroupReset	begingroup
\begingroupSandbox	begingroup
\beta	
\beth	**ams**
\between	**ams**
\bf	**base**, text-base
\bfbbalpha	bboldx
\bfbbbeta	bboldx
\bfbbchi	bboldx
\bfbbDelta	bboldx
\bfbbdelta	bboldx
\bfbbdotlessi	bboldx
\bfbbdotlessj	bboldx
\bfbbepsilon	bboldx
\bfbbeta	bboldx
\bfbbGamma	bboldx
\bfbbgamma	bboldx
\bfbbiota	bboldx
\bfbbkappa	bboldx
\bfbbLambda	bboldx
\bfbblambda	bboldx
\bfbbLangle	bboldx
\bfbbLbrack	bboldx

continues on next page

Table 3 – continued from previous page

\bfbbLparen	bboldx
\bfbbmu	bboldx
\bfbbnu	bboldx
\bfbbOmega	bboldx
\bfbbomega	bboldx
\bfbbPhi	bboldx
\bfbbphi	bboldx
\bfbbPi	bboldx
\bfbbpi	bboldx
\bfbbPsi	bboldx
\bfbbpsi	bboldx
\bfbbRangle	bboldx
\bfbbRbrack	bboldx
\bfbbrho	bboldx
\bfbbRparen	bboldx
\bfbbSigma	bboldx
\bfbbsigma	bboldx
\bfbbtau	bboldx
\bfbbTheta	bboldx
\bfbbtheta	bboldx
\bfbbUpsilon	bboldx
\bfbbupsilon	bboldx
\bfbbXi	bboldx
\bfbbxi	bboldx
\bfbbzeta	bboldx
\BIC	*bussproofs*
\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**
\bigtimes	mathtools
\bigtriangledown	

continues on next page

Table 3 – continued from previous page

\bigtriangleup	
\biguplus	
\bigvee	
\bigwedge	
\BinaryInf	*bussproofs*
\BinaryInfC	*bussproofs*
\binom	**ams**
\blacklozenge	**ams**
\blacksquare	**ams**
\blacktriangle	**ams**
\blacktriangledown	**ams**
\blacktriangleleft	**ams**
\blacktriangleright	**ams**
\bmod	
\bmqty	physics
\boldsymbol	*boldsymbol*
\bot	
\bowtie	
\Box	**ams**
\boxdot	**ams**
\boxed	**base**, **ams**
\boxminus	**ams**
\boxplus	**ams**
\boxtimes	**ams**
\bqty	physics
\Bqty	physics
\bra	*braket*, physics
\Bra	*braket*
\brace	
\bracevert	
\brack	
\braket	*braket*, physics
\Braket	*braket*
\break	
\breakAlign	
\breve	
\buildrel	
\bullet	
\bumpeq	**ams**
\Bumpeq	**ams**

24.8.4 C

\cal	**base**, text-base
\cancel	*cancel*
\cancelto	*cancel*
\cap	
\Cap	**ams**
\cases	
\cdot	

continues on next page

Table 4 – continued from previous page

\cdotp	
\cdots	
\ce	*mhchem*
\cellcolor	colortbl
\celsius	gensymb
\centerdot	**ams**
\centernot	centernot
\centerOver	centernot
\cfrac	**ams**
\char	*unicode*, text-base
\check	
\checkmark	**ams**
\chi	
\choose	
\circ	
\circeq	**ams**
\circlearrowleft	**ams**
\circlearrowright	**ams**
\circledast	**ams**
\circledcirc	**ams**
\circleddash	**ams**
\circledR	**ams**
\circledS	**ams**
\clap	mathtools
\class	*html*, text-base
\clubsuit	
\colon	
\colonapprox	mathtools
\Colonapprox	mathtools
\colondash	mathtools
\Colondash	mathtools
\coloneq	mathtools
\Coloneq	mathtools
\coloneqq	mathtools
\Coloneqq	mathtools
\colonsim	mathtools
\Colonsim	mathtools
\color	*color*, colorv2, text-base
\colorbox	*color*, text-base
\columncolor	colortbl
\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

continues on next page

Table 4 – continued from previous page

\cp	physics
\cr	
\cramped	mathtools
\crampedclap	mathtools
\crampedllap	mathtools
\crampedrlap	mathtools
\crampedsubstack	mathtools
\cross	physics
\crossproduct	physics
\csc	**base**, physics
\csch	physics
\cssId	*html*, text-base
\cup	
\Cup	**ams**
\curl	physics
\curlyeqprec	**ams**
\curlyeqsucc	**ams**
\curlyvee	**ams**
\curlywedge	**ams**
\curvearrowleft	**ams**
\curvearrowright	**ams**

24.8.5 D

\dagger	**base**, text-base
\daleth	**ams**
\dashcolon	mathtools
\Dashcolon	mathtools
\dashedLine	*bussproofs*
\dashleftarrow	**ams**
\dashrightarrow	**ams**
\dashv	
\data	*html*, text-base
\dbinom	**ams**
\dblcolon	mathtools
\dd	physics
\ddagger	**base**, text-base
\ddddot	**base**, **ams**
\dddot	**base**, **ams**
\ddot	
\ddots	
\DeclareMathOperator	**ams**
\DeclarePairedDelimiter	mathtools
\DeclarePairedDelimiters	mathtools
\DeclarePairedDelimitersX	mathtools
\DeclarePairedDelimitersXPP	mathtools
\DeclarePairedDelimiterX	mathtools
\DeclarePairedDelimiterXPP	mathtools
\def	**newcommand**
\definecolor	*color*

continues on next page

Table 5 – continued from previous page

\deg	
\degree	gensymb
\delta	
\Delta	
\derivative	physics
\det	**base**, physics
\determinant	physics
\dfrac	**ams**
\diagdown	**ams**
\diagonalmatrix	physics
\diagup	**ams**
\diamond	
\Diamond	**ams**
\diamondsuit	
\diffd	physics
\differential	physics
\digamma	**ams**
\dim	
\displaylines	
\displaystyle	
\div	**base**, physics
\divergence	physics
\divideontimes	**ams**
\divisionsymbol	physics
\divsymbol	physics
\dmat	physics
\dot	
\doteq	
\Doteq	**ams**
\doteqdot	**ams**
\dotplus	**ams**
\dotproduct	physics
\dots	
\dotsb	
\dotsc	
\dotsi	
\dotsm	
\dotso	
\doublebarwedge	**ams**
\doublecap	**ams**
\doublecup	**ams**
\downarrow	
\Downarrow	
\downdownarrows	**ams**
\downharpoonleft	**ams**
\downharpoonright	**ams**
\dv	physics
\dyad	physics

24.8.6 E

\ell	
\emph	text-base
\empheqbigl	empheq
\empheqbiglangle	empheq
\empheqbiglbrace	empheq
\empheqbiglbrack	empheq
\empheqbiglceil	empheq
\empheqbiglfloor	empheq
\empheqbiglparen	empheq
\empheqbiglvert	empheq
\empheqbiglVert	empheq
\empheqbigr	empheq
\empheqbigrangle	empheq
\empheqbigrbrace	empheq
\empheqbigrbrack	empheq
\empheqbigrceil	empheq
\empheqbigrfloor	empheq
\empheqbigrparen	empheq
\empheqbigrvert	empheq
\empheqbigrVert	empheq
\empheql	empheq
\empheqlangle	empheq
\empheqlbrace	empheq
\empheqlbrack	empheq
\empheqlceil	empheq
\empheqlfloor	empheq
\empheqlparen	empheq
\empheqlvert	empheq
\empheqlVert	empheq
\empheqr	empheq
\empheqrangle	empheq
\empheqrbrace	empheq
\empheqrbrack	empheq
\empheqrceil	empheq
\empheqrfloor	empheq
\empheqrparen	empheq
\empheqrvert	empheq
\empheqrVert	empheq
\emptyset	
\enclose	*enclose*
\end	
\endgroup	begingroup
\enspace	**base**, text-base
\epsilon	
\eqalign	
\eqalignno	
\eqcirc	**ams**
\eqcolon	mathtools
\Eqcolon	mathtools
\eqqcolon	mathtools

continues on next page

Table 6 – continued from previous page

\Eqqcolon	mathtools
\eqref	**ams**, text-base
\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

24.8.7 F

\fallingdotseq	**ams**
\fbox	
\fCenter	*bussproofs*
\fcolorbox	*color*, text-base
\fderivative	physics
\fdv	physics
\Finv	**ams**
\flat	
\flatfrac	physics
\footnotesize	**base**, text-base
\forall	
\frac	**base**, **ams**
\frak	**base**, text-base
\framebox	
\frown	
\functionalderivative	physics

24.8.8 G

\Game	**ams**
\gamma	
\Gamma	
\gcd	
\gdef	begingroup
\ge	
\genfrac	**ams**
\geq	
\geqq	**ams**
\geqslant	**ams**

continues on next page

Table 7 – continued from previous page

\gets	
\gg	
\ggg	**ams**
\gggtr	**ams**
\gimel	**ams**
\global	begingroup
\gnapprox	**ams**
\gneq	**ams**
\gneqq	**ams**
\gnsim	**ams**
\goodbreak	
\grad	physics
\gradient	physics
\gradientnabla	physics
\grave	
\gt	
\gtrapprox	**ams**
\gtrdot	**ams**
\gtreqless	**ams**
\gtreqqless	**ams**
\gtrless	**ams**
\gtrsim	**ams**
\gvertneqq	**ams**

24.8.9 H

\hat	
\hbar	
\hbox	
\hdashline	
\heartsuit	
\hfil	
\hfill	
\hfilll	
\hline	
\hom	
\hookleftarrow	
\hookrightarrow	
\hphantom	**base**, text-base
\href	*html*, text-base
\hsize	
\hskip	**base**, text-base
\hslash	**ams**
\hspace	**base**, text-base
\huge	**base**, text-base, fontsizev3
\Huge	**base**, text-base, fontsizev3
\HUGE	**base**, text-base
\hypcosecant	physics
\hypcosine	physics
\hypcotangent	physics
\hypsecant	physics
\hypsine	physics
\hyptangent	physics

24.8.10 I

\iddots	
\identitymatrix	physics
\idotsint	**ams**
\iff	
\iiiint	**ams**
\iiint	
\iint	
\Im	**base**, physics
\imaginary	physics
\imat	physics
\imath	
\imathbb	bboldx
\imathbfbb	bboldx
\impliedby	**ams**
\implies	**ams**
\in	
\inf	
\infty	
\injlim	**ams**
\innerproduct	physics
\int	
\intercal	**ams**
\intop	
\iota	
\ip	physics
\it	**base**, text-base
\itextbb	text-bboldx
\itextbfbb	text-bboldx

24.8.11 J

\jmath	
\jmathbb	bboldx
\jmathbfbb	bboldx
\Join	**ams**
\jtextbb	text-bboldx
\jtextbfbb	text-bboldx

24.8.12 K

\kappa	
\ker	
\kern	**base**, text-base
\ket	*braket*, physics
\Ket	*braket*
\ketbra	*braket*, physics
\Ketbra	*braket*

24.8.13 L

\label	
\lambda	
\Lambda	
\land	
\langle	
\laplacian	physics
\large	**base**, text-base
\Large	**base**, text-base
\LARGE	**base**, text-base
\LaTeX	
\lbrace	
\lbrack	
\lceil	
\ldotp	
\ldots	**base**, text-base
\le	
\leadsto	**ams**
\left	
\Leftarrow	
\leftarrow	
\leftarrowtail	**ams**
\leftharpoondown	
\leftharpoonup	
\LeftLabel	*bussproofs*
\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	

continues on next page

Table 8 – continued from previous page

\limsup	
\ll	
\LL	*bussproofs*
\llap	**base**, text-base
\llcorner	**ams**
\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	
\longmapsto	
\longrightarrow	
\Longrightarrow	
\looparrowleft	**ams**
\looparrowright	**ams**
\lor	
\lower	
\lozenge	**ams**
\lparen	mathtools
\lrcorner	**ams**
\Lsh	**ams**
\lt	
\ltimes	**ams**
\lvert	**ams**
\lVert	**ams**
\lvertneqq	**ams**

24.8.14 M

\MakeAboxedCommand	mathtools
\makebox	
\maltese	**ams**
\mapsto	
\mathbb	**base**, bboldx
\mathbbm	bbm
\mathbbmss	bbm
\mathbbmtt	bbm
\mathbf	
\mathbfbb	bboldx

continues on next page

Table 9 – continued from previous page

\mathbfcal	
\mathbffrak	
\mathbfit	
\mathbfscr	
\mathbfsf	
\mathbfsfit	
\mathbfsfup	
\mathbfup	
\mathbin	
\mathcal	
\mathchoice	
\mathclap	mathtools
\mathclose	
\mathds	dsfont
\mathfrak	
\mathinner	
\mathit	
\mathllap	mathtools
\mathmakebox	mathtools
\mathmbox	mathtools
\mathnormal	
\mathop	
\mathopen	
\mathord	
\mathpunct	
\mathrel	
\mathring	**ams**
\mathrlap	mathtools
\mathrm	
\mathscr	
\mathsf	
\mathsfit	
\mathsfup	
\mathstrut	
\mathtip	*action*
\mathtoolsset	mathtools
\mathtt	
\mathup	
\mathversion	bbm
\matrix	
\matrixdeterminant	physics
\matrixel	physics
\matrixelement	physics
\matrixquantity	physics
\max	
\mbox	
\mdet	physics
\measuredangle	**ams**
\mel	physics
\mhchemBondDTD	*mhchem*
\mhchemBondTD	*mhchem*
\mhchemBondTDD	*mhchem*

continues on next page

Table 9 – continued from previous page

\mhchemleftarrow	*mhchem*
\mhchemleftrightarrow	*mhchem*
\mhchemlongleftarrow	*mhchem*
\mhchemlongleftrightarrow	*mhchem*
\mhchemlongleftrightarrows	*mhchem*
\mhchemlongLeftrightharpoons	*mhchem*
\mhchemlongrightarrow	*mhchem*
\mhchemlongrightleftharpoons	*mhchem*
\mhchemlongRightleftharpoons	*mhchem*
\mhchemrightarrow	*mhchem*
\mhchemxleftarrow	*mhchem*
\mhchemxleftrightarrow	*mhchem*
\mhchemxleftrightarrows	*mhchem*
\mhchemxLeftrightharpoons	*mhchem*
\mhchemxrightarrow	*mhchem*
\mhchemxrightleftharpoons	*mhchem*
\mhchemxRightleftharpoons	*mhchem*
\mho	**ams**
\micro	gensymb
\mid	
\middle	
\min	
\minCDarrowheight	*amscd*
\minCDarrowwidth	*amscd*
\mit	**base**, text-base
\mkern	**base**, text-base
\mmlToken	**base**, text-base
\mod	
\models	
\MoveEqLeft	mathtools
\moveleft	
\moveright	
\mp	
\mqty	physics
\mskip	**base**, text-base
\mspace	**base**, text-base
\MTFlushSpaceAbove	mathtools
\MTFlushSpaceBelow	mathtools
\MTThinColon	mathtools
\mu	
\multimap	**ams**

24.8.15 N

\nabla	
\natural	
\naturallogarithm	physics
\ncong	**ams**
\ndownarrow	mathtools
\ne	

continues on next page

Table 10 – continued from previous page

\nearrow	
\neg	
\negmedspace	**ams**
\negthickspace	**ams**
\negthinspace	**base**, text-base
\neq	
\newcolumntype	
\newcommand	**newcommand**
\newenvironment	**newcommand**
\Newextarrow	*extpfeil*
\newline	
\newtagform	mathtools
\nexists	**ams**
\ngeq	**ams**
\ngeqq	**ams**
\ngeqslant	**ams**
\ngtr	**ams**
\ni	
\nicefrac	units
\nleftarrow	**ams**
\nLeftarrow	**ams**
\nleftrightarrow	**ams**
\nLeftrightarrow	**ams**
\nleq	**ams**
\nleqq	**ams**
\nleqslant	**ams**
\nless	**ams**
\nmid	**ams**
\nobreak	
\nobreakspace	**ams**
\nolimits	
\noLine	*bussproofs*
\nonscript	
\nonumber	
\norm	physics
\normalsize	**base**, text-base, fontsizev3
\not	
\notag	**ams**
\notChar	
\notin	
\nparallel	**ams**
\nprec	**ams**
\npreceq	**ams**
\nrightarrow	**ams**
\nRightarrow	**ams**
\nshortmid	**ams**
\nshortparallel	**ams**
\nsim	**ams**
\nsubseteq	**ams**
\nsubseteqq	**ams**
\nsucc	**ams**
\nsucceq	**ams**

continues on next page

Table 10 – continued from previous page

\nsupseteq	**ams**
\nsupseteqq	**ams**
\ntriangleleft	**ams**
\ntrianglelefteq	**ams**
\ntriangleright	**ams**
\ntrianglerighteq	**ams**
\nu	
\nuparrow	mathtools
\nvdash	**ams**
\nvDash	**ams**
\nVdash	**ams**
\nVDash	**ams**
\nwarrow	

24.8.16 O

\odot	
\ohm	gensymb
\oiiint	
\oiint	
\oint	
\ointop	
\oldstyle	**base**, text-base
\omega	
\Omega	
\omicron	
\ominus	
\op	physics
\operatorname	**ams**
\oplus	
\order	physics
\ordinarycolon	mathtools
\oslash	
\otimes	
\outerproduct	physics
\over	
\overbrace	
\overbracket	mathtools
\overleftarrow	
\overleftrightarrow	
\overline	
\overparen	
\overrightarrow	
\overset	
\overunderset	
\overwithdelims	
\owns	

24.8.17 P

\parallel	
\parbox	
\partial	
\partialderivative	physics
\paulimatrix	physics
\pb	physics
\pderivative	physics
\pdv	physics
\perp	
\perthousand	gensymb
\phantom	**base**, text-base
\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**
\prescript	mathtools
\prime	
\principalvalue	physics
\Probability	physics
\prod	
\projlim	**ams**
\propto	
\psi	
\Psi	
\pu	*mhchem*
\pv	physics
\PV	physics

24.8.18 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	**base**, text-base
\qsince	physics
\qthen	physics
\qty	physics
\quad	**base**, text-base
\quantity	physics
\QuaternaryInf	*bussproofs*
\QuaternaryInfC	*bussproofs*
\QuinaryInf	*bussproofs*
\QuinaryInfC	*bussproofs*
\qunless	physics
\qusing	physics

24.8.19 R

\raise	
\rangle	
\rank	physics
\rbrace	
\rbrack	
\rceil	
\Re	**base**, physics
\real	physics
\ref	**base**, text-base
\refeq	mathtools
\renewcommand	**newcommand**
\renewenvironment	**newcommand**
\renewtagform	mathtools
\require	**require**

continues on next page

Table 14 – continued from previous page

\Res	physics
\Residue	physics
\restriction	**ams**
\rfloor	
\rgroup	
\rhd	**ams**
\rho	
\right	
\Rightarrow	
\rightarrow	
\rightarrowtail	**ams**
\rightharpoondown	
\rightharpoonup	
\RightLabel	*bussproofs*
\rightleftarrows	**ams**
\rightleftharpoons	**base**, **ams**
\rightrightarrows	**ams**
\rightsquigarrow	**ams**
\rightthreetimes	**ams**
\risingdotseq	**ams**
\RL	*bussproofs*
\rlap	**base**, text-base
\rm	**base**, text-base
\rmoustache	
\root	
\rootAtBottom	*bussproofs*
\rootAtTop	*bussproofs*
\rowcolor	colortbl
\rparen	mathtools
\Rrightarrow	**ams**
\Rsh	**ams**
\rtimes	**ams**
\rule	**base**, text-base
\Rule	**base**, text-base
\rvert	**ams**
\rVert	**ams**

24.8.20 S

\S	**base**, text-base
\sbmqty	physics
\scriptscriptstyle	
\scriptsize	**base**, text-base, fontsizev3
\scriptstyle	
\searrow	
\sec	**base**, physics
\secant	physics
\sech	physics
\set	*braket*
\Set	*braket*

continues on next page

Table 15 – continued from previous page

\setminus	
\setOptions	setoptions
\sf	**base**, text-base
\sharp	
\shortmid	**ams**
\shortparallel	**ams**
\shortvdotswithin	mathtools
\shoveleft	**ams**, mathtools
\shoveright	**ams**, mathtools
\sideset	**ams**
\sigma	
\Sigma	
\sim	
\simcolon	mathtools
\Simcolon	mathtools
\simeq	
\sin	**base**, physics
\sine	physics
\singleLine	*bussproofs*
\sinh	**base**, physics
\skew	
\small	**base**, text-base, fontsizev3
\Small	**base**, text-base
\SMALL	**base**, text-base
\smallfrown	**ams**
\smallint	
\smallmatrixquantity	physics
\smallsetminus	**ams**
\smallsmile	**ams**
\smash	**base**, text-base
\smdet	physics
\smile	
\smqty	physics
\solidLine	*bussproofs*
\Space	**base**, text-base
\space	
\spadesuit	
\sphericalangle	**ams**
\splitdfrac	mathtools
\splitfrac	mathtools
\spmqty	physics
\sPmqty	physics
\sqcap	
\sqcup	
\sqrt	
\sqsubset	**ams**
\sqsubseteq	
\sqsupset	**ams**
\sqsupseteq	
\square	**ams**
\stackbin	
\stackrel	

continues on next page

Table 15 – continued from previous page

\star	
\strut	
\style	*html*, text-base
\subset	
\Subset	**ams**
\subseteq	
\subseteqq	**ams**
\subsetneq	**ams**
\subsetneqq	**ams**
\substack	**ams**
\succ	
\succapprox	**ams**
\succcurlyeq	**ams**
\succeq	
\succnapprox	**ams**
\succneqq	**ams**
\succnsim	**ams**
\succsim	**ams**
\sum	
\sup	
\supset	
\Supset	**ams**
\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	

24.8.21 T

\tag	**ams**
\tan	**base**, physics
\tangent	physics
\tanh	**base**, physics
\tau	
\tbinom	**ams**
\TeX	
\text	
\textacutedbl	textcomp
\textasciiacute	textcomp
\textasciibreve	textcomp
\textasciicaron	textcomp
\textasciicircum	textcomp
\textasciidieresis	textcomp
\textasciimacron	textcomp
\textasciitilde	textcomp
\textasteriskcentered	textcomp
\textbackslash	textcomp
\textbaht	textcomp
\textbar	textcomp
\textbardbl	textcomp
\textbb	text-bboldx
\textbf	**base**, text-base
\textbfbb	text-bboldx
\textbigcircle	textcomp
\textblank	textcomp
\textborn	textcomp
\textbraceleft	textcomp
\textbraceright	textcomp
\textbrokenbar	textcomp
\textbullet	textcomp
\textcelsius	textcomp
\textcent	textcomp
\textcentoldstyle	textcomp
\textcircledP	textcomp
\textclap	mathtools
\textcolonmonetary	textcomp
\textcolor	*color*, text-base
\textcompwordmark	textcomp
\textcopyleft	textcomp
\textcopyright	textcomp
\textcurrency	textcomp
\textdagger	textcomp
\textdaggerdbl	textcomp
\textdegree	textcomp
\textdied	textcomp
\textdiscount	textcomp
\textdiv	textcomp
\textdivorced	textcomp
\textdollar	textcomp

continues on next page

Table 16 – continued from previous page

\textdollaroldstyle	textcomp
\textdong	textcomp
\textdownarrow	textcomp
\texteightoldstyle	textcomp
\textellipsis	textcomp
\textemdash	textcomp
\textendash	textcomp
\textestimated	textcomp
\texteuro	textcomp
\textexclamdown	textcomp
\textfiveoldstyle	textcomp
\textflorin	textcomp
\textfouroldstyle	textcomp
\textfractionsolidus	textcomp
\textgravedbl	textcomp
\textgreater	textcomp
\textguarani	textcomp
\textinterrobang	textcomp
\textinterrobangdown	textcomp
\textit	**base**, text-base
\textlangle	textcomp
\textlbrackdbl	textcomp
\textleftarrow	textcomp
\textless	textcomp
\textlira	textcomp
\textllap	mathtools
\textlnot	textcomp
\textlquill	textcomp
\textmarried	textcomp
\textmho	textcomp
\textminus	textcomp
\textmu	textcomp
\textmusicalnote	textcomp
\textnaira	textcomp
\textnineoldstyle	textcomp
\textnormal	**base**, text-base
\textnumero	textcomp
\textohm	textcomp
\textonehalf	textcomp
\textoneoldstyle	textcomp
\textonequarter	textcomp
\textonesuperior	textcomp
\textopenbullet	textcomp
\textordfeminine	textcomp
\textordmasculine	textcomp
\textparagraph	textcomp
\textperiodcentered	textcomp
\textpertenthousand	textcomp
\textperthousand	textcomp
\textpeso	textcomp
\textpm	textcomp
\textquestiondown	textcomp

continues on next page

Table 16 – continued from previous page

\textquotedblleft	textcomp
\textquotedblright	textcomp
\textquoteleft	textcomp
\textquoteright	textcomp
\textrangle	textcomp
\textrbrackdbl	textcomp
\textrecipe	textcomp
\textreferencemark	textcomp
\textregistered	textcomp
\textrightarrow	textcomp
\textrlap	mathtools
\textrm	**base**, text-base
\textrquill	textcomp
\textsection	textcomp
\textservicemark	textcomp
\textsevenoldstyle	textcomp
\textsf	**base**, text-base
\textsixoldstyle	textcomp
\textsterling	textcomp
\textstyle	
\textsurd	textcomp
\textthreeoldstyle	textcomp
\textthreequarters	textcomp
\textthreesuperior	textcomp
\texttildelow	textcomp
\texttimes	textcomp
\texttip	*action*
\texttrademark	textcomp
\texttt	**base**, text-base
\texttwooldstyle	textcomp
\texttwosuperior	textcomp
\textunderscore	textcomp
\textup	**base**, text-base
\textuparrow	textcomp
\textvisiblespace	textcomp
\textwon	textcomp
\textyen	textcomp
\textzerooldstyle	textcomp
\tfrac	**ams**
\therefore	**ams**
\theta	
\Theta	
\thickapprox	**ams**
\thicksim	**ams**
\thinspace	**base**, text-base
\TIC	*bussproofs*
\tilde	
\times	
\tiny	**base**, text-base, fontsizev3
\Tiny	**base**, text-base, fontsizev3
\to	
\toggle	*action*

continues on next page

Table 16 – continued from previous page

\top	
\tr	physics
\Tr	physics
\trace	physics
\Trace	physics
\triangle	
\triangledown	**ams**
\triangleleft	
\trianglelefteq	**ams**
\triangleq	**ams**
\triangleright	
\trianglerighteq	**ams**
\TrinaryInf	*bussproofs*
\TrinaryInfC	*bussproofs*
\tripledash	*mhchem*
\tt	**base**, text-base
\twoheadleftarrow	**ams**
\twoheadrightarrow	**ams**
\txtbbalpha	text-bboldx
\txtbbbeta	text-bboldx
\txtbbchi	text-bboldx
\txtbbDelta	text-bboldx
\txtbbdelta	text-bboldx
\txtbbdotlessi	text-bboldx
\txtbbdotlessj	text-bboldx
\txtbbepsilon	text-bboldx
\txtbbeta	text-bboldx
\txtbbGamma	text-bboldx
\txtbbgamma	text-bboldx
\txtbbiota	text-bboldx
\txtbbkappa	text-bboldx
\txtbbLambda	text-bboldx
\txtbblambda	text-bboldx
\txtbbLangle	text-bboldx
\txtbbLbrack	text-bboldx
\txtbbLparen	text-bboldx
\txtbbmu	text-bboldx
\txtbbnu	text-bboldx
\txtbbOmega	text-bboldx
\txtbbomega	text-bboldx
\txtbbPhi	text-bboldx
\txtbbphi	text-bboldx
\txtbbPi	text-bboldx
\txtbbpi	text-bboldx
\txtbbPsi	text-bboldx
\txtbbpsi	text-bboldx
\txtbbRangle	text-bboldx
\txtbbRbrack	text-bboldx
\txtbbrho	text-bboldx
\txtbbRparen	text-bboldx
\txtbbSigma	text-bboldx
\txtbbsigma	text-bboldx

continues on next page

Table 16 – continued from previous page

\txtbbtau	text-bboldx
\txtbbTheta	text-bboldx
\txtbbtheta	text-bboldx
\txtbbUpsilon	text-bboldx
\txtbbupsilon	text-bboldx
\txtbbXi	text-bboldx
\txtbbxi	text-bboldx
\txtbbzeta	text-bboldx
\txtbfbbalpha	text-bboldx
\txtbfbbbeta	text-bboldx
\txtbfbbchi	text-bboldx
\txtbfbbDelta	text-bboldx
\txtbfbbdelta	text-bboldx
\txtbfbbdotlessi	text-bboldx
\txtbfbbdotlessj	text-bboldx
\txtbfbbepsilon	text-bboldx
\txtbfbbeta	text-bboldx
\txtbfbbGamma	text-bboldx
\txtbfbbgamma	text-bboldx
\txtbfbbiota	text-bboldx
\txtbfbbkappa	text-bboldx
\txtbfbbLambda	text-bboldx
\txtbfbblambda	text-bboldx
\txtbfbbLangle	text-bboldx
\txtbfbbLbrack	text-bboldx
\txtbfbbLparen	text-bboldx
\txtbfbbmu	text-bboldx
\txtbfbbnu	text-bboldx
\txtbfbbOmega	text-bboldx
\txtbfbbomega	text-bboldx
\txtbfbbPhi	text-bboldx
\txtbfbbphi	text-bboldx
\txtbfbbPi	text-bboldx
\txtbfbbpi	text-bboldx
\txtbfbbPsi	text-bboldx
\txtbfbbpsi	text-bboldx
\txtbfbbRangle	text-bboldx
\txtbfbbRbrack	text-bboldx
\txtbfbbrho	text-bboldx
\txtbfbbRparen	text-bboldx
\txtbfbbSigma	text-bboldx
\txtbfbbsigma	text-bboldx
\txtbfbbtau	text-bboldx
\txtbfbbTheta	text-bboldx
\txtbfbbtheta	text-bboldx
\txtbfbbUpsilon	text-bboldx
\txtbfbbupsilon	text-bboldx
\txtbfbbXi	text-bboldx
\txtbfbbxi	text-bboldx
\txtbfbbzeta	text-bboldx

24.8.22 U

\U	*unicode*, text-base
\u	text-base
\UIC	*bussproofs*
\ulcorner	**ams**
\UnaryInf	*bussproofs*
\UnaryInfC	*bussproofs*
\underbrace	
\underbracket	mathtools
\underleftarrow	
\underleftrightarrow	
\underline	**base**, text-base
\underparen	
\underrightarrow	
\underset	
\unicode	*unicode*, text-base
\unitfrac	units
\units	units
\unlhd	**ams**
\unrhd	**ams**
\upalpha	upgreek
\uparrow	
\Uparrow	
\upbeta	upgreek
\upchi	upgreek
\updelta	upgreek
\Updelta	upgreek
\updownarrow	
\Updownarrow	
\upepsilon	upgreek
\upeta	upgreek
\upgamma	upgreek
\Upgamma	upgreek
\upharpoonleft	**ams**
\upharpoonright	**ams**
\upiota	upgreek
\upkappa	upgreek
\uplambda	upgreek
\Uplambda	upgreek
\uplus	
\upmu	upgreek
\upnu	upgreek
\upomega	upgreek
\Upomega	upgreek
\upomicron	upgreek
\upphi	upgreek
\Upphi	upgreek
\uppi	upgreek
\Uppi	upgreek
\uppsi	upgreek
\Uppsi	upgreek

continues on next page

Table 17 – continued from previous page

\uprho	upgreek
\uproot	
\upsigma	upgreek
\Upsigma	upgreek
\upsilon	
\Upsilon	
\uptau	upgreek
\uptheta	upgreek
\Uptheta	upgreek
\upuparrows	**ams**
\upupsilon	upgreek
\Upupsilon	upgreek
\upvarepsilon	upgreek
\upvarphi	upgreek
\upvarpi	upgreek
\upvarrho	upgreek
\upvarsigma	upgreek
\upvartheta	upgreek
\upxi	upgreek
\Upxi	upgreek
\upzeta	upgreek
\urcorner	**ams**
\usetagform	mathtools

24.8.23 V

\v	text-base
\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	
\varPhi	**ams**
\varpi	
\varPi	**ams**
\varprojlim	**ams**
\varpropto	**ams**
\varPsi	**ams**
\varrho	
\varsigma	
\varSigma	**ams**

continues on next page

Table 18 – continued from previous page

\varsubsetneq	**ams**
\varsubsetneqq	**ams**
\varsupsetneq	**ams**
\varsupsetneqq	**ams**
\vartheta	
\varTheta	**ams**
\vartriangle	**ams**
\vartriangleleft	**ams**
\vartriangleright	**ams**
\varUpsilon	**ams**
\varXi	**ams**
\vb	physics
\vbox	
\vcenter	
\vcentercolon	mathtools
\vdash	
\vDash	**ams**
\Vdash	**ams**
\vdot	physics
\vdots	**base**, text-base
\vdotswithin	mathtools
\vec	
\vectorarrow	physics
\vectorbold	physics
\vectorunit	physics
\vee	
\veebar	**ams**
\verb	*verb*
\Vert	
\vert	
\vmqty	physics
\vnabla	physics
\vphantom	**base**, text-base
\vqty	physics
\vtop	
\vu	physics
\Vvdash	**ams**

24.8.24 W

\wedge
\widehat
\widetilde
\wp
\wr

24.8.25 X

\xcancel	*cancel*
\xhookleftarrow	mathtools
\xhookrightarrow	mathtools
\xi	
\Xi	
\xleftarrow	**ams**
\xLeftarrow	mathtools
\xleftharpoondown	mathtools
\xleftharpoonup	mathtools
\xleftrightarrow	mathtools
\xLeftrightarrow	mathtools
\xleftrightharpoons	mathtools
\xlongequal	*extpfeil*
\xlongleftarrow	mathtools
\xLongleftarrow	mathtools
\xlongrightarrow	mathtools
\xLongrightarrow	mathtools
\xmapsto	*extpfeil*, mathtools
\xmat	physics
\xmathstrut	mathtools
\xmatrix	physics
\xrightarrow	**ams**
\xRightarrow	mathtools
\xrightharpoondown	mathtools
\xrightharpoonup	mathtools
\xrightleftharpoons	mathtools
\xtofrom	*extpfeil*
\xtwoheadleftarrow	*extpfeil*
\xtwoheadrightarrow	*extpfeil*

24.8.26 Y

\yen	**ams**

24.8.27 Z

\zeromatrix	physics
\zeta	
\zmat	physics

24.8.28 Environments

align	**ams**
align*	**ams**
alignat	**ams**
alignat*	**ams**

continues on next page

Table 19 – continued from previous page

aligned	**ams**
alignedat	**ams**
array	
bmatrix	**ams**
Bmatrix	**ams**
bmatrix*	mathtools
Bmatrix*	mathtools
bsmallmatrix	mathtools
Bsmallmatrix	mathtools
bsmallmatrix*	mathtools
Bsmallmatrix*	mathtools
cases	**ams**
cases*	mathtools
CD	*amscd*
crampedsubarray	mathtools
darray	
dcases	mathtools
dcases*	mathtools
displaymath	
drcases	mathtools
drcases*	mathtools
empheq	empheq
eqnarray	
eqnarray*	**ams**
equation	
equation*	**ams**
flalign	**ams**
flalign*	**ams**
gather	**ams**
gather*	**ams**
gathered	**ams**
indentalign	
lgathered	mathtools
math	
matrix	**ams**
matrix*	mathtools
multline	**ams**
multline*	**ams**
multlined	mathtools
numcases	cases
pmatrix	**ams**
pmatrix*	mathtools
prooftree	*bussproofs*
psmallmatrix	mathtools
psmallmatrix*	mathtools
rcases	mathtools
rcases*	mathtools
rgathered	mathtools
smallmatrix	**ams**, physics
smallmatrix*	mathtools
split	**ams**
spreadlines	mathtools

continues on next page

Table 19 – continued from previous page

subarray	**ams**
subnumcases	cases
vmatrix	**ams**
Vmatrix	**ams**
vmatrix*	mathtools
Vmatrix*	mathtools
vsmallmatrix	mathtools
Vsmallmatrix	mathtools
vsmallmatrix*	mathtools
Vsmallmatrix*	mathtools
xalignat	**ams**
xalignat*	**ams**
xxalignat	**ams**

CHAPTER

TWENTYFIVE

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 MathJax v2, these two actions were performed by distinct components (the `mml2jax` preprocessor, and the MathML input jax); in v3 and above, the `mml2jax` functionality has been folded into the MathML input jax.

MathJax's internal format is essentially MathML (with a few additions), implemented as javascript objects rather than DOM elements, and MathJax's various input processors all convert their original format into this internal MathML format; 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 modern browsers have native support for rendering MathML, most implement the MathML-Core standard, which is a limited subset of MathML. Unfortunately, it does not include everything that MathJax needs in order to produce the required output for its various input formats. For example, MathML-Core does not include the <mlabeledtr> element needed to produce equation numbers near the margins, or the table attributes that full-fledged MathML uses for making aligned equations. Furthermore, the quality of the output varies from browser to browser, and some systems require you to download and install extra fonts to support the mathematical notation.

So even though MathML could be used in recent versions of most browsers, MathJax makes it possible to view MathML notation in almost *all* browsers, in a consistent and convenient way.

In addition, MathJax provides support for assistive technology, such as screen readers and braille output devices, even when those tools don't understand MathML directly. MathJax can produce speech strings from math in several formats, and in several languages, including Braille.

25.1 MathML in HTML pages

For MathML that is embedded in HTML pages, you should not use named MathML entities, like `&Integral;` 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 may 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* specification 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.

If you are working in HTML rather than XHTML, you should not use a namespace prefix like `m:` or `mml:` for your MathML elements. That is, you should use <math> rather than <m:math> or <mml:math>. This is because HTML5 has deprecated namespaces, so they are no longer necessary, and it makes it harder for MathJax to identify the math when there are namespaces. If you properly declare the namespace in the <html> tag, MathJax will be able to find the namespaced math tags, but if you don't then MathJax may miss them.

25.2 HTML in MathML token nodes

The HTML5 specification allows for mixing HTML nodes inside MathML token nodes, and it is a long-standing request for MathJax to implement that as well. Version 4 includes this ability, you can now use HTML nodes as children of token nodes, such as <mtext>. Thus

```
<mtext>a button <input type="button" value="Push Me"> to press</mtext>
```

is allowed, and would produce an <mtext> element containing a button surrounded by some plain text.

Because MathJax does not try to sanitize the HTML in any way, allowing HTML in token elements would be a security issue for sites that allow user-supplied MathML. For this reason, the MathML input jax has a new option `allowHtmlInTokenNodes` to control whether to allow it, and it is `false` by default; you have to opt into this new feature if you want to use it on your site.

See the *Specifying the size of HTML in Expressions* section for information about how MathJax determines the size of HTML that is embedded in MathML token nodes.

25.3 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.

25.4 Content MathML

The version 2 `content-mathml` extension is not yet available in version 3 and above.

25.5 Experimental mml3 extension

MathML includes a number of tags that support elementary-school mathematics, like <mstack> and <mlongdiv>. MathJax has only experimental support for these tags via the *mml3* extension. This uses an XSLT transform to convert these tags into other presentation MathML tags that MathJax has implemented. This does a reasonable job for some constructs, and a poorer job for others, but it does make it possible to process elementary math within MathJax. Better support is planned for the future.

To activate experimental features in your documents, simply include `[mml]/mml3` in the `load` array of the `loader` section of your configuration:

```
MathJax = {
  loader: {load: ['[mml]/mml3']}
};
```

This will install a pre-filter on the MathML input jax that performs the XSLT transform before processing it.

25.6 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` submenu. See the MathML Annotation Framework documentation from the W3C, and the *Contextual Menu Options* section of this document for details.

CHAPTER

TWENTYSIX

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 and above, 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.

26.1 Loading the AsciiMath Component

The AsciiMath input jax has not yet been fully ported to version 3 or 4. 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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4/startup.js">
</script>
```

would load the AsciiMath input jax, the CommonHTML output jax, and the contextual menu component.

It is also possible to load the AsciiMath input processor in addition to other input formats. For example,

```
<script>
MathJax = {
  loader: {load: ['input/asciimath']},
};
</script>
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/tex-chtml.js">
</script>
```

would load the AsciiMath input along side TeX input (and produce CHTML output), so you could enter mathematics in either format.

26.2 AsciiMath delimiters

By default, the AsciiMath processor defines the back-tick (`) as the delimiter 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.

26.3 AsciiMath Inline and Display Modes

Normally, AsciiMath produces output that is in-line math but typeset with display-style rules. That means you need to use HTML tags to make a displayed equation be centered on a separate line. That is how AsciiMath is designed. It is possible, however, to configure MathJax's version of AsciiMath to allow for both in-line and displayed equations, just like in TeX.

Here is a configuration that does that:

```
<script>
MathJax = {
  loader: {load: ['input/asciimath', 'output/chtml']},
  output: {
    font: 'mathjax-modern'
  },
  asciimath: {
    delimiters: [['```','```'], ['``','``']]
  },
```

(continues on next page)

(continued from previous page)

```
  startup: {
    ready() {
      const {AsciiMath} = MathJax._.input.asciimath_ts;
      Object.assign(AsciiMath.prototype, {
        _compile: AsciiMath.prototype.compile,
        compile(math, document) {
          math.display = (math.start?.delim === '```');
          const result = this._compile(math, document);
          const mstyle = result.childNodes[0].childNodes.pop();
          mstyle.childNodes.forEach(child => result.appendChild(child));
          if (math.display) {
            result.attributes.set('display', 'block');
          }
          return result;
        }
      });
      MathJax.startup.defaultReady();
    }
  }
};
</script>
<script src="https://cdn.jsdelivr.net/npm/mathjax@4/startup.js"></script>
```

This overrides the AsciiMath `compile()` method to properly handle the two different delimiters.

26.4 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 `<`, `>` and `&` 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

TWENTYSEVEN

SPECIFYING THE SIZE OF HTML IN EXPRESSIONS

MathJax v4 allows you to embed HTML within your mathematical expressions. See the *texhtml* extension documentation page for more about how to do this in LaTeX expressions, and the *HTML in MathML token nodes* section for how this is done in MathML.

In a browser, MathJax can measure the size of the HTML that is embedded in a mathematical expression, and so can provide the proper amount of space for it within the equation, but in node applications not running in a browser, that is not possible, so MathJax provides a method for you to specify the size of the HTML explicitly. To specify the dimensions, add `data-mjx-hdw="H D W"` to the top-level HTML element inside the `<tex-html>` tag or MathML token element, where `H`, `D`, and `W` are the height, depth, and width of the HTML. They can be in any units, but em units will work best.

For example,

```
\(x + <tex-html><span data-mjx-hdw="0.75em 0.25em 1em">
<img src="mypic.png" style="width: 1em; height: 1em; vertical-align: -.25em" />
</span></tex-html> + y\)
```

could be used to insert an image with a given size into a TeX expression, while

```
<math>
  <mi>x</mi>
  <mo>+</mo>
  <mtext>
    <span data-mjx-hdw="0.75em 0.25em 1em">
      <img src="mypic.png" style="width: 1em; height: 1em; vertical-align: -.25em" />
    </span>
  </mtex>
  <mo>+</mo>
  </mi>y</mi>
</math>
```

would do the same in MathML.

How this attribute is used is determined by a new option to the output jax, `htmlHDW`, which can be set to `'auto'` (the default), `'ignore'`, `'use'`, or `'force'`. When set to `ignore`, the `data-mjx-hdw` attribute is ignored and MathJax will try to measure the size of the HTML directly. This works well in the browser, but not in the liteDOM, jsdom, linkedom, or other non-browser adaptors. The `force` option means that MathJax will use the `data-mjx-hdw` values and will surround the HTML with additional nodes that force the HTML to have the given dimensions. This would make the browser and node both have the same representation, not relying on the browser measurements. The value `use` means that MathJax will assume the `data-mjx-hdw` values are correct and will use them in its size computations without forcing the HTML to have the given dimensions. Finally, `auto` means that MathJax will determine which option to use; this will be `ignore` when in the browser and `force` when in node applications.

Having accurate values for the `data-mjx-hdw` attribute is crucial to the quality of the output. To that end, the following HTML file computes the needed values. These values depend on the surrounding font; the page below gives you a place to enter the HTML you want to measure and the surrounding font to use. Press the "Compute HDW" and the HTML is shown below together with modified HTML source that includes the needed `data-mjx-hdw` attribute. You can copy that and replace the original HTML with it. The original HTML is displayed with red lines at the right and left indicating the height and depth of the HTML, and with horizontal lines indicating the baseline position.

The code for this tool is the following:

```
<!DOCTYPE html>
<html>
<head>
<title>Compute HDW values for HTML in Token nodes</title>
<style>
h1 {font-size: 120%}
</style>
<script>
function GetHDW() {
  const html = document.querySelector("mjx-html");
  const content = html.getBoundingClientRect();
  const baseline = document.querySelector("mjx-baseline").getBoundingClientRect();
  const em = parseFloat(window.getComputedStyle(html).fontSize);
  const h = baseline.top - content.top;
  const d = content.bottom - baseline.top;
  const w = content.right - content.left;
  return [h, d, w].map(x => (x / em).toFixed(3).replace(/\.?0+$/, "") + "em").join(" ");
}
function ShowHDW() {
  const html = document.querySelector("#html").value;
  const content = document.querySelector("mjx-html");
  content.style.fontFamily = document.querySelector("#family").value;
  content.innerHTML = html;
  const output = document.querySelector("#output");
  if (content.childNodes.length > 1) {
    const span = document.createElement("span");
    while (content.childNodes.length) {
      span.append(content.lastChild);
    }
    content.append(span);
  }
  content.firstChild.setAttribute("data-mjx-hdw", GetHDW());
  output.innerHTML = content.innerHTML.replace(/&/g, "&amp;").replace(/</g, "&lt;");
}
</script>
<style>
mjx-measure {
  display: inline-block;
  border-left: 2px solid red;
  border-right: 2px solid red;
}
mjx-baseline {
  display: inline-block;
  height: 0;
  width: 0;
```

(continues on next page)

(continued from previous page)

```
}
mjx-html {
  display: inline-block;
}
mjx-line {
  display: inline-block;
  height: 0;
  width: 1em;
  border-top: 1px solid blue;
}
#input {
  display: inline-block;
}
#input textarea {
  margin-bottom: 3px;
}
#input input[type="button"] {
  float: right;
}
</style>
</head>
<body>

<h1>Compute HDW values for HTML in Token nodes</h1>

<p id="input">
<textarea id="html" cols="80" rows="10">
<span>HTML</span>
</textarea><br>
Font family: <input type="text" id="family" value="serif" />
<input type="button" value="Compute HDW" onclick="ShowHDW()" />
</p>
<h2>The HTML:</h2>
<p>
<mjx-line></mjx-line><mjx-measure><mjx-baseline></mjx-baseline><mjx-html>
 
</mjx-html></mjx-measure><mjx-line></mjx-line>
</p>
<h2>The HTML with the HDW attribute:</h2>
<p id="output">
</p>

</body>
</html>
```

CHAPTER

TWENTYEIGHT

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 and above. 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 not using a combined component but instead are performing your own in-line or file-based configuration, you select which renderer 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 and above. These may be available in future releases if there is demand for them. For now, the plain source can be shown by entering

```
MathJax.startup.document.state(0, true);
```

in the browser's developer console.

28.1 HTML Support

The *CommonHTML* (CHTML) 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. Version 3 only supports MathJax's limited TeX fonts, but version 4 introduces almost a dozen additional font options. See the *MathJax Font Support* section for more information on what fonts are available and how to select them.

The CHTML output component (*output/chtml*) is included in all the *Combined Components* that include *chtml* in their names. You can also load it explicitly by including `output/chtml` in the `load` array of the `loader` block of your MathJax configuration.

```
MathJax = {
  loader: {
    load: ['output/chtml']
  }
};
```

See *CommonHTML Output Processor Options* for information about the options that control the CommonHTML output processor.

28.2 SVG Support

The `SVG` output processor uses *Scalable Vector Graphics* to render the mathematics on the page. The SVG output mode offers high-quality results, 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. A 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 placed using the initial window width as well, and were fixed at those positions, so the equation number would not reposition with changes in window size. In version 3 and above, however, equation numbers now are relative to the container size, and move with changes in its width, 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 copied and pasted, as the output of the CommonHTML processor can.

The SVG output component (*output/svg*) is included in all the *Combined Components* that include *svg* in their names. You can also load it explicitly by including `output/svg` in the `load` array of the `loader` block of your MathJax configuration.

```
MathJax = {
  loader: {
```

(continues on next page)

(continued from previous page)

```
    load: ['output/svg']
  }
};
```

See *SVG Output Processor Options* for information about the options that control the SVG output.

28.3 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, support for MathmL in browsers was spotty, and the results varied across browsers and operating systems, sometimes requiring additional fonts or plugins to be used. This meant that the quality of the output could not be assured, and for these reasons, the MathML output renderer was removed in version 3.

Today, most modern browsers include support for MathML-Core, which is a limited subset of the full MathML specification. While this is an improvement over the past, MathML-Core does not include some important features that are needed by MathJax (the `<mlabeledtr>` element needed for numbered equations, and most of the table attributes needed for alignments, for example), so a native MathML output format is still not included in version 4.

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>
<style id="mjx-explorer-styles"></style>
<script>
MathJax = {
  //
  //  Load only TeX input and the contextual menu
  //
  loader: {load: ['input/tex', 'a11y/explorer', 'ui/menu']},
  startup: {
    //
    //  Set the styles needed for the expression explorer
    //  and disable the assistive MathML menu
    //
    ready() {
      MathJax.startup.defaultReady();
      const doc = MathJax.startup.document;
      const {StyleJsonSheet} = MathJax._.util.StyleJson;
      const css = new StyleJsonSheet(doc.constructor.speechStyles);
      const sheet = document.getElementById('mjx-explorer-styles');
```

(continues on next page)

(continued from previous page)

```
      sheet.textContent = css.cssText;
      doc.menu.menu.findID('Options', 'AssistiveMml').disable();
    },
    //
    //  Perform the initial typesetting (we don't have an output
    //  jax, so this isn't done automatically).
    //
    pageReady() {
      return MathJax.startup.document.renderPromise();
    }
  },
  options: {
    //
    //  Override the usual typeset render action with one that generates MathML output
    //
    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)
      ]
    },
    //
    //  Don't add assistive MathML
    //
    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>
<script defer src="https://cdn.jsdelivr.net/npm/mathjax@4/startup.js">
</script>
```

This example uses the *startup* component to load just the *input/tex*, *a11y/explorer*, 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 (already included by MathJax) 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 will make the container be a block-level element with some top and bottom margin.

The *ready()* function is used to create the CSS needed by the expression explorer. This uses a predefined `style`

element in the page and populates it with the CSS definitions contained in the MathDocument's *speechStyles* property.

Because there is no actual output jax, the startup component doesn't create the *MathJax.typesetPromise()* method and there is no initial typeset action performed. We use the *pageReady()* function to do our own typesetting by calling the document's `renderPromise()` function.

The example also takes several steps to disable the Assistive MathML extension that inserts hidden MathML for the usual output renderers. 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 *ready()* function also includes a line that disables the assistive-MathML item in the menu, so user's can't accidentally 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 that MathJax's internal MathML is based on the MathML3 specification. Most browsers that support MathML implement MathML-Core (which was developed several years after MathJax was initially written). The MathML-Core specification is more limited than MathML3, and does not support all the features that MathJax uses. For example, it does not include the `mathvariant` attribute in most cases, and MathJax uses that to implement `\mathbf`, `\mathfrak`, `\mathbb`, `\mathcal`, and the other font variants. That means that the output generated here may not produce the desired variants when the MathML is displayed in a browser. The *Convert Mathvariant to Unicode* example provides a way to work around that shortcoming in MathML-Core. Other limitations remain, however, such as the lack of support for most of the `mtable` element's attributes, which are used to implement LaTeX's various alignment environments, for example. So while it is possible to produce MathML output, the browser's native MathML rendering may not be up to the task for displaying it as well as MathJax would.

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 example does not, as it just uses the generic MathML. So the output generated here may not reproduce all the features available in the CHTML and SVG renderers. One would need to replace `MathJax.startup.toMML()` by a more sophisticated version that works around the limitations in MathML-Core in order to faithfully reproduce those.

CHAPTER

TWENTYNINE

LAZY TYPESETTING

MathJax offers an extension that is designed to improve the performance of pages with large numbers of equations. It implements a "lazy typesetting" approach that only processes an expression when it comes into view. This means that expressions will not be typeset when they are not visible, and your readers will not have to wait for the entire document to typeset, speeding up their initial view of the page. Furthermore, any expressions that are never seen will never be typeset, saving the processing time that would normally have been spent on those expressions.

This also helps with the situation where you may link to a particular location in your page (via a URL with a hash); typesetting the material above that point can cause the browser to change the scroll position, and so the user may not end up at the proper location in the page. With the lazy extension, the material above that point is not typeset until the user scrolls upwards, and so there is no position change.

To use the lazy typesetting extension, simply add it to your configuration as follows:

```
MathJax = {
  loader: {load: ['ui/lazy']}
};
```

This will adjust the typesetting pipeline to implement the lazy-typesetting functionality.

Lazy typesetting works best with SVG output, but the CHTML output is nearly as fast. With TeX input, the lazy extension makes sure that previous expressions are processed by TeX (though not output to the page) so that any macro definitions or automatic equation numbers are in place when the visible expressions are processed. Currently, documents that contain `\ref` or `\eqref` links may not yet work properly, since target equations may not have been typeset, and so the link location may not be marked in the document. In particular, forward references are unlikely to work if the target expression has not already been typeset. We hope to improve this situation in a future release.

29.1 Lazy Typesetting Options

Adding the *ui/lazy* extension to the `loader.load` array adds the following options to the MathJax configuration:

```
MathJax = {
  options: {
    lazyMargin: '200px',
    lazyAlwaysTypeset: null
  }
};
```

`lazyMargin: '200px'`

This gives the extent of the typesetting margin outside the visible viewport. When mathematics appears within this range of the viewport, it will be typeset. This allows typesetting to occur slightly before the math appears in the window, for a smoother effect.

`lazyAlwaysTypeset: null`

This gives an array of containers whose math expressions should always be typeset during the initial typesetting pass, rather than waiting for them to scroll into view. This may be useful if MathJax output appears in diagrams or other layout that must be sized and placed during initial page layout.

CHAPTER

THIRTY

AUTOMATIC LINE BREAKING

One of the important new features of MathJax version 4 is that of automatic line breaking, which was missing in v3. The support in v4 is an improvement over that in v2 in a number of ways. In particular, version 4 includes the option of breaking in-line expressions so that long expressions near the end of a line will automatically break and wrap to the next line. This is accomplished by allowing the browser to break the expressions where it needs to (following TeX's rules for what constitutes a valid in-line breakpoint). For display equations, version 4 provides support not only for automatic line breaking, but also for several other options for handling wide equations, including scaling the equation (to fit the container size), and scrolling if it is too wide. The page author can set the default, but there is also a new menu item where the viewer can switch the overflow handling to match their preferences. Version 4 also implements line-breaking of `<mtext>` elements (which are created by `\text{}` and other text-mode macros) , so long textual material can be broken automatically; this was not possible in version 2.

As part of the line-breaking support, a number of new TeX macros have been made available to control line breaks (to make them more or less desirable, or force or prevent them entirely). Also, support has been added for additional *array* environment template patterns that can be used to control the width and automatic line breaking of table cells, as well as insert text before or after every cell in a column, or adjust the spacing between columns. New macros for creating boxes of specific widths in which line breaking will occur are also available, and there are options for controlling justification and indentation of the text. Such boxes are also available in MathML via additional options for the `mpadded` element.

Finally, version 4 now attempts to break cells within tables based on the size of table as a whole (whereas v2 broke cells only if they individually were too wide for the container, and broke them to the container width regardless of the size of the rest of the table). Version 4 also includes the ability to control the vertical alignment of cells that include line breaks, so that, for example, in an *align* environment, if the left-hand side has line breaks, the cell will align on its bottom line, while if the right-hand side has break, the cell will align at its top line. This makes the flow across the columns more natural.

30.1 Display breaking

The algorithm used in version 4 for breaking displayed equations is based on the one from version 2, but is not identical to it. Unlike version 2, the results should be nearly identical between the CHTML and SVG output renders, and the code is set up so that the algorithm can be updated or even replaced much easier than in v2. We do have plans for improvements that we hope to make in the future.

The page author can control the way long expressions are handled using the new `displayOverflow` output jax configuration option, which can be set to `overflow`, `scale`, `scroll`, `truncate`, `linebreak`, or `elide`, though the latter is not yet implemented. The reader can override that default using the MathJax contextual menu, which has a new item in the *Math Settings* submenu for handling *Wide Expressions*. For MathML input, MathJax version 4 now honors the `overflow` attribute of the `math` element, so you can mark a single long expression for line breaking, or for scrolling, for example.

Note that there is now a new `output` configuration block that can be used to provide options that are common to both CHTML and SVG output, so that options you may have set for your default output jax will stay in effect when the user changes renderers via the contextual menu.

When `displayOverflow` is set to `linebreak`, the breaking is controlled by the settings in the `linebreaks` sub-block of the `output` (or `chtml` or `svg`) block. The default settings are

```
MathJax = {
  output: {
    displayOverflow: 'linebreak',  // break long lines
    linebreaks: {                  // options for when overflow is linebreak
      inline: true,                   // true for browser-based breaking of in-line␣
↪equations
      width: '100%',                  // a fixed size or a percentage of the container␣
↪width
      lineleading: .2,                // the default lineleading in em units
      LinebreakVisitor: null,         // The LinebreakVisitor to use
    }
  }
}
```

The last option is used to replace the line-breaking algorithm with a new one, so is a developer option, but the others are author-level settings that control things like how wide the lines are allowed to be, and how much extra space to put between lines.

30.2 In-line Breaking

In version 4, in-line expressions can be allowed to break automatically by the browser. This is controlled via the `inline` option of the `linebreaks` block described above. When `true` (the default), MathJax will arrange for in-line expressions to be broken into pieces so that the browser can move parts of the equation onto the next line, if they would otherwise extend beyond the width of the expression's container. In version 2, in-line expressions are only broken when the expression by itself would be wider than the container, and in that case, the expression would essentially act like it was inside a `<div>` element, so it badly disrupts the flow of the paragraph, and could cause misleading wrapping of text around the broken expression. This is no longer the case in version 4.

Note, however, that in order to do this, MathJax must make several separate elements containing math, and for SVG output in particular, several separate top-level `<svg>` elements. For this reason, node applications that are trying to create single SVG images for the mathematics would want to set `linebreaks.inline` to `false` to avoid that.

Finally, because the browser is doing the actual determination of the locations for in-line breaks, these breaks are chosen purely by how much of the expression can fit at the end of the line before the break. That is, the parameters that mark breakpoints as good or bad (described below) are not taken into effect; however, forced breaks and no-break markers are respected.

30.3 TeX Array Line-Break Column Types

To help support line breaking within cells of wide tables, MathJax v4 includes support for the preamble column declarations defined in the array TeX package. These include the traditional `c`, `l`, and `r` for alignment of the contents of the cell (centered, left, or right), but adds support for `p{width}`, `m{width}`, and `b{width}` for vertical alignment of a fixed-width column in which line-breaking will occur at the given width, as well as `w{align}{width}` and `W{align}{width}`. There is also new support for `>{...}` and `<{...}` for adding content that is put before or after

every entry in a column, as well as `@{...}` for replacing the inter-column space with the given content, and `!{...}` for replacing inter-column rules. Support for `|` and the non-standard `:` are improved so multiple copies of `|` and `:` now produce multiple rules that are close together. The `*{n}{...}` option for repeating a column declaration *n* times is also supported. Finally, non-standard `P{width}`, `M{width}`, and `B{width}` are defined that produce math-mode versions of their corresponding lower-case counterparts. The `\newcolumntype` macro for declaring new column specifications is also available.

Note that for `p`, `m`, `b`, `w`, `W`, `P`, `M`, and `B` columns, line-breaking will occur within the given column only if line-breaking is the active overflow setting. Otherwise, wide content will overflow the width, as in actual LaTeX.

30.4 Line-breaking macros in TeX

In MathML, <mo> and <mspace> items can be marked as either good or bad breakpoint options via the `linebreak="goodbreak"` or `linebreak="badbreak"` options, or linebreaks can be prevented via `linebreak="nobreak"` or forced with `linebreak="newline"`. In TeX, these can be controlled via the `\goodbreak`, `\badbreak`, `\nobreak`, and `\break` (or `\\`) macros. These will try to mark the operator that follows (or in some case precedes) the macro using the appropriate `linebreak` attribute. If there is no operator, then an empty one having the appropriate attribute will be introduced into the expression at that location. There is also the `\allowbreak` macro that inserts a breakpoint that can be used if one is needed.

The `\parbox[align]{width}{text}` macro has been added in v4 to provide a line-breaking context of a given width and vertical alignment (`t`, `b`, `c` for top, bottom, center (the default), with `m` allowed as an alias for `c`) for text-mode material. Previous versions of MathJax include `\vcenter{}` for vertical centering, and v4 adds `\vtop{}` and `\vbox{}` for material to be aligned on the top line or bottom line of the contents. In LaTeX, their content is text-mode, but in MathJax, they are in math mode (since MathJax mainly does math-mode, and for backward compatibility with the original `\vcenter{}` implementation). The width of these boxes can be controlled using `\hsize=<dimen>` within the box, so `\vtop{\hsize=10em ...}` would make a box that is 10em wide whose content is line broken and aligned on the baseline of the first line. Finally, the `\makebox[width][align]{text}` macro can also be used to produce a line-breaking text box of a given width and vertical alignment. (This complements the `\mathmakebox[width][align]{math}` macro already in the *mathtools* package.)

In addition, version 4 introduces a new non-standard `\breakAlign` macro that can be used to set the vertical alignment for the various cells, rows, or columns in an alignment. The format is `\breakAlign{type}{align}`, where `type` is one of `c`, `r`, or `t`, indicating whether the alignment is for the single cell in which it occurs, the row in which it occurs, or for the entire table, and `align` is one of `t`, `c`, `m`, `b`, for top, center, middle, or bottom. The difference between `c` and `m` is that `c` always centers the cell regardless of line breaks, while `m` only centers if there are line breaks, and otherwise aligns on the cell baseline. When `type` is `r` or `t`, then `align` can be a sequence of these letters giving the alignments to use in each entry in the row, with the last one being repeated if there are more columns than letters. When type is `t` the alignments are applied as row alignments to each row in the table.

For example, `\breakAlign{t}{bt}` could be used at the beginning of an alignment to make the baseline of the bottom line of the first column align with that of the top line of the second column. This is the default for *align* environments, as shown in the example above.

When line-breaking is enabled, you may want to have more control over how long lines of an alignment are broken. You can use `\hbox` or `\mbox` to avoid line breaks, but when you do allow breaks, you may want more control over indenting and alignment in such settings. For this reason, MathJax v4 introduces a non-standard `indentalign` environment that can be used within a cell of a table (or in any line-breaking context) to adjust the indentation amount and the horizontal alignment of any wrapped lines:

```
\begin{indentalign}[first][middle][last]{align}
  (long line of math)
\end{indentalign}
```

where `first`, `middle`, and `last` are optional dimensions that specify how much indentation to use for the first, middle, and last lines (where middle is any but the first or last lines). If only `first` and `middle` are provided, `last` will be the same as `middle`, and if only `first` is given, all three will use the same value. The `align` argument is one to three letters, each being one of `l`, `c`, or `r`, and these represent the alignments for the first, middle, and last lines. So

```
\begin{indentalign}[0em][2em]{l}
  x = A + B + C\\
    + D + E + F + G\\
    + H + I + J
\end{indentalign}
```

would left align all lines, and indent the second and third lines by 2em, when used in a context where line-breaking is in effect.

30.5 Breaking and Alignment in MathML

Control of line-breaking and alignment like that in TeX can be accomplished in MathML input using the new `data-break-align` attribute on the `mtable`, `mtr`, or `mlabeledtr` elements, or the `data-vertical-align` attribute for `mtd` elements. These can have values of `top`, `center`, `middle`, or `bottom` (repeated and space-separated for tables and rows).

The `data-vertical-align` attribute can be used on `msqrt`, `mroot`, and `mrow` elements as well to adjust how they are aligned when they contain line breaks. The default for roots is `bottom`, so that if line-breaks occur within a root, the root will align on its bottom line.

In TeX there is no direct control over this attribute within roots.

30.6 Options for <mpadded> elements

The various line-breaking boxes described above for LaTeX expressions are implemented via the MathML <mpadded> element. In order to facilitate that, MathJax v4 adds two non-standard attributes to the `mpadded` element: `data-overflow` and `data-align`. When `data-overflow="linebreak"` is used, the contents performs line-breaking to the width specified in the element's `width` attribute. (No other value for `data-linebreak` is implemented). The `data-align` attribute value can be `left`, `center` or `right`, to get the contents (line-broken or not) aligned to the left, center, or right of the specified width. You can use an <mstyle> element within the <mpadded> element in order to set the `indentshift`, `indentalign`, and similar attributes (for first and last lines) of the content, or can specify those attributes on the individual <mo> or <mspace> elements within the <mpadded> container.

For example:

```
<math xmlns="http://www.w3.org/1998/Math/MathML" display="block">
  <mpadded data-overflow="linebreak" data-align="right" width="5em" style="border: 2px␣
↪solid lightgrey">
    <mi>x</mi>
    <mo>=</mo>
    <mi>A</mi>
    <mo>+</mo>
    <mi>B</mi>
    <mo>+</mo>
    <mi>C</mi>
    <mo>+</mo>
```

(continues on next page)

(continued from previous page)

```
    <mi>D</mi>
    <mo>+</mo>
    <mi>E</mi>
    <mo>+</mo>
    <mi>F</mi>
    <mo>+</mo>
    <mi>G</mi>
  </mpadded>
</math>
```

would show a box with a grey outline whose content is broken into several right-aligned lines.

CHAPTER

THIRTYONE

MATHJAX FONT SUPPORT

MathJax version 4 includes support for a number of new font sets for MathJax, and changes the default font to one based on the New Computer Modern fonts, which offer support for a much larger range of characters than MathJax's original TeX font set, but is consistent with the look-and-feel of the original MathJax TeX fonts. The new set is slightly lighter, so will not seem so bold and will fit in better on Windows machines, without losing too much on linux, Mac OS, and iOS displays. The original MathJax TeX font set is also available as an option, for those who are unwilling to part with it.

There are 11 fonts available for MathJax v4:

Font Name	Original Source
mathjax-newcm	Based on New Computer Modern (now the default font)
mathjax-asana	A version of the Asana-Math font
mathjax-bonum	A version of the Gyre Bonum font
mathjax-dejavu	A version of the Gyre DejaVu font
mathjax-fira	A version of the Fira and Fira-Math fonts
mathjax-modern	A version of Latin-Modern
mathjax-pagella	A version of the Gyre Pagella font
mathjax-schola	A version of the Gyre Schola font
mathjax-stix2	A version of the STIX2 font
mathjax-termes	A version of the Gyre Termes font
mathjax-tex	The original MathJax TeX font

You can specify the font you want to use by setting the `font` option in the new `output` block of your MathJax configuration (where options common to both output renderers can be placed). For example,

```
MathJax = {
  output: {
    font: 'mathjax-stix2'
  }
};
```

will select the `mathjax-stix2` font. For in-browser use, this will obtain the font and its data from `cdn.jsdelivr.net` and no other configuration is necessary. For node applications, first install the font via

```
npm install @mathjax/mathjax-stix2-font
```

(add `-font` to the name of whichever font you want, or `-font-extension` for a font extension, and obtain it from the `@mathjax` scope); MathJax should find the font in your `node_modules/@mathjax` folder.

By default, MathJax components use the predefined `fonts` path to locate the fonts. That is, when the font is specified as `mathjax-stix2`, for example, MathJax will use `[fonts]/mathjax-stix2-font` as the path to the font. This path

is set to `https://cdn.jsdelivr.net/npm/@mathjax` for web applications and `@mathjax` for node applications. If you host your own copy of MathJax and its fonts, you can set this path to the location where the fonts are stored. For example,

```
MathJax = {
  loader: {
    paths: {
      fonts: '/mathjax-fonts',
  }
};
```

would tell MathJax to obtain fonts from the top-level `mathjax-fonts` directory on the server where web page came from.

It is also possible to configure the path to the fonts using the `fontPath` option of the `output` block. This should be set to a string that indicates where the font can be found; that string should include `%%FONT%%` in any part of the path where the font name needs to appear. For example,

```
MathJax = {
  output: {
    fontPath: '[fonts]/%%FONT%%-font'
  }
};
```

is the default path for MathJax components.

Finally, you can specify an explicit URL to a font as the font name in the configuration:

```
MathJax = {
  output: {
     font: 'https://cdn.jsdelivr.net/npm/@mathjax/mathjax-stix2-font'
  }
};
```

For those who wish to use the original MathJax font as it appears in version 3, specify the font as `mathjax-tex`.

The combined component files, like `tex-chtml.js` and `mml-svg.js`, include the new `mathjax-newcm` font as part of the component so that only one file needs to be downloaded. But if you want to use a different font, you probably don't want to download `mathjax-newcm` first and then the font you actually want to use. Instead, you should use a component ending in `-nofont.js`, for example, `tex-chtml-nofont.js`, so that the initial download is smaller, as it doesn't include `mathjax-newcm`.

31.1 Font Extensions

MathJax v4 also includes the ability to add new ranges of characters to an existing font, or to replace some characters with alternative ones. An extension may only apply to a specific font (if it relies on the existing characters to make stretchy assemblies, for example), but others may be able to apply to any font.

Currently, there are four extensions, and all can be applied to any of the fonts listed above.

Font Name	Original Source
mathjax-euler	A version of the Neo Euler font
mathjax-bbm	The bbm double-struck fonts
mathjax-bboldx	The bboldx double-struck fonts
mathjax-dsfont	The dsfont double-struck fonts

The last three of these are loaded automatically by the *bbm*, *bboldx*, and *dsfont* extensions, respectively, when they are added to the `load` array in the `loader` section of your configuration, or if you use `\require` to load the extension. They don't actually replace the original double-struck characters, but instead, place the new ones in a separate *pseudo-variant* used internally by MathJax, so are available only through the macros provided by the corresponding TeX extension.

For `mathjax-euler`, configure MathJax to load the given extension using the `fontExtensions` array of the `output` block of your configuration. For example,

```
MathJax = {
  output: {
    fontExtensions: ['mathjax-euler']
  }
};
```

would load the `mathjax-euler` font extension onto the default font being used.

31.2 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 can enhance the coverage in a particular font by combining characters that already exist in order to form new ones. For example, in the `mathjax-tex` font, which has a double integral (U+222C) but no quadruple integral (U+2A0C), MathJax an use two copies of the double integral to generate a quadruple integral.

If MathJax can't find or create a needed character in its fonts, it will look through a fallback chain for the font variant in use. For example, if an expression requests a double-struck letter for which no double-struck glyph is available, a bold-faced one will be used, if possible, otherwise, the normal version will be shown, if there is one.

When a character is not available anywhere in the fallback chain, 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 for 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 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.

Fortunately, the new fonts in v4 all have much greater character coverage than the original `mathjax-tex` font, so there should be far fewer instances where the fallback mechanisms come into play.

31.3 Dynamically Loaded Font Ranges

Because the new MathJax fonts include more extensive character coverage, meaning much more data is required, the fonts have been broken down into smaller pieces that can be loaded dynamically, rather than being one big data file, as was the case with version 3. This allows the initial download of MathJax to be smaller, while still accommodating rarely used glyphs for those who need them.

As a result, however, when the data for one of these ranges is needed, MathJax will pause and wait for the data to arrive from the CDN or from your server. That means that producing MathJax output is now potentially an asynchronous process, which was not the case in v3.

In version 3, as long as you pre-loaded all the TeX extensions that you needed, you could use synchronous calls to *MathJax.typeset()*, *MathJax.tex2svg()*, or the other similar functions. With the new (larger) dynamic fonts in version 4, that is no longer guaranteed to work. Instead, if you are using a font other than `mathjax-tex`, you should use the promise-based versions of these calls, like *MathJax.typesetPromise()* or *MathJax.tex2svgPromise()*, in order to properly handle the potential for dynamically loaded font data. Without this, you may get a *"retry" error*, which is what MathJax uses to mediate its asynchronous loading actions.

If you can not avoid using synchronous calls, then you may need to load all the font dynamic data up front using a single promise-based call before you start using MathJax synchronously. This can be done using

```
MathJax.startup.document.outputJax.font.loadDynamicFiles();
```

to load all the font dynamic data. This function returns a promise, and you should wait for it to resolve before calling any MathJax conversion functions using either `await` or the promise's `then()` method. For example, with the configuration

```
MathJax = {
  startup: {
    pageReady() {
      return MathJax.startup.document.outputJax.font
        .loadDynamicFiles()
        .then(() => MathJax.startup.defaultPageReady());
    }
  }
};
```

you will be able to use synchronous calls once the *MathJax.startup.promise* resolves, so you will only have to handle one asynchronous call and the rest can be synchronous.

Note, however, that this approach will load a *lot* of font data, and this can greatly slow down your initial page processing, especially on slow network connections like those for mobile devices. Only do this if you absolutely have to. It is *far better* to use the promise-based typesetting and conversion functions if you can.

If you know which font ranges you will need, it is possible to load only the ones you will be using, which will still allow synchronous typesetting, but not incur the startup penalty of loading *all* the data files. Here is a configuration that implements that approach:

```
MathJax = {
  fontFiles: ['calligraphic'],  // The dynamic font files to load
  startup: {
    pageReady() {
      const font = MathJax.startup.document.outputJax.font;
      const prefix = font.options.dynamicPrefix;
      const dynamic = font.constructor.dynamicFiles;
      const files = MathJax.config.fontFiles;
      return MathJax.loader
```

(continues on next page)

(continued from previous page)

```
        .load(...(files.map((name) => `${prefix}/${name}`)))
        .then(() => files.forEach((name) => dynamic[name].setup(font)))
        .then(() => MathJax.startup.defaultPageReady());
    }
  }
};
```

This example loads the `calligraphic` range at startup so that `\mathcal{}` can be used with synchronous typesetting calls. You can add more ranges to the `fontFiles` lists as needed. You can find the names of the font ranges (which vary from font to font), by entering

```
Object.keys(MathJax.startup.document.outputJax.font.constructor.dynamicFiles)
```

in the browser's developer console.

An alternative approach would be to create a custom build of MathJax that preloads additional ranges of characters. Examples based on MathJax components are given in the *Using Components Synchronously* section, and ones that use direct calls to the MathJax modules are given in the *Linking to MathJax Directly in Node* section. These are node-based examples, but they can be modified for browser use.

31.4 The MathJax Font Tools

MathJax needs to know a lot of information about each of the characters in the fonts that it uses, so MathJax has to provide the necessary font data; this font data is generated during the creation of the font npm packages and cannot be determined easily on the fly within a browser.

The tools for building the data needed by MathJax for your own font or font extension will be made available after version 4 is officially released. They were used to create these new fonts, but are not yet ready for public release, as they need cleaning up and documentation. But in the future, you will be able to generate an extension to an existing font (for example, to replace the letters and numbers with a different font while leaving all the rest of the characters unchanged), or produce a completely new font. So look for that functionality in the future.

CHAPTER

THIRTYTWO

DARK MODE SUPPORT

As of v4.1.0, MathJax provides support for browsers using dark-mode system settings. MathJax dialogs, menus, and expression explorer all will use a dark-mode color set automatically when dark-mode is enabled in the system preferences.

This works well when the webpage itself has dark-mode support. But if not, then the explorer dark-mode colors can make the text somewhat harder to read. For those pages, MathJax v4.1.1 and later provides an extension to disable dark mode.

To disable the dark-mode colors, include `ui/no-dark-mode` in the `load` array of the `loader` section of your MathJax configuration object:

```
MathJax = {
  loader: {load: ['ui/no-dark-mode']}
};
```

This will prevent dark mode from being used in dialogs and the explorer; menus will remain in dark mode, however, but that should not cause an issue, as the menu's dark color scheme also works well in light mode.

CHAPTER

THIRTYTHREE

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 and above 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.

Warning

While MathJax v3 still supported Internet Explorer version 11, MathJax v4 no longer supports any version of Internet Explorer. While IE may be able to handle some MathJax expressions, we no longer do any testing in IE, and make no accommodations to get MathJax to work with IE.

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.

33.1 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">
```

at the top of the <head> section of your HTML pages.

CHAPTER

THIRTYFOUR

MATHJAX CONFIGURATION OPTIONS

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. Some blocks may contain further sub-blocks. For example, the `tex` block can have a `macros` sub-block that pre-defines 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 + ']'
    }
  }
};
```

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 the *enclose* package 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.

34.1 Input Processor Options

34.1.1 TeX Input Processor Options

The options below control the operation of the *TeX input processor* that is run when you include `'input/tex'` 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
      ['\\(', '\\)']
    ],
    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
    numberPattern:                  // pattern for recognizing numbers
       /^(?:[0-9]+(?:\{,\}[0-9]{3})*(?:\.[0-9]*)?|\.[0-9]+)/,
    initialDigit: /[0-9.,]/,        // pattern for initial digit or decimal point for a␣
↪number
    identifierPattern: /^[a-zA-Z]+/, // pattern for multiLetterIdentifiers in \mathrm,␣
```

(continues on next page)

(continued from previous page)

```
↪etc.
    initialLetter: /[a-zA-Z]/,       // pattern for initial letter in identifiers
    tags: 'none',                    // or 'ams' or 'all'
    tagSide: 'right',                // side for \tag macros
    tagIndent: '0.8em',              // amount to indent tags
    tagAlign: 'baseline',            // The rowalign value to use for tag cells
    useLabelIds: true,               // use label name rather than tag for ids
    ignoreDuplicateLabels: false,    // prevent error messages for duplicate label ids?
    mathStyle: 'TeX',                // one of TeX, ISO, French, or upright
    maxBuffer: 5 * 1024,             // maximum size for the internal TeX string (5K)
    maxTemplateSubtitutions: 10000,  // maximum number of array template substitutions
    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),
    preFilters: [],                  // A list of pre-filters to add to the TeX input jax
    postFilters: [],                 // A list of post-filters to add to the TeX input␣
↪jax
  }
};
```

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 and above.

Note

Prior to version 3.2, the `multlineWidth` option used to be in the main `tex` block, but it is now in the `ams` sub-block of the `tex` block. Version 3.2 includes code to move the configuration from its old location to its new one, but that backward-compatibility code has been removed in version 4.

Note

The `digits` option has been renamed `numberPattern` in version 4.

Additional options are described in the *Options Common to All Input Processors* section.

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-base* components automatically add to this list the packages that they load. If you explicitly load additional 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 activates it by including it in the package list.

You can remove packages from the default list using `'[-]'` rather than `[+]`, as in the following 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 add dollar sign delimiters to the default list, causing 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.

> **Warning**
>
> 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. It is possible, however, to use a custom render action to look for such tags. The *Changes in the MathJax API* section includes an example of how to do this for the v2-style `<script type="math/tex">` tags.

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.

> **Warning**
>
> 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. It is possible, however, to use a custom render action to look for such tags. The *Changes in the MathJax API* section includes an example of how to do this for the v2-style `<script type="math/tex">` tags.

`processEscapes: true`

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 `\\$...$` to get a backslash just before in-line math). When `false`, `\$` will not be altered, and its dollar sign may be considered part of a math delimiter. Typically this is set to `true` if you enable the `$ ... $` 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{...}` and `\eqref{}` macros outside of math mode.

`processEnvironments: true`

When `true`, MathJax 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 the TeX input jax. When `false`, LaTeX environments will not be processed outside of math mode. Note that *any* environment will be picked up this way, and initiates display-style mathematics, not just those that would do so in LaTeX.

`numberPattern: /^(?:[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: {
    digits: /^(?:[0-9]+(?:\{,\}[0-9]*)?|\{,\}[0-9]+)/
  }
};
```

`initialDigit: /[0-9.,]/`

This gives a regular expression that tells what characters can appear as the first character in a number. Once one of these characters has been found, the `numberPattern` above is applied to the input string to see if a number is found.

`identifierPattern: /^[a-zA-Z]+/`

This gives a regular expression that determines what constitutes a single identifier within one of the macros that specifies a font style, like `\mathrm{}` or `\mathcal{}`, or within `\operatorname{}`. A string that matches this pattern will produce a single `<mi>` element in the internal MathML representation of your formula. Thus, `\operatorname{max}` will produce `<mi>max</mi>` rather than three separate `<mi>`, one for each letter.

`initialLetter: /[a-zA-Z]/`

This gives a regular expression that specifies what letters can initiate a multi-letter identifier inside macros like `\\mathrm{}` or `\operatorname()`. Once one of these characters has been found, the `identifierPattern` above ks applied to the input string to see if a multi-letter identifier is found.

`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.

tagAlign: 'baseline'

This specifies how equation tags should be vertically aligned with equations that include line breaks. Its value can be `'baseline'`, `'top'`, `'center'`, or `'bottom'`. The default is `baseline`, which is usually the baseline of the top line of the equation.

useLabelIds: true

This controls whether element `id` attributes for tags use the `\label` name or the equation number. When `true`, use the label, when `false`, use the equation number.

ignoreDuplicateLabels: false

Normally, if MathJax typesets two expressions that have the same `\label`, that will generate an error for the second equation indicating the duplicate label. Setting this to `true`, however, will prevent the error message from occurring. That can be useful in a setting where you have removed the previous equation, such as in an editor where you are retypesetting the same equation when the content is edited.

mathStyle: 'TeX'

This determines how single-letter upper- and lower-case Latin and Greek variable names are typeset. The value can be `'TeX'`, `'ISO'`, `'French'`, or `'upright'`, and the result is as describe in the following table:

mathStyle	latin	Latin	greek	Greek
TeX	italic	italic	italic	upright
ISO	italic	italic	italic	italic
French	italic	upright	upright	upright
upright	upright	upright	upright	upright

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.

maxTemplateSubstitutions: 10000

In an array environment preamble, it is possible to make a column declaration that loops infinitely. For example, `\begin{array}{c@{\\}c} a&b \end{array}` would cause MathJax to loop. This value prevents this from looping infinitely by limiting the number of template substitutions that can be applied to an array.

baseURL: URL

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. By default, it is either the URL for the current document if there is a `<base>` element, or an empty string if not.

formatError: (jax, err) => jax.formatError(err)

This is a function that is called when the TeX input jax identifies 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/catch` block). Your function should return the MathML tree that is used for the error, in MathJax's internal format. The `jax.mmlFactory.create()` function can be used to create such trees. For example,

```
jax.mmlFactory.create('mtext', {mathvariant: 'bold'}, [
  jax.mmlFactory.text('An error occurred!')
]);
```

would create the internal representation for

```
<mtext mathvariant="bold">
An error occurred!
</merror>
```

preFilters: []

This specifies a list of functions to run as pre-filters for the TeX input jax. Each entry is either a function, or an array consisting of a function followed by a number, which is the priority of the pre-filter (lower priorities run first). The functions are passed an object with three properties: `math`, giving the `MathItem` being processed, `document` giving the `MathDocument` for the math item, and `data` giving the `ParseOptions` object that holds the details of the input jax configuration. The pre-filters are executed when the TeX input jax is asked to parse a TeX expression, and before the TeX string is processed, so you can use a pre-filter to adjust the TeX string prior to it being parsed. The math item's `math` property is the original TeX string.

See the *MathJax Pre- and Post-Filters* section for examples of pre-filters.

postFilters: []

This specifies a list of functions to run as post-filters for the TeX input jax. Each entry is either a function, or an array consisting of a function followed by a number, which is the priority of the pre-filter (lower priorities run first). The functions are passed an object with three properties: `math`, giving the `MathItem` being processed, `document` giving the `MathDocument` for the math item, and `data` giving the `ParseOptions` object that holds the details of the input jax configuration. The post-filters are executed when the TeX input jax has finished parsing the TeX expression and has converted it to the internal MathML format. The math item's `root` property holds the root of the parsed MathML tree (the internal representation of the <math> element).

See the *MathJax Pre- and Post-Filters* section for examples of post-filters.

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 the `FindTeX` class, create an instance of it, and pass that to the TeX input jax to use instead of the usual one. 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 predefine macros for use in TeX expressions:

```
MathJax = {
  tex: {
    macros: {
      R: '\\mathbf{R}'
    }
  }
}
```

The options for the various TeX packages (that have options) are described in the links below:

- *ams Options*
- *amscd Options*
- *autoload Options*
- *bbm Options*
- *bboldx Options*
- *begingroup Options*
- *color Options*
- *configmacros Options*
- *dsfont Options*
- *mathtools Options*
- *noundefined Options*
- *physics Options*
- *require Options*
- *setoptions Options*
- *tagformat Options*
- *texhtml Options*
- *units Options*

Setting Options from within TeX Expressions

It is sometimes convenient to be able to change the value of a TeX or TeX extension option from within a TeX expression. For example, you might want to change the tag side for an individual expression. The *setoptions* extension allows you to do just that. It defines a `\setOptions` macro that allows you to change the values of options for the TeX parser, or the options for a given TeX package.

Because this functionality can have potential adverse consequences on a page that allows community members to enter TeX notation, this extension is not loaded by default, and can't be loaded by `\require{}`. You must load it and add it to the tex package list explicitly in order to allow the options to be set. The extension has configuration parameters that allow you to control which packages and options can be modified from within a TeX expression, and you may wish to adjust those if you are using this macro in a community setting.

34.1.2 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
    allowHtmlInTokenNodes: false,       // True if HTML is allowed in token nodes
    fixMisplacedChildren: true,         // True if we want to use heuristics to try to␣
↪fix
                                        //   problems with the tree based on HTML not␣
↪handling
                                        //   self-closing tags properly
    parseError: function (node) {       // function to process parsing errors
      this.error(this.adaptor.textContent(node).replace(/\n.*/g, ''));
    },
    verify: {                           // parameters controlling verification of MathML
      checkArity: true,                 //   check if number of children is correct
      checkAttributes: false,           //   check if attribute names are valid
      checkMathvariants: true,          //   check for valid mathvariant values
      fullErrors: false,                //   display full error messages or just error␣
↪node
      fixMmultiscripts: true,           //   fix unbalanced mmultiscripts
      fixMtables: true                  //   fix incorrect nesting in mtables
    },
    preFilters: [],                     // A list of pre-filters to add to the MathML␣
↪input jax
    mmlFilters: [],                     // A list of mathml-filters to add to the␣
↪MathML input jax
    postFilters: [],                    // A list of post-filters to add to the MathML␣
↪input jax
  }
};
```

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 applications (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.

allowHtmlInTokenNodes: false

HTML5 specifies that HTML can be included within token nodes in MathML. This is now supported in MathJax v4, so you can include images or form inputs, for example, in your mathematical expressions. Because this HTML is not sanitized in any way by MathJax, it is a potential security risk for sites that allow user-supplied MathML. For this reason, MathJax includes the `allowHtmlInTokenNodes` option, which is `false` by default. If you want to process HTML in MathML token nodes, set this option to `true`. See the *HTML in MathML token nodes* section for more details.

fixMisplacedChildren: true

Specifies whether MathJax should try to fix problems created by improper nesting of MathML tags. This can be due to a missing or extra close tag, or by using self-closing tags in an HTML document, where some browsers require explicit close tags for MathML.

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.

checkMathvariant: true

This specifies whether the values for the `mathvariant` attributes are checked for validity. If an invalid variant is used, MathJax can crash, so correct variants are important.

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- and 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>.

preFilters: []

This specifies a list of functions to run as pre-filters for the MathML input jax. Each entry is either a function, or an array consisting of a function followed by a number, which is the priority of the pre-filter (lower priorities run first). The functions are passed an object with three properties: math, giving the MathItem being processed, document giving the MathDocument for the math item, and data giving the serialized MathML string to be parsed. The pre-filters are executed only when the MathML input jax is asked to process a mathml string (such as when *MathJax.mathml2svg()* is called), or when the *forceReparse* option is set. When the MathML is taken directly from a document DOM, it is already parsed, and so is not a serialized MathML string.

See the *MathJax Pre- and Post-Filters* section for examples of pre-filters.

mmlFilters: []

This specifies a list of functions to run as MathML-filters for the MathML input jax. Each entry is either a function, or an array consisting of a function followed by a number, which is the priority of the pre-filter (lower priorities run first). The functions are passed an object with three properties: math, giving the MathItem being processed, document giving the MathDocument for the math item, and data giving the MathML DOM elements for the expression. The MathML-filters are executed just before the MathML input jax converts the DOM elements into MathJax's internal format. This can be used to manipulate the expression before it is processed.

postFilters: []

This specifies a list of functions to run as post-filters for the MathML input jax. Each entry is either a function, or an array consisting of a function followed by a number, which is the priority of the pre-filter (lower priorities run first). The functions are passed an object with three properties: math, giving the MathItem being processed, document giving the MathDocument for the math item, and data giving the root of the internal representation of the MathML tree (the internal form of the top-level <math> node). The post-filters are executed when the MathML input jax has finished converting it to the internal MathML format, but before the MathItem's root property is set.

See the *MathJax Pre- and Post-Filters* section for examples of post-filters.

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 the FindMathML class, create an instance of it, and pass that to the MathML input jax to use in place of the default one. A null value means use the usual 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 the MathMLCompile class, make an instance of it, and pass that to the MathML input jax to use in place of the default one. A null value means use the usual MathMLCompile class and make a new instance of that.

MmlFactory: null

The MmlFactory object instance the will override the default one. This allows you to create a subclass of the MmlFactory class, make an instance of it, and pass that to the MathML input jax to use in place of the default one. A null value means use the usual MmlFactory class and make a new instance of that.

34.1.3 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 or above). 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: {
    delimiters: [['`', '`']],  // The start/end delimiter pairs for asciimath code
    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
  }
};
```

Additional options are described in the *Options Common to All Input Processors* section.

Option Descriptions

`delimiters: [['`', '`']]`

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 add dollar sign delimiters to the default list, causing 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.

> **Warning**
>
> 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. It is possible, however, to use a custom render action to look for such tags. The *Changes in the MathJax API* section includes an example of how to do this for the v2-style `<script type="math/tex">`.

`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.

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 the `FindAsciiMath` class and pass that to the AsciiMath input jax to use in place of the usual one. A `null` value means use the default `FindAsciiMath` class and make a new instance of that.

34.1.4 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.

34.2 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.

34.2.1 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.

In addition to the options listed below, you can also include any of the options from the `output` block listed in the *Output Processor Options* section.

The Configuration Block

```
MathJax = {
  chtml: {
    matchFontHeight: true,  // True to scale the math to match the ex-height of the␣
↪surrounding font
    fontURL: URL,           // The URL where the fonts are found
    dynamicPrefix: URL,     // The URL where dynamic ranges of the font data are located
    adaptiveCSS: true,      // true means only produce CSS that is used in the processed␣
↪equations
  }
};
```

Option Descriptions

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 does not have its ex-height set properly (and not all fonts do), it can cause the math to *not* match the surrounding text.

While a `true` value 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).

fontURL: URL

This is the URL to the location where the MathJax fonts are stored. The `URL` is set up by the default font to point to its CDN location. For the `mathjax-newcm` font, the `URL` would be set to `https://cdn.jsdelivr.net/npm/@mathjax/mathjax-newcm-font/chtml/woff2`, for example.

While v3 included the fonts as part of the MathJax distribution, in v4, the fonts are in separate npm packages. Each font sets up its own location when it is loaded, and the default is to take the fonts from `cdn.jsdelivr.net`. If you are serving your own copy of MathJax, you may want to include your own copy of the fonts, and so may need to set this value accordingly.

dynamicPrefix: URL

This is the location where MathJax should look for font data that has to be loaded dynamically. The `URL` is set up by the default font to point to its CDN location. For the `mathjax-newcm` font, the `URL` would be set to `[mathjax-newcm]/chtml/dynamic`, for example, with the `[mathjax-newcm]` path being set to the CDN location.

Version 3 included all the font data in one file, but in v4, where the fonts include much greater character coverage, the fonts are broken into several smaller pieces that are loaded only when needed.

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.

34.2.2 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.

In addition to the options listed below, you can also include any of the options from the `output` block listed in the *Output Processor Options* section.

The Configuration Block

```
MathJax = {
  svg: {
    blacker: 3,           // the stroke-width to use for SVG character paths
    fontCache: 'local',   // or 'global' or 'none'
    useXlink: true,       // true to include xlink namespace for <use> hrefs, false to
↪not
  }
};
```

Note

The `internalSpeechTitles` attribute from v3 has been removed in v4.

Option Descriptions

`blacker: 3`

This specifies the stroke-width to use for SVG character paths in units that are 1/1000 of an em. Enlarging this makes the characters a bit bolder, but can also cause them to render poorly, as some details may begin to overlap and become unreadable. You probably don't want to go above 20 or so.

`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.

`useXlink: true`

When a font cache is used, MathJax employs `<use>` tags to access the character path definitions. Traditionally, the `href` attributes that reference the path IDs are required to be in the `xlink` namespace, and so appear as `xlink:href`. HTML5 has deprecated namespaces, so in HTML pages, they should appear as plain `href` attributes instead. The `useXlink` attribute determines whether the `xlink` namespace should be included in the `href` attributes or not.

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.

34.2.3 Options Common to All Output Processors

The following options are common to all the output processors listed above. These can be specified in the `output` block of your MathJax configuration (they apply to any output jax), or can be included in the configuration for the specific output jax that you are using. It is best to use the `output` section for these options, since then if your reader uses the MathJax contextual menu to switch renderers, they will apply to the new renderer as well.

The options are given here with their default values.

```
MathJax = {
  output: {
    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: 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'
    displayOverflow: 'overflow',  // default for overflow (scroll/scale/truncate/elide/
↪linebreak/overflow)
    linebreaks: {                 // options for when overflow is linebreak
      inline: true,               // true for browser-based breaking of inline equations
      width: '100%',              // a fixed size or a percentage of the container width
      lineleading: .2,            // the default lineleading in em units
    },
    font: '',                     // the font component to load
    fontPath: FONTPATH,           // The path to the font definitions
    fontExtensions: [],           // The font extensions to load
    htmlHDW: 'auto',              // 'use', 'force', or 'ignore' data-mjx-hdw attributes
    preFilters: [],               // A list of pre-filters to add to the output jax
    postFilters: [],              // A list of post-filters to add to the output jax
  }
};
```

Note

The `matchFontHeight` option is no longer available on the SVG output processor, so it is no longer listed here. It is now described among the CommonHTML output options.

Other options specific to an output renderer are listed in the sections linked at the top of this page.

34.2.4 Option Descriptions

scale: 1

The scaling factor for math compared to the surrounding text. The MathJax output processors try to match the ex-size of the mathematics with that of the text where it is placed, so that the lower-case letters in the mathematics are the same height as lower-case letters in the surrounding text. Note that this may mean that upper-case letters in the mathematics may not match those in the surrounding font, as not all fonts have the same height ratio between upper- and lower-case letters. You may want to adjust the results using this scaling factor to suit your situation. 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.

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 example, if you enter unicode characters directly, these may not be in MathJax's font, and so they will be taken from the font or fonts specified here.

mathmlSpacing: false

This specifies whether to use TeX spacing or MathML spacing rules 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.

displayOverflow: 'overflow'

This specifies how displayed equations that are too wide for their containers should be treated. The possible values are:

- `'scroll'` to use a horizontal scroll bar to allow the rest of the equation scroll into view.
- `'scale'` to scale the equation until it fits into its container.
- `'truncate'` to clip the expression at the container size.
- `'elide'` is not yet implemented.
- `'linebreak'` to insert line breaks to keep the expression within the container.
- `'overflow'`, to allow the expression to overflow the width of the container. This is the default.

Note that this option sets the `overflow` attribute of the underlying MathML expression, if there isn't one already.

The user can change this value globally using the MathJax contextual menu.

linebreaks: {...}

This block of options controls the line-breaking that is performed when the `displayOverflow` is set to `'linebreak'` or the user selects linebreaking in the MathJax contextual menu. The options include:

inline: true

When set to true, in-line equations will be allowed to break (at locations that TeX would allow for line-breaks). The browser will then break the mathematics when needed, if the expression extends beyond the container's width.

width: '100%'

Gives the width for where displayed equations should be broken, either as a fixed size (e.g. `'500px'` or `'20em'`), or as a percentage of the container's width (e.g., the default value of `'100%'`).

lineleading: .2

The amount of extra vertical space, in em units, to be inserted between the lines of a displayed equation when it is broken.

Note that in-line breaks can change when the window size changes, since they are handled by the browser; but displayed equations are broken when initially typeset, and the breaks are not altered after that unless you explicitly rerender the equation.

See the *Automatic Line Breaking* section for more details on controlling line breaking within expressions.

font: ''

This specifies the font to use from among the fonts available in MathJax, either as a name like `mathjax-stix2` or as a path to the font npm package, like `https://cdn.jsdelivr.net/npm/@mathjax/mathjax-stix2-font@4` for in-browser use, or `@mathjax/mathjax-stix2-font` for use in node.

See the *MathJax Font Support* section for more details about the fonts available and how to use them.

fontPath: FONTPATH

This specifies the path for locating fonts by name. The default is `https://cdn.jsdelivr.net/npm/@mathjax/%%FONT%%-font` in the MathJax components for the browser, and `@mathjax/%%FONT%%-font` in node applications. Any occurrences of `%%FONT%%` in the path will be replaced by the font name when the font is accessed.

fontExtensions: []

This gives a list of font extensions to load when the output jax is loaded. For example, setting this to `['mathjax-euler']` would load the euler font extension.

htmlHDW: 'auto'

This controls how MathJax handles the size of HTML code embedded in your mathematics when its top-level element has an `data-mjx-hdw` attribute that gives the size of the content. The possible values are:

- `'ignore'` to ignore the value of `data-mjs-hdw`.
- `'force'` to use the `data-mjx-hdw` values to surround the HTML with additional nodes that force the HTML to have the given dimensions. (This makes the result in node and the browser always be the same.)
- `'use'` to assume the `data-mjx-hdw` values are correct so that MathJax will use them in its size computations without forcing the HTML to have the given dimensions.
- `'auto'` to allow MathJax to determine which option to use; this will be `ignore` when in the browser and `force` when in node applications.

See *Specifying the size of HTML in Expressions* for more information, and for a tool for computing the values to use for the `data-mjx-hdw` attributes.

preFilters: []

This specifies a list of functions to run as pre-filters for the output jax. Each entry is either a function, or an array consisting of a function followed by a number, which is the priority of the pre-filter (lower priorities run first). The functions are passed an object with three properties: `math`, giving the `MathItem` being processed, `document` giving the `MathDocument` for the math item, and `data` giving the `mjx-container` DOM node for the math (empty at this point). The pre-filters are executed when the output jax is asked to typeset an expression, but before typesetting has occurred. The pre-filters can be used to adjust the internal MathML before any output is produced; the math item's `root` property holds the internal structure.

See the *MathJax Pre- and Post-Filters* section for examples of pre-filters.

postFilters: []

This specifies a list of functions to run as post-filters for the TeX input jax. Each entry is either a function, or an array consisting of a function followed by a number, which is the priority of the pre-filter (lower priorities run first). The functions are passed an object with three properties: `math`, giving the `MathItem` being processed, `document` giving the `MathDocument` for the math item, and `data` giving the `mjx-container` DOM node for the math. The pre-filters are executed when the output jax has completed typesetting the expression into DOM elements, but before other actions involving the DOM tree for the expression (such as adding event handlers, adding speech, inserting it into the page, etc.) have occurred. The `mjx-container` now holds the DOM tree for the typeset math.

See the *MathJax Pre- and Post-Filters* section for examples of post-filters.

34.2.5 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 the WrapperFactory class, make an instance of it, and pass and instance of that to the output jax to use in place of the usual one. A null value means use the default WrapperFactory class and make a new instance of that.

fontData: null

The FontData object instance to use for the font to use. This is usually obtained from a font package, such as MathJaxNewcmFont imported from @mathjax/mathjax-newcm-font/js/chtml.js. This allows you to override the default font with a different one. It is also possible to subclass one of the MathJax fonts, make an instance of that, and pass that to the output jax to use in place of its usual one. A null value means use the default FontData class (the mathjax-newcm font) and make a new instance of that.

cssStyles: null

The CssStyles object instance to use for collecting the CSS styles from the various MathML classes, the font, and so on. This allows you to create a subclass of the CssStyles class, make an instance of it, and pass that to the output jax in place of the usual one. A null value means use the default CssStyles class and make a new instance of that.

linebreaks.LinebreakVisitor: null

The LinebreakVisitor object class to use for breaking long displayed equations. This allows you to create a subclass of the LinebreakVisitor class and pass that to the output jax in place of the usual one. MathJax will make an instance of the class you pass it, or of its default class if this value is null.

34.3 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.

34.3.1 The Configuration Block

```
MathJax = {
  options: {
    skipHtmlTags: [            //  HTML tags that won't be searched for math
        'script', 'noscript', 'style', 'textarea', 'pre', 'code',
        'math', 'select', 'option', 'mjx-container'],
    ],
    includeHtmlTags: {         //  HTML tags that can appear within math
        br: '\n', wbr: '', '#comment': ''
    },
    ignoreHtmlClass: 'mathjax_ignore',    //  class that marks tags not to search
    processHtmlClass: 'mathjax_process',  //  class that marks tags that should be␣
↪searched
    compileError: (doc, math, err) => doc.compileError(math, err),
    typesetError: (doc, math, err) => doc.typesetError(math, err),
    renderActions: {...}
  }
};
```

34.3.2 Option Descriptions

skipHtmlTags: ['script', 'noscript', 'style', 'textarea', 'pre', 'code', 'math', 'select', 'option', 'mjx-container']

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.

The value associate with a tag is either a string, which replaces the tag in the math string, or a function of the form `(node, adaptor) => string` that takes two arguments, the DOM node matching the given tag and the current DOM adaptor, and returns the replacement string for the DOM node in the math string.

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: (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: (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 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), and during the promise-based versions of all these functions. 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 `getMetrics()` call for the `MathDocument` when the document is rendered, but does nothing during a `rerender()` or `convert()` call on 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`, and `update`. 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, and insert the output into the document.

You can add your own actions by adding new named actions to the `renderActions` object, or override existing ones by reusing 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.

See the *MathJax Render Actions* section for more information and examples.

34.3.3 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 instance 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 instance 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 the MmlFactory class and pass that to the document to use in place of the usual one. 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.

34.4 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*
- *Speech 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, when the *ui/menu* is loaded, you use the *Contextual Menu Options* to determine 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: {
        enrich: true,         // true to enable semantic-enrichment
```

(continues on next page)

(continued from previous page)

```
        collapsible: false,   // true to enable collapsible math
        speech: true,         // true to enable speech generation
        braille: true,        // true to enable Braille generation
        assistiveMml: false,  // true to enable assistive MathML
      }
    }
  }
};
```

Note

The `explorer` option has been removed in v4, and is replaced by the `speech` and `braille` options, which determine whether the explorer is active (either one will activate the explorer). The semantic-enrichment is controlled by the new `enrich` option, and is required for both the complexity computations and the speech and Braille generation; disabling enrichment effectively disables nearly all the accessibility tools at once.

In version 3, the accessibility tools were off and the assistive MathML was on by default, meaning users had to enable to explorer by hand if they wanted to use it. In version 4, the reverse is true: the accessibility extensions are included and enabled in all the combined components, and the assistive MathML is off. This means users will be able to explore expressions immediately without the need to change menu settings.

If you are not using a combined component, you can load the extensions explicitly using the *Loader Options*, but it is probably better to use the menu options above, so that if a user turns the extensions off, they will not incur the network and startup costs of loading the extensions they will not be using.

Note

In version 4, the MathJax contextual menu has been redesigned to give more prominence to the accessibility tools, and they are now at the top level of the menu rather than hidden in a submenu.

34.4.1 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 an `enrich` action to the document's default *renderActions* object.

The Configuration Block

```
MathJax = {
  options: {
    enableEnrichment: true,   // false to disable enrichment
    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 when the *semantic-enrich* extension is loaded. This is controlled automatically by the settings of the context menu, so you should use those to control semantic-enrichment if the menu component is present. If not, you can use it to disable semantic enrichment if the *semantic-enrich* component has been loaded automatically and you don't need it.

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 default behavior by providing a function that does whatever you want, such as recording the error, or replacing the original MathML with alternative MathML containing an error message.

Note

In version 3, the *semantic-enrich* extension handled both enrichment and speech generation. These two functions have been separated in version 4, and speech is now processed in the new *speech* extension described below. The *sre* block that was listed here in v3 has been moved to the *speech* extension.

34.4.2 Speech Extension Options

This extension coordinates the generation of speech strings Braille notation that are added to the HTML or SVG nodes within the page where they can be used by screen readers, or by the *a11y/explorer*. The *speech* extension adds an `attachSpeech` action to the document's default *renderActions* object.

The Configuration Block

```
MathJax = {
  options: {
    enableSpeech: true,       // false to disable speech strings
    enableBraille: true,      // false to disable Braille notation
    speechError: (doc, math, err) => doc.speechError(doc, math, err),  // called if␣
↪speech generation fails
    sre: {
      domain: 'mathspeak',    // speech rules domain
      style: 'default',       // speech rules style
      locale: 'en'            // the language to use (en, fr, es, de, it)
    },
    a11y: {
      speech: true,           // switch on speech output when enabled
      braille: true,          // switch on Braille output when enabled
    },
    worker: {
      path: 'path-to-bundle/a11y/sre',   // full path to bundle/a11y/sre (set␣
↪automatically)
      maps: 'path-to-sre/lib/mathmaps',  // full path to sre's speech rules
      worker: 'speech-worker.js',        // name of worker script to load as a webworker
      debug: false,                      // true to include debugging messages in the␣
```

(continues on next page)

(continued from previous page)

```
↪browser console about
                                        //   the communications between the page,␣
↪worker pool, and workers.
    },
  }
};
```

Option Descriptions

enableSpeech: true

This setting controls whether speech strings are generated and attached to the DOM elements within the page when the *speech* extension is loaded. This is controlled automatically by the settings of the context menu, so you should use those to control speech generation if the menu component is present. If not, you can use it to disable speech generation if the *speech* component has been loaded automatically and you don't need it.

enableBraille: true

This setting controls whether Braille labels are generated and attached to the DOM elements within the page when the *speech* extension is loaded. This is controlled automatically by the settings of the context menu, so you should use those to control Braille labels if the menu component is present. If not, you can use it to disable Braille generation if the *speech* component has been loaded automatically and you don't need it.

enrichError: (doc, math, err) => doc.enrichError(doc, math, err)

This setting provides a function that gets called when the speech or Braille generation fails for some reason. The default is to call the MathDocument's `speechError()` method, which simply prints a warning message in the browser console window. You can override the default behavior by providing a function that does whatever you want, such as recording the error.

sre: {...}

This block sets configuration values for the Speech-Rule Engine (SRE) that underlies MathJax's speech and Braille features. See the SRE documentation for more details.

a11y: {...}

This block gives boolean values that essentially duplicate the `enableSpeech` and `enableBraille` values above.

worker: {...}

This block gives parameters that control the speech generation, which is performed using webworkers so that this time-consuming process will not interfere with the responsiveness of the page. You should not need to change these.

34.4.3 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. 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 when it is loaded. The value is controlled automatically by the settings of the context menu, so you should use those to control the complexity computations if the menu component is present. If not, you can use it to disable the computations 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.

34.4.4 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
    enableExplorerHelp: true,           // set to false to disable the help icon and␣
↪dialog
    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
      help: true,                       // include "press h for help" messages on focus
      roleDescription: 'math',          // the role description to use for math␣
↪expressions
      voicing: false,                   // switch on speech output

      backgroundColor: 'Blue',          // color for background of selected sub-
```

(continues on next page)

(continued from previous page)

```
↪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 when it is loaded. The value is controlled automatically by the settings of the context menu, so you should use those to control whether expressions are explorable if the menu component is present. If not, you can use it to disable the explorer if the *explorer* component has been loaded automatically and you don't need it.

enableExplorerHelp: true

This controls whether the small "info" icon is displayed at the upper right-hand corner of an expression when it is focused. Setting it to `false` will prevent the icon from appearing, will disable the "press H for help" message from being spoken when an expression is first focused, and disables the help dialog when the "h" key is pressed during expression exploration. This means that users of assistive technology will be unable to obtain help on how to use MathJax's assistive features, so you should not set this to `false` if your pages may be viewed by users with accessibility needs.

The `a11y` options are all controlled by the MathJax contextual menu, when the menu component is present, so you should use the corresponding menu options to set these values in that case. If the menu component is not loaded, you can use the options below to control the explorer directly.

The options belong roughly to one of the following four categories:

Speech Options

speech: true

Determines whether speech output is produced. By default, speech is computed for every expression on the page, and will be voiced by a screen reader when the page is read, or when the explorer is started.

braille: true

Determines whether Braille output is produced. By default, Braille is computed for every expression on the page, and will be sent to a Braille output device when the page is read, or when the explorer is started.

subtitles: true

This option specifies 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 specifies whether Braille output will be displayed under the expression as it is explored.

help: true

This option specifies whether the explorer should voice "press h for help" when an expression becomes focused. This helps new users to realize that help is available, but experienced users may wish to disable this feature.

roleDescription: 'math'

This option specifies what description should be voiced by screen readers when reading a MathJax expression; for example, the expression `E = mc^2` might be read as "E equals m c squared, math". The value to use can be set using the MathJax contextual menu to one of several options, including no description.

voicing: true

This option determines whether MathJax will read expressions using the Browser's voice API during expression exploration. That can be useful for people who are not using a screen reader, but still want to hear the spoken expression.

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 *Speech 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, with 1 begin fully opaque, and 0 being totally transparent.

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, with 1 begin fully opaque, and 0 being totally transparent.

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.

34.4.5 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 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.

> **Note**
>
> In version 3, the *assistive-mml* extension was included in all the combined components, and was active by default. That is no longer the case in v4, where the other accessibility tools are included and enabled by default. Users who prefer the assistive MathML can turn off the semantic enrichment (which will disable the other tools), and turn on the assistive MathML using the MathJax contextual menu (in the *Options* submenu in the Accessibility section of the main menu.

The Configuration Block

```
MathJax = {
  options: {
    enableAssistiveMml: false
  }
};
```

Option Descriptions

enableAssistiveMml: false

This setting controls whether the *assistive-mml* extension is to run or not when it is loaded. The value is controlled automatically by the settings of the context menu, so when the menu component is present, you should use those to control whether assistive MathML is inserted. If the menu is not available, you can use this option to disable the assistive MathML if the *assistive-mml* component has been loaded automatically and you don't need it.

34.5 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 are 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`.

34.5.1 The Configuration Block

```
MathJax = {
  options: {
    enableMenu: true,          // set to false to disable the menu
    menuOptions: {
      settings: {
        showSRE: false;        // true to include semantic attributes in MathML output
        showTex: false;        // true to include original LaTeX commands in MathML␣
↪output
        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
        overflow: 'Scroll',    // how to handle math that is wider than its container
        breakInline: true,     // true to allow automatic line breaks with in-line math
        inTabOrder: true,      // true if tabbing includes math

        enrich: true,          // true if semantic-enrichment should be performed
        collapsible: false,    // true if complex math should be collapsible
        assistiveMml: true,    // true if hidden assistive MathML should be generated␣
↪for screen readers

        // also these a11y options from the explorer extension

        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
        help: true,                // include "press h for help" messages on focus
        roleDescription: 'math',   // the role description to use for math expressions
        voicing: false,            // switch on speech output

        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-
```

(continues on next page)

(continued from previous page)

```
↪expressions
        treeColoring: false,        // tree color expression

        magnification: 'None',      // type of magnification
        magnify: '400%',            // percentage of magnification of zoomed expressions

        infoType: false,            // show semantic type on mouse hovering
        infoRole: false,            // show semantic role on mouse hovering
        infoPrefix: false,          // show speech prefixes on mouse hovering
      },
      annotationTypes: {
        TeX: ['TeX', 'LaTeX', 'application/x-tex'],
        StarMath: ['StarMath 5.0'],
        Maple: ['Maple'],
        ContentMathML: ['MathML-Content', 'application/mathml-content+xml'],
        OpenMath: ['OpenMath']
      }
    }
  }
};
```

Note

The *a11y* options in the `settings` section above control the `a11y` settings of the *a11y/explorer*. The settings here override the ones in the `options.a11y` configuration block when the menu extension is loaded, so you should use the values here to control those settings in that case.

34.5.2 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, and in the *Explorer Extension Options* section.

`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.

34.5.3 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 input and 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.

34.6 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. For example, this allows you to prevent `javascript:` or `data:` URLs from appearing in `href` attributes that might otherwise cause potential security issues. This extension is designed to support sites that allow users to enter mathematics that is viewed by others, as, for example, in a question-and-answer website, or a blog with user comments.

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.

34.6.1 The Configuration Block

```
MathJax = {
  loader: {load: ['ui/safe']},
  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,
        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
    }
  }
};
```

34.6.2 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 allowed 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-`.

34.6.3 Developer Options

```
MathJax = {
  options: {
    safeOptions: {
      //
```

(continues on next page)

(continued from previous page)

```
      //  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]
      }
    }
  }
};
```

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.

34.7 Startup and Loader Options

MathJax's components system is based on two tools that handle 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.

34.7.1 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: {                                    // the path prefixes for use in␣
↪specifying components
      mathjax: Loader.getRoot(),
      fonts: (typeof window === 'undefined'
               ? '@mathjax'
               : 'https:cdn.jsdelivr.net/npm/@mathjax')
    },
    source: {},                                 // the URLs for components, when␣
↪defaults aren't right
    dependencies: {},                           // arrays of dependencies for each␣
↪component
    provides: {},                               // components provided by each component
    require: null,                              // function to use for loading␣
↪components
    pathFilters: [],                             // functions to use to process package␣
↪names
    versionWarnings: true,                      // whether to check components for␣
↪version compatibility
                                                //   with the version of MathJax that␣
↪is running
  }
};
```

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 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, then 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(), fonts: ...}

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.

The `fonts` path is predefined to point to the location where MathJax will load fonts. For node applications, it is set to `@mathjax` so that fonts are taken from the `node_modules/@mathjax` directory. For web applications, it is set to `https://cdn.jsdelivr.net/npm/@mathjax` so that fonts are loaded from that CDN.

When the TeX *require* extension is loaded, an additional `tex` path is created in order to be able to load the various TeX extensions. Other paths that may be predefined include `sre` and `mathmaps` for the locations of the speech-rule-engine files, `mml` for MathML extensions, and font-specific paths like `mathjax-newcm` for the locations of the files needed by the fonts.

You can define your own prefixes; for example,

```
MathJax = {
  loader: {
    paths: {custom: 'https://my.site.com/mathjax'},
    load: ['[custom]/myComponent']
  }
};
```

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']
  }
};
```

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',
      '[tex]/configmacros'
    ]
```

(continues on next page)

(continued from previous page)

```
  }
}
```

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, otherwise it should return nothing. If there is an error loading the component, it should throw an error.

If set to `null`, the default is to insert a `<script>` tag into the document that loads the component.

For use in CommonJS *node* applications, you can set this value to `require`, which will use node's `require` command to load components. E.g.

```
MathJax = {
  loader: {
    require: require
  }
};
```

For use in ESM *node* applications, you can set this value to use `import()`, as in

```
MathJax = {
  loader: {
    require: (file) => import(file)
  }
};
```

`pathFilters: []`

This is an array of functions that are used to process the names of components to produce the actual URL used to locate the component. There are built-in filters that perform actions like converting the prefix `[tex]` to the path for the TeX extensions, and adding *.js* to the end of the name, and so on. You can provide your own filters if

you need to manage the URLs in a different way. The array consists of entries that are either functions that take a data object as an argument, or an array consisting of such a function and a number representing its priority in the list of filters (lower numbers are earlier in the list). The data object that is passed to these functions is

```
{
  name: string,            // the current name for the package (this becomes the␣
↪url in the end)
  original: string,        // the original package name (should not be modified)
  addExtension: boolean,   // true if .js should be added to this name at some␣
↪stage in the filter list
}
```

The filter can change the *name* value to move it closer to the final URL used for loading the given package. The *original* property should be the original name of the package, and should not be modified.

The function should return `true` if the *name* should be further processed by other filters in the list, and `false` to end processing with the *name* now representing the final URL for the component.

There are four default filters: one that replaces *name* with its value in the `source` list, if any; one that normalizes package names by adding `[mathjax]/` if there is no prefix or protocol already; one that replaced prefixes with their values in the `paths` list; and one that adds `.js` if there is no extension. These have priorities 0, 10, 20, and 30, respectively, and you can use priorities (including negative ones) with your own functions to insert them into this list in any location.

`versionWarnings: true`

This determines whether a message is issued to the browser console when a component that is loaded was built using a different version of MathJax than the one that is currently running.

Component Configuration

In addition to the options listed above, individual components 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: {
    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 `package`, 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 takes no argument and is called when the component is loaded, but before the `ready()` function is called. It can be used to 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.

34.7.2 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,          // Called when components are loaded
    pageReady: Startup.defaultPageReady,  // Called when MathJax and page are ready
    document: document,      // The document (or fragment or string) to work in
    invalidOption: 'fatal',  // Are invalid options fatal or produce an error?
    optionError: Startup.defaultOptionError,  // Function used to report invalid options
    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
    adaptor: null,           // The name for the DOM adaptor to use from among those␣
↪loaded
    polyfillHasOwn: true,    // True if a polyfill for Object.hasOwn should be used when
                             //   it is not available natively
  }
};
```

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. Setting it to `false` means MathJax will not typeset automatically; you will have to call *`MathJax.typesetPromise()`* or a similar function yourself when you want the math to be processed.

ready: MathJax.startup.defaultReady

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, configuring `pageReady()` may be the better choice.

pageReady: MathJax.startup.defaultPageReady

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 when that is complete, but you can override it to do whatever you would like. You should 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.

invalidOption: 'fatal' // or 'warn'

This determines whether an invalid option will cause a fatal error (when set to `'fatal'`) that stops MathJax from running, or a warning (when set to `'warn'`) that allows MathJax to go on. Prior to version 3.2, invalid options were fatal, but this option now allows control over that behavior.

optionError: Startup.defaultOptionError

This option gives a function that is called whenever there is an invalid option provided by the user. It takes two string arguments, the first being the message, and the second being the name of the invalid option. The default function looks at the `invalidOption` value and if it is `'fatal'` it throws an error using the given message, otherwise it logs the message to the browser console, allowing further options to be processed.

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. There are several other DOM adaptors for use in *node*, as well.

polyfillHasOwn: true

This indicates whether MathJax should provide a polyfill for the `Object.hasOwn()` method when it is not available natively in the browser. It is defined in all modern browsers, but some browsers that are several years old may not have an implementation for it, so MathJax will provide one when needed unless this option is set to `false`.

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 document object, the DOM adaptor, and so on.

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 *The Configuration Variable* 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

THIRTYFIVE

MATHJAX IN DYNAMIC CONTENT

When MathJax is initially loaded, it will typeset any math that is available in the page. For static pages, that is all that you will need to do to make your mathematics appear and be accessible. Many pages, however, load or create content dynamically, and if that new material contains mathematics, MathJax will need to be told to typeset that new mathematics, as that is not done automatically by MathJax.

The following sections show how this can be done within the *MathJax Components* framework using the commands that are part of the `MathJax` global variable. For node applications or custom builds of MathJax that don't use MathJax Components, you would need to use the lower-level direct calls to the MathJax `MathDocument` methods that correspond to those commands.

35.1 Handling New Content

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 one of the *`MathJax.typeset()`* or *`MathJax.typesetPromise()`* functions. These cause MathJax to look for unprocessed mathematics on the page and typeset it, leaving unchanged any math that has already been typeset.

The first of these runs synchronously, but if the mathematics on the page uses `\require` or causes an extension to be auto-loaded (via the *autoload* component), or needs characters from a region of the font that hasn't already been loaded, this will cause the *`MathJax.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. See the *Handling Asynchronous Typesetting* section for more details.

Note

The *`MathJax.typeset()`* command corresponds to

```
(elements = null) => {
  const doc = MathJax.startup.document;
  doc.options.elements = elements;
  doc.reset();
  doc.render();
}
```

and *`MathJax.typesetPromise()`* corresponds to

```
(elements = null) => {
  const doc = MathJax.startup.document;
  return doc.whenReady(async () => {
    doc.options.elements = elements;
    doc.reset();
    await doc.renderPromise();
  });
}
```

Those not using MathJax Components can use these definitions, where `const doc = ...` is replaced by your own `MathDocument` object created by `mathjax.document()`.

35.2 Handling Content that Changes

Some web pages replace old content with new content in some circumstances. For example, a "book reader" may load individual pages to be displayed in its main content area, with each new page replacing the previously viewed one, or a page that allows users to enter content may include an editor with a preview that updates as the user types new content.

When the changing content includes typeset mathematics, special care must be taken to inform MathJax that previously typeset mathematics is being removed before doing so. This is because MathJax keeps track of the expressions that it typesets so that they can be updated if changes are made to the menu settings. For example, if the renderer is changed, MathJax needs to go back and re-render all the expressions, so it needs to know where they are in the page.

Information about the mathematics that MathJax has typeset is stored in a list of `MathItem` objects that is part of the `MathDocument` maintained by MathJax. If content containing typeset mathematics is removed from the page, the corresponding `MathItem` objects need to be removed from that list, otherwise MathJax will think that math is still in the page, which can lead to problems if MathJax tries to re-render those items. It also means that the list can grow unexpectedly large, and that the old typeset expressions are not freed from memory, causing MathJax's memory usage to grow. In a situation like an editor with preview, where the content is updated for each keystroke, that can lead to a rapid growth in memory and a corresponding decrease in performance over time.

To deal with changing content, MathJax provides a function that tells it to forget about math that it has previously typeset: *MathJax.typesetClear()*.

If you are removing a portion of your document that may include typeset mathematics, you should call this function **before** removing the content from the DOM so that MathJax can determine which expressions it contains and remove them from its internal list of expressions. If you fail to do this, MathJax's expression list will contain orphan expressions that are no longer part of the DOM.

Note

The *MathJax.typesetClear()* method corresponds to

```
(elements = null) => {
  const doc = MathJax.startup.document;
  if (elements) {
    doc.clearMathItemsWithin(elements);
  } else {
    doc.clear();
  }
}
```

Those not using MathJax Components can use these definitions, where `const doc = ...` is replaced by your own `MathDocument` object created by `mathjax.document()`.

Once you have called *MathJax.typesetClear()*, you can remove the elements that you passed it or clear their contents and replace them with other content. Then call *MathJax.typesetPromise()* to typeset that new content.

If your input format is LaTeX, then the new content will have access to any macro definitions or labels that were defined in any previous content, and automatic equation numbering will continue with the next number, even if you remove equations with earlier numbers. In an editor setting, where the same content is typeset over and over, this can lead to errors about multiply-defined labels, incorrect application of macros before they are supposed to be defined, and equation numbers going up on each re-rendering.

To overcome these problems, you can use the *MathJax.texReset()* method to remove any previously-defined labels, and optionally set the automatic equation numbering starting value.

Note

The *MathJax.texReset()* command corresponds to

```
(...args) => {
  const jax = MathJax.startup.document.inputJax.tex;
  jax.reset(...args);
}
```

Those not using MathJax Components can use these definitions, where `const jax = ...` is replaced by your TeX input jax instance.

To reset macro definitions, you can use the *begingroup* extension to isolate the definitions used for one typesetting pass from the following ones. Its `\begingroupSandbox` is one way to do that, which you can process using

```
MathJax.tex2mml('\\begingroupSandbox');
```

These techniques are illustrated in the example in the next section.

35.3 Editor Preview Example

The following code combines the mechanisms discussed above into an example that implements a basic editor with a preview that is updated on every change the the input area. This example uses the HTML that the user enters, updates an output area using that, and calls MathJax to process the expressions it contains. Of course, in practice, you would want to sanitize the user input to prevent the user from entering malicious code, so this is just the bare-bones version meant to highlight how to handle the MathJax update portion of the editor tasks.

The details are discussed after the code listing below.

```
<!DOCTYPE html>
<html>
<head>
<title>editor testing</title>
<style>
h1 {
  font-size: 133%;
  margin: 1em 0 .5em;
}
textarea {
  box-sizing: border-box;
  width: 100%;
  height: 15em;
  padding: 3px 5px;
}
#mathOutput {
  border: 1px solid #999;
  padding: 3px 5px;
}
</style>
</head>
```

(continues on next page)

(continued from previous page)

```
<body>

<h1>Enter HTML with mathematics here:</h1>
<textarea id="mathInput" placeholder="Type your math here..."></textarea>

<h1>Preview:</h1>
<div id="mathOutput"></div>

<script>
(function () {
  //
  //  The input and output areas
  //
  const input = document.getElementById('mathInput');
  const output = document.getElementById('mathOutput');

  let mjRunning = true;           // true when MathJax is running
  let updatePending = false;      // true if an update is needed after MathJax completes

  //
  //  Add a listener that either runs MathJax if it isn't already running,
  //  or records that an update is needed if it already is running.
  //
  document.getElementById('mathInput').addEventListener('input', function() {
    if (mjRunning) {
      updatePending = true;
    } else {
      updatePreview();
    }
  });

  //
  //  Update the preview area and typeset any math it contains
  //
  function updatePreview() {
    //
    //  Record that we are running MathJax and that no additional update
    //  is needed after that.
    //
    mjRunning = true;
    updatePending = false;
    //
    //  Forget about any old math expressions from the preview
    //
    MathJax.startup.document.clearMathItemsWithin([output]);
    //
    //  Reset any TeX labels or equation numbers
    //  Start a new sandbox for new macro definitions (and remove any old ones)
    //
    MathJax.texReset();
    MathJax.tex2mml('\\begingroupSandbox');
    //
```

(continues on next page)

(continued from previous page)

```
74      //  Update the preview HTML and typeset the math
75      //
76      output.innerHTML = input.value;
77      MathJax.typesetPromise()
78        .then(() => {
79          //
80          //  MathJax has completed, so is no longer running
81          //  If an update was needed while MathJax was running, update the
82          //    preview again.
83          //
84          mjRunning = false;
85          if (updatePending) updatePreview();
86        })
87        .catch((err) => console.error('Math typeset failed:', err));
88    }
89
90    //
91    //  The MathJax configuration
92    //
93    window.MathJax = {
94      loader: {load: ['[tex]/begingroup']},
95      tex: {
96        packages: {'[+]': ['begingroup']},
97        inlineMath: {'[+]': [['$', '$']]},
98      },
99      startup: {
100       pageReady() {
101         //
102         //  Do the initial typesetting and update the preview if
103         //  the textarea already contains content (e.g., on a page reload).
104         //
105          return MathJax.startup.defaultPageReady().then(() => {
106            mjRunning = false;
107            if (input.value) updatePreview();
108          });
109       }
110     }
111   };
112 })();
113 </script>
114 <script defer src="https://cdn.jsdelivr.net/npm/@mathjax@4/tex-chtml.js"></script>
115 </body>
116 </html>
```

The `style` element at lines 5 through 20 just set the look and size of the input and output areas and header, which are found at lines 24 through 28. The script that begins at line 30 defines the code that runs the editor. It is placed within a function that is called immediately after it is defined in order to make the editor's variables be local to the editor and not pollute the global namespace.

Lines 35 and 36 obtain the input and output elements for easy reference later.

Lines 38 and 39 define variables that are used to control when MathJax is called to do more typesetting. You don't want to call *`MathJax.typesetPromise()`* on every keystroke, because if the user is typing quickly, that could cause several typesetting calls to be made while a typesetting action is already being performed, plus it is not a good idea to

change the DOM that MathJax is actively updating while it is typesetting.

To avoid this, we use the variable `mjRunning` to specify when MathJax has been asked to typeset, but that typesetting hasn't completed yet. We won't queue any more typeset calls until the active one has finished. This is initially set to `true` so that no updates are performed until MathJax is loaded and ready (it is set to `false` later). The `updatePending` variable is used to specify that an update has been requested while MathJax is still typesetting, which means that when MathJax finishes typesetting, we should update again. That update may include several new characters that were entered while MathJax was typesetting, but we only update once, and we only change the DOM when MathJax is idle.

Lines 45 through 51 add an event listener to the `textarea` element that runs when its content changes (due to keystrokes, or copy and paste, etc.). If MathJax is currently running, we set the `updatePending` value so that we don't update now, but do so when MathJax is finished. Otherwise, we do the update immediately.

The `updatePreview()` function at lines 56 through 88 does the work of updating the preview area and typesetting its mathematics. It sets `mjRunning` to `true` so that if further changes occur to the input area, updates will be put off until after MathJax has typeset the current preview. We set the `updatePending` value to `false` since we are doing an update, and so we can tell if any updates are requested while MathJax is running.

Line 66 tells MathJax to forget about any mathematics that was previously typeset in the preview area. Without this, MathJax's list of typeset math would grow with each character typed, as it holds onto all the previous copies of the math that it had typeset in the past. This call must come before the DOM is changed, so that MathJax can tell which math is inside the material being removed.

Line 71 tells MathJax to forget about any *label* commands that it has processed in the past and to reset automatic equation numbering to start at 1 again. Line 72 tells MathJax to throw away any macro definitions from the previous preview and start fresh. (This doesn't remove any macros defined by auto-loaded extensions or ones loaded explicitly by *require*, however).

Line 76 replaces the preview output area with the content of the input area, and line 77 asks MathJax to typeset any mathematics it contains. This is a promise-based call, so the typesetting doesn't start immediately, and the function returns right away. The `.then()` function is performed when the typesetting completes. It sets the `mjRunning` variable to `false` and checks if any additional updates were requested while MathJax was typesetting, and if so, it does another `updatePreview()` call. If an error occurred during typesetting, line 87 traps that and reports the error.

The configuration at lines 93 through 111 tell MathJax to load the *begingroup* TeX extension (needed for `\begingroupSandbox`), and adds that extension to the list of packages that TeX should use. It also adds single dollar signs as in-line math delimiters. Finally, it sets up a *pageReady()* function that does the normal page-ready actions, unsets the `mjRunning` variable, and then checks if the input area has any content (as it will in some browsers if the page is reloaded, for example), and if so, it updates the preview to show that initial content.

Line 114 loads MathJax into the page.

CHAPTER

THIRTYSIX

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.

36.1 Getting Things Ready

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

- If you use `npm` or `pnpm`, you may want to install the `@mathjax/src` 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 are only going to use the webpacked components, then installing the `mathjax` package will be sufficient.
- 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 `mjs`, `cjs`, `bundle`, and `components` directories, either in the `node_modules/@mathjax/src` 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
```

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 acquire MathJax; if you used `git`, then you will need to change `node_modules/@mathjax/src` in the paths that include them.

36.2 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 several kinds of components you could build:

- A **combined component** that brings together several other components (the `tex-chtml` component is a combined component)
- An **extension component** that contains what is needed for one feature and can be loaded along with other components to add that feature to MathJax.
- A **custom build** of MathJax that uses only the pieces of MathJax that you need for a specialized setting.

We describe how you can create each of these in the links below.

36.3 Customized Component Examples

36.3.1 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 Component 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` or `pnpm` to install MathJax. If not, you may need to adjust the locations in the `import()` commands).

Listing 1: custom-mathjax.js

```
import {startup} from '@mathjax/src/components/js/startup/init.js';
import {Loader} from '@mathjax/src/js/components/loader.js';
import {insert} from '@mathjax/src/js/util/Options.js';

//
// Load the components that we want to combine into one component
//   (listed in the preLoaded() call below)
//
import '@mathjax/src/components/js/core/core.js';

import '@mathjax/src/components/js/input/tex-base/tex-base.js';
import '@mathjax/src/components/js/input/tex/extensions/ams/ams.js';
import '@mathjax/src/components/js/input/tex/extensions/newcommand/newcommand.js';
import '@mathjax/src/components/js/input/tex/extensions/configmacros/configmacros.js';
import '@mathjax/src/components/js/ui/menu/menu.js';

```

(continues on next page)

(continued from previous page)

```
//
// Load the output jax and the code for loading its font
//
import {loadFont} from '@mathjax/src/components/js/output/svg/svg.js';

//
// Load speech-generation code
//
import '@mathjax/src/components/js/a11y/util.js';

//
// Mark the components that you have loaded
//
Loader.preLoaded(
  'loader', 'startup',
  'core',
  'input/tex-base',
  '[tex]/ams',
  '[tex]/newcommand',
  '[tex]/configmacros',
  'output/svg',
  'ui/menu'
);

//
// Update the configuration's mathjax and sre paths and add the loaded TeX packages
//
insert(MathJax.config, {
  loader: {paths: {mathjax: 'https://cdn.jsdelivr.net/npm/mathjax@4'}},
  options: {worker: {path: https://cdn.jsdelivr.net/npm/mathjax@4/sre'}},
  tex: {
    packages: {'[+]': ['ams', 'newcommand', 'configmacros']}
  }
});

//
// Mark the MathJax version being used for this combined configuration
//
Loader.saveVersion('custom-mathjax.js');

//
// Do the normal startup operations
//
loadFont(startup, true);
```

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.

Note

This file uses ES6 `import` commands to load the MathJax modules. It is possible to use ES5 `require()` calls instead, if you wish. For example,

```
import {startup} from '@mathjax/src/components/js/startup/init.js';
```

could be replaced by

```
const {startup} = require('@mathjax/src/components/js/startup/init.js');
```

and similarly for the other `import` commands. Note that the MathJax `package.json` file is set up to route `@mathjax/src/js` to the MathJax `mjs` directory when used in an `import` command, and to the `cjs` directory when used in a `require()` statement, so you can use the same path in either case. Similarly `@mathjax/src/components/js` maps either to the `components/mjs` or `components/cjs` directory based on whether `import` or `require()` is used.

Line 25 causes the accessibility tools to be included in this combined component. For situations where these are not needed, leaving out lines 22 through 25 would mean that they would not be bundled into this component; but because the *ui/menu* component is included, its default menu settings would cause the accessibility components to be loaded individually at run time. To prevent that, you would need to change lines 46 to 50 to

```
insert(MathJax.config, {
  loader: {paths: {mathjax: 'https://cdn.jsdelivr.net/npm/mathjax@4'}},
  tex: {
    packages: {'[+]': ['ams', 'newcommand', 'configmacros']}
  },
  options: {
    enableSpeech: false,
    enableBraille: false,
    menuOptions: {
      settings: {
        enrich: false,
      }
    }
  },
}, false);
```

This turns off semantic enrichment, and disables speech and Braille generation.

Lines 45 and 56 hard codes the path to the location where MathJax components are stored and where to get the web-worker code for speech generation; these would override those settings if they were part of the MathJax configuration set by the page that loads this combined component. Similarly, the accessibility settings in the code snipped above would override any settings made in the web page, and the three TeX packages would always be included, even if the MathJax configuration from the page explicitly removed them. This is because the changes made by the `insert()` command are made *after* the page configuration is moved to `MathJax.config` (which occurs during the first `import` at line 1), so these override the page settings.

It is possible to not overwrite these values, though it requires an extra configuration file that you import before the other `import` commands. (In the case where you are using `require()` rather than `import`, you can put this code above the first *require()* in *custom-mathjax.js* rather than using a separate file.)

If you create the file `custom-mathjax-config.js` given below:

Listing 2: custom-mathjax-config.js

```
1 import {insert} from '@mathjax/src/js/util/Options.js';
2
3 const GLOBAL = typeof window === 'undefined' ? global : window;
4
```

(continues on next page)

(continued from previous page)

```
GLOBAL.MathJax = insert({
  loader: {
    paths: {
      mathjax: 'https://cdn.jsdelivr.net/npm/@mathjax@4',
    }
  },
  tex: {
    packages: ['base', 'ams', 'newcommand', 'configmacros'],
  },
  //
  // Uncomment this options block if you are NOT including the
  // accessibility extensions in the combined component.
  //
  /*
  options: {
    enableSpeech: false,
    enableBraille: false,
    menuOptions: {
      settings: {
        enrich: false,
      }
    },
  },
  */
}, GLOBAL.MathJax || {}, false);
```

Here, we obtain the `insert()` function from the `util/Options` file, which combines user configurations with default ones, and use it to set the global `MathJax` configuration variable to our default configuration with the user's `MathJax` values, if any, merged in. We don't have to set the `worker.path` in this case, since it is determined from the `mathjax` path that we have already set. (We needed to do it above because the worker path was set using the original default value of the `mathjax` path, not the new on at line 45.)

If you are removing the accessibility extensions from the combined component (by not importing `components/js/a11y/util.js`), uncomment the `options` block of the configuration to prevent the menu extension from loading them individually at run time.

Once this file is created, import it before any other imports, and remove the original lines 41 to 50, along with the import command that obtains the `insert` function. That should leave you with the following:

Listing 3: custom-mathjax.js

```
import './custom-mathjax-config.js';
import {startup} from '@mathjax/src/components/js/startup/init.js';
import {Loader} from '@mathjax/src/js/components/loader.js';

//
// Load the components that we want to combine into one component
//   (listed in the preLoaded() call below)
//
import '@mathjax/src/components/js/core/core.js';

import '@mathjax/src/components/js/input/tex-base/tex-base.js';
import '@mathjax/src/components/js/input/tex/extensions/ams/ams.js';
```

(continues on next page)

(continued from previous page)

```
import '@mathjax/src/components/js/input/tex/extensions/newcommand/newcommand.js';
import '@mathjax/src/components/js/input/tex/extensions/configmacros/configmacros.js';
import '@mathjax/src/components/js/ui/menu/menu.js';

//
// Load the output jax and the code for loading its font
//
import {loadFont} from '@mathjax/src/components/js/output/svg/svg.js';

//
// Load speech-generation code
//
import '@mathjax/src/components/js/a11y/util.js';

//
// Mark the components that you have loaded
//
Loader.preLoaded(
  'loader', 'startup',
  'core',
  'input/tex-base',
  '[tex]/ams',
  '[tex]/newcommand',
  '[tex]/configmacros',
  'output/svg',
  'ui/menu'
);

//
// Mark the MathJax version being used for this combined configuration
//
Loader.saveVersion('custom-mathjax.js');

//
// Do the normal startup operations
//
loadFont(startup, true);
```

This will allow the page that loads your combined configuration file to have full control over the MathJax configuration.

The Component Configuration File

Next, create a file `config.json` that includes the following:

Listing 4: config.json

```
{
  "webpack": {
    "name": "custom-mathjax",
    "dist": "."
  }
}
```

This file gives the name that will be used for this component (`custom-mathjax` in this case), and where to put the webpacked file (`"."` means the directory containing the `config.json` file). When the directory is the same as the one containing the component file, the packed component file will end in `.min.js` rather than just `.js`.

Most of the real work is done by the `@mathjax/src/components/webpack.config.mjs` file, which will be called automatically by the commands in the following section.

Building the Component

Once these files are created, 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/src/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.

Note

If you have changed the `import` commands to `require()`, then you will need to use the command

```
node ../node_modules/@mathjax/src/components/bin/makeAll --cjs
```

in order to tell `makeAll` to use MathJax's `webpack.config.cjs` file rather than the `webpack.config.mjs` one.

If you put the `custom-mathjax.min.js` file somewhere 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 defer src="custom-mathjax.min.js"></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).

36.3.2 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` or `\U{NNNN}` (where the `N` are hexadecimal digits, exactly four in the first case, or between 1 and 6 in the second) 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:

Listing 5: mml.js

```
import {HandlerType, ConfigurationType} from '@mathjax/src/js/input/tex/HandlerTypes.js';
import {Configuration}  from '@mathjax/src/js/input/tex/Configuration.js';
import {CommandMap} from '@mathjax/src/js/input/tex/TokenMap.js';
import TexError from '@mathjax/src/js/input/tex/TexError.js';
import {replaceUnicode} from '@mathjax/src/js/util/string.js';
import {VERSION} from '@mathjax/src/js/components/version.js';
import {Loader} from '@mathjax/src/js/components/loader.js';

/**
 * Check that we are loaded from the right version of MathJax
 */
Loader.checkVersion('[custom]/mml.min.js', VERSION, 'tex-extension');

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

/**
 * Allowed attributes on any token element other than the ones with default values
 */
const ALLOWED = new Set(['style', 'href', 'id', 'class']);

/**
 * 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];
```

(continues on next page)

(continued from previous page)

```
      const value = match[2] || match[3];
      if (Object.hasOwn(type.defaults, name) || ALLOWED.has(name) || name.substr(0,5)
↪=== 'data-') {
        attr[name] = replaceUnicode(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;
}

/**
 * Create a MathML token element of the given type
 */
function mmlToken(parser, name, type) {
  const typeClass = parser.configuration.nodeFactory.mmlFactory.getNodeClass(type);
  const def = parseAttributes(parser.GetBrackets(name), typeClass);
  const text = replaceUnicode(parser.GetArgument(name));
  const mml = parser.create('token', type, def, text);
  if (type === 'mi') {
    mml.setTeXclass = classORD;
  }
  parser.Push(mml);
}

/**
 *  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']
});

/**
 * The configuration used to enable the MathML macros
 */
const MmlConfiguration = Configuration.create(
  'mml', {
    [ConfigurationType.HANDLER]: {
      [HandlerType.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 function named `mmlToken` defined previously and the name of the MathML token element to create. The `mmlToken` function 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 from the TeX string, and any optional attributes, and creates the MathML element with those attributes, using the argument as the text of the element.

Note

This file uses ES6 `import` commands to load the MathJax modules. It is possible to use ES5 `require()` calls instead, if you wish. For example,

```
import {Configuration}  from '@mathjax/src/js/input/tex/Configuration.js';
```

could be replaced by

```
const {Configuration}  = require('@mathjax/src/js/input/tex/Configuration.js');
```

and similarly for the other `import` commands. Note that the MathJax `package.json` file is set up to route `@mathjax/src/js` to the MathJax `mjs` directory when used in an `import` command, and to the `cjs` directory when used in a `require()` statement, so you can use the same path in either case. Similarly `@mathjax/src/components/js` maps either to the `components/mjs` or `components/cjs` directory based on whether `import` or `require()` is used.

The Extension Configuration File

Next, create a file `config.json` that includes the following:

Listing 6: config.json

```
{
  "webpack": {
    "name": "mml",
    "libs": [
      "components/js/core/lib",
      "components/js/startup/lib",
      "components/js/input/tex-base/lib"
    ],
    "dist": "."
  }
}
```

This file gives the name that will be used for this component (`mml` in this case), an array of components that we assume are already loaded when this one is loaded (the `core`, `startup`, and `tex-base` components in this case), and the directory where we want the final packaged extension to go (`"."` means the directory containing the `config.json` file). When the directory is the same as the one containing the extension file, the packed extension file will end in `.min.js` rather than just `.js`.

Most of the real work is done by the `@mathjax/src/components/webpack.config.mjs` file, which will be called automatically by the commands in the following section.

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/src/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 described in the next section.

Note

If you have changed the `import` commands to `require()`, then you will need to use the command

```
node ../node_modules/@mathjax/src/components/bin/makeAll --cjs
```

in order to tell `makeAll` to use MathJax's `webpack.config.cjs` file rather than the `webpack.config.mjs` one.

Loading the Extension

To load your custom extension, you will need to tell MathJax where it is located, and include it in the list of files 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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4/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 defer src="https://cdn.jsdelivr.net/npm/mathjax@4/startup.js"></script>
```

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

36.3.3 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 MathJax web 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.

The Custom Build File

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

Listing 7: mathjax-speech.js

```
1  //
2  //  Load the desired components
3  //
4  import {mathjax} from '@mathjax/src/js/mathjax.js';                          // MathJax␣
   ↪core
5  import {TeX} from '@mathjax/src/js/input/tex.js';                            // TeX input
6  import {browserAdaptor} from '@mathjax/src/js/adaptors/browserAdaptor.js'; // browser DOM
7  import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';       // the HTML␣
   ↪handler
8  import {STATE} from '@mathjax/src/js/core/MathItem.js';                      // the␣
   ↪various states
9  import {SerializedMmlVisitor} from '@mathjax/src/js/core/MmlTree/SerializedMmlVisitor.js
   ↪';
10 import * as Sre from 'speech-rule-engine/js/common/system.js';              // Speech␣
   ↪generation
11
12 //
```

(continues on next page)

(continued from previous page)

```
//  Load the needed TeX extensions
//
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';

//
//  Register the HTML handler with the browser adaptor
//
RegisterHTMLHandler(browserAdaptor());

//
//  Initialize mathjax with a blank DOM.
//
const html = mathjax.document('', {
  InputJax: new TeX({
    packages: ['base', 'ams', 'newcommand']
  })
});

//
//  The visitor to produce serialized MathML
//
const visitor = new SerializedMmlVisitor();

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

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

  //
  //  A function to serialize the internal MathML format
  //
  toMML(node) {
    return visitor.visitTree(node, this.html);
  },

  //
  //  A function to convert TeX/LaTeX to a speech string
  //
  tex2speech(tex, display = true) {
    return this.Sre.engineReady().then(() => {
      return mathjax.handleRetriesFor(() =>
        this.toMML(html.convert(tex, {format: 'TeX', end: STATE.COMPILED, display}))
      )
```

(continues on next page)

(continued from previous page)

```
    }).then((mml) => this.Sre.toSpeech(mml));
  }
};

//
// Setup SRE's engine
//
Sre.setupEngine({domain: 'clearspeak', ...(CONFIG.sre || {})});

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

Unlike the component-based example in the *Building a Custom Component* section, this custom build calls on the MathJax source files directly. The `import` 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.

Note

This file uses ES6 `import` commands to load the MathJax modules. It is possible to use ES5 `require()` calls instead, if you wish. For example,

```
import {mathjax} from '@mathjax/src/js/mathjax.js';
```

could be replaced by

```
const {mathjax} = require('@mathjax/src/js/mathjax.js');
```

and similarly for the other `import` commands. Note that the MathJax `package.json` file is set up to route `@mathjax/src/js` to the MathJax `mjs` directory when used in an `import` command, and to the `cjs` directory when used in a `require()` statement, so you can use the same path in either case. Similarly `@mathjax/src/components/js` maps either to the `components/mjs` or `components/cjs` directory based on whether `import` or `require()` is used.

The Custom Configuration File

Next, create a file `config.json` that includes the following:

Listing 8: config.json

```
{
  "webpack": {
    "name": "mathjax-speech",
    "dist": "."
  }
}
```

This file gives the name that will be used for this component (`mathjax-speech` in this case), and the directory where we want the final packaged build to go. (`"."` means the directory containing the `config.json` file). When the directory

is the same as the one containing the control file, the packed build file will end in `.min.js` rather than just `.js`.

Most of the real work is done by the `@mathjax/src/components/webpack.config.mjs` file, which will be called automatically by the commands in the following section.

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/src/components/bin/makeAll
```

to process your custom build.

Note

If you have changed the `import` commands to `require()`, then you will need to use the command

```
node ../node_modules/@mathjax/src/components/bin/makeAll --cjs
```

in order to tell `makeAll` to use MathJax's `webpack.config.cjs` file rather than the `webpack.config.mjs` one.

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.

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 file will include all that you need to use the `MathJax.tex2speech()` command in your pages, provided they don't need any additional TeX extensions.

Note

If you do need additional extensions, you can add them into the `mathjax-speech.js` file above. Add `import` commands for the extensions you need, and add them into the `packages` list in the `new TeX()` command. Note that you can not use `\require` or autoload any extensions in this setup, since this is not a component-based implementation (it doesn't have the `loader` and `startup` modules needed for that), so every extension you plan to use must be loaded in advance.

To load your custom MathJax build, just add

```
<script defer src="mathjax-speech.min.js"></script>
```

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

```
const speech = await 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.

Alternatively, you can use the `then()` and `catch()` methods of the promise that is returned by `MathJax.tex2speech()`, as in

```
MathJax.tex2speech('\\sqrt{x^2+1}', true).then(
  (speech) => console.log(speech);
}).catch((err) => console.error(err));
```

to produce and handle the speech.

Configuring the Speech Generation

The speech-generation software can produce strings in a variety of languages, or in Braille notation, and this custom build of MathJax allows you to specify which language to use, or set other parameters of the speech-rule engine (SRE). This is done by setting the `MathJax` variable to a configuration that includes an `sre` block with the properties you want to customize. For example:

```
MathJax = {
  sre: {
    locale: 'fr'
  }
}
```

would tell the SRE to produce speech strings in the French language rather than English.

The complete list of options for the `sre` block can be found in the Speech-Rule Engine documentation.

Here is a complete page that converts a math expression into Nemeth Braille.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<meta content="width=device-width, initial-scale=1" name="viewport" />
<title>Use mathjax-speech to generate Braille</title>
<script>
  MathJax = {
    sre: {
      modality: 'braille',
      locale: 'nemeth'
    }
  }
</script>
<script src="mathjax-speech.min.js" defer></script>
<script type="module">
  console.log(await MathJax.tex2speech('\\sqrt{x^2+1}', true));
</script>
</head>
<body>
```

Of course, you could create a more sophisticated version that takes an expression typed by a user and processes that using `MathJax.tex2speech()`, then displays the result in the web page. That is left as an exercise for the interested reader.

Performing Actions at Startup

If you load `mathjax-speech.min.js` with the `defer` attribute, then your own code would need to wait for `mathjax-speech.js` to load before it can call `MathJax.tex2speech()`. One way to do that is to use a script with `type="module"` that follows the script that loads `mathjax-speech.js`, as is done in the example above.

Another way is to use the *ready()* function in the `Mathjax`, which will be run after the MathJax file has been loaded, and SRE has been initialized. For example

```
MathJax = {
  ready() {
    MathJax.tex2speech('\\sqrt{x^2+1}').then((speech) => console.log(speech));
  }
}
```

could be used to perform the speech conversion after everything is ready.

CHAPTER

THIRTYSEVEN

THE MATHJAX PROCESSING MODEL

The purpose of MathJax is to bring the ability to include mathematics easily in web pages to as wide a range of browsers as possible. Authors can specify mathematics in a variety of formats (e.g., *MathML*, *LaTeX*, or *AsciiMath*), and MathJax provides high-quality mathematical typesetting even in those browsers that do not have native MathML support. This all happens without the need for special downloads or browser plugins, or installation of fonts, etc.

MathJax is broken into several different kinds of components: page handlers, input processors, output processors, and the MathJax object classes that organize and connect the others. The input and output processors are called *jax*, and are described in more detail below.

When MathJax runs, it looks through the page for special markers that indicate mathematics; for each such marker, it locates an appropriate input jax, which it uses to convert the mathematics into an internal form (a MathML tree as javascript objects), and then calls an output jax to transform the internal format into HTML content that displays the mathematics within the page. The page author configures MathJax by indicating which input and output jax are to be used, and which extensions should be included.

While MathML notation is an XML format that consists of tags like those that make up the HTML language, it is not a format that is easy for page authors to write by hand. Many sites prefer to use a more natural input format like LaTeX or AsciiMath, especially when the sites allow readers to enter comments or post questions and answers. In this case, the mathematics is regular text within the page, and the author must indicate what parts of the text are mathematics by using special *delimiters* to mark the start and of the expressions within the text. For example, LaTeX expressions may be delimited by `\(...\)` for in-line expressions, and `\[...\]` or `$$...$$` (though MathJax can be configured to use other ones as well).

See the *The Configuration Variable* section for details about configuring the math delimiters. See the *TeX and LaTeX Support*, *AsciiMath Support*, or *MathML Support* sections for information about using those formats, and for important caveats about how to include mathematical expressions within an HTML page.

Internally, the processing of the page is mediated by an object called a `MathDocument`. This receives the HTML document that it is to process, and the input and output jax to be used in processing that page, along with any configuration options to control that processing. The individual expressions are representing internally using `MathItem` objects that hold pointers to the locations of the math, the math strings to be processed, and other information needed by MathJax to handle the individual expressions.

These `MathItem` objects are stored in a list within the `MathDocument` so that they can be reprocessed, if needed. For example, changes in the MathJax contextual menu may require that the math be reprocessed, e.g., if the renderer is changed, of if speech settings change.

This list must be kept up to date, so if you remove sections of the page that may contain mathematics, you should inform MathJax of that beforehand. The *Updating Previously Typeset Content* section discusses this in more detail. Note that if you typeset expressions by hand using the *conversion functions*, they will not be added into the math list, and so will not participate in any changes request by the contextual menu.

37.1 The components of MathJax

The main components of MathJax are its input and output jax, the MathDocument, which coordinates the actions of the other components.

Input jax
: These are associated with the different input formats (TeX/LaTeX, MathML, and AsciiMath), and includes information about what delimiters it uses, and how to locate math within the document. The main role of the input jax is to convert the math notation entered by the author into the internal format used by MathJax, which is essentially MathML (represented as JavaScript objects). So an input jax acts as a translator of its mathematical notation into MathML.

Input extensions
: A number of extensions are available for the TeX input jax that implement optional LaTeX macros or features. The MathJax configuration object can be used to load them explicitly, or the *autoload* extension may load some of them automatically when they are needed. The non-standard `\require` macro can be used to load an extension by name from within a typeset expression (see the *require* page). The *The TeX/LaTeX Extension List* gives the names of the various extensions and links to their descriptions and configuration options. See the *TeX and LaTeX extensions* section for information about loading extensions and configuring the TeX input jax to include them.

  The MathML input jax also supports extensions, though there currently is only one, the *mml3* extension that gives experimental support for the MathML3 elementary math tags. See the *mml3* section for details.

Output jax
: These convert the internal MathML format into a specific output format. For example, the CHTML output jax uses HTML with CSS styling to lay out the mathematics, while the SVG output jax use Scalable Vector Graphic (SVG) elements to do so. Output jax could be produced that render the mathematics using HTML5 canvas elements, for example. The MathJax contextual menu can be used to switch between the output jax that are available.

Fonts
: In v4, several different fonts are available for the output jax. See the *MathJax Font Support* section for details. The tools for creating the data needed by MathJax to support a font will be made available so that page authors can their own fonts or font extensions for use with MathJax.

Contextual Menu
: This component gives the reader access to several important informational options (such as viewing or copying the mathematical notation, or generating an SVG image from an expression), along with options that modify how MathJax operates. These include control over how long lines are treated, which render to use, and whether speech or other assistive extensions are to be activated. While it is possible to disable the MathJax contextual menu, doing so will impair the ability of your users to control MathJax, and in particular, users with accessibility needs may not be able to make the changes they need to support their particular disabilities.

MathDocument
: This is the internal object that handles the coordination of all of MathJax actions on a particular HTML page. It contains the input and output jax that are in use, and maintains a list of the expressions that have been typeset in the page. It maintains a list of actions to take when the page, or a portion of it, is to be typeset (see the *renderActions* description for details). For applications using the *MathJax Components framework*, `MathJax.startup.document` gives the `MathDocument` instance being used.

37.2 The MathJax API Documentation

Many of the important details about the MathJax Application Programming Interface (API) are described within these documentation pages, but certainly not every function and object within the MathJax code.

The *Examples of MathJax in a Browser* and *Examples of MathJax in Node* pages give links to most of the examples within these documentation pages that illustrate the MathJax API, along with links to repositories containing more

examples of web-based and node-based applications.

The MathJax source code internals are documented using jsDoc comments, and the resulting HTML pages are available on line.

CHAPTER

THIRTYEIGHT

SYNCHRONIZING YOUR CODE WITH MATHJAX

MathJax provides several mechanisms for synchronizing your code with the actions taken by MathJax, including function that return promises, a processing queue to which you can add your own actions, filter queues for the input and output jax that allow you to run functions before or after they process the math, and configuration options that allow you to perform actions at startup when MathJax is ready to process math. All of these give you the ability to hook into MathJax's workflow, as described in the sections below.

38.1 MathJax Promises

MathJax version 2 used *queues*, *callbacks*, and *signals* as a means of coordinating your code with the actions of MathJax. Version 3 and above use the more modern tool know as a promise to synchronize your code with MathJax. Many of the of MathJax's functions return promises that you can use to tell when MathJax completes the actions that you have requested, allowing you to perform code that requires the results of those actions. These functions include

- *MathJax.typesetPromise()*
- *MathJax.tex2mmlPromise()*
- *MathJax.tex2chtmlPromise()*
- *MathJax.tex2svgPromise()*
- *MathJax.mathml2chtmlPromise()*
- *MathJax.mathml2svgPromise()*
- *MathJax.asciimath2mmlPromise()*
- *MathJax.asciimath2chtmlPromise()*
- *MathJax.asciimath2svgPromise()*
- *MathJax.whenReady()*

for typesetting and converting math when you are using *The MathJax Components* framework, either in web pages or in node applications. The first two and last one are described in the *Typesetting Mathematics* and *MathJax in Dynamic Content* sections, with the remaining eight are discussed in the *Converting a Math String to Other Formats* section.

For node applications that use direct access to the MathJax modules, there are

- *mathDocument.convertPromise()*
- `mathDocument.renderPromise()`
- `mathDocument.rerenderPromise()`
- `mathDocument.whenReady()`

- *mathjax.handleRetriesFor()*

as methods of the MathDocument instance produced by a mathjax.document() call.

The first of these (described in the *The Basics of Linking Directly to MathJax* section), is the function that underlies the eight conversion functions in the first list above. The second underlies the *MathJax.typesetPromise()* call, and the third is used to rerender the output without re-running the input processor (e.g., when the output jax changes, or if the speech or explorer settings are changed in the MathJax menu). The mathDocument.whenReady() method is called by *MathJax.whenReady()*, and can be used to queue actions for MathJax to perform when it has finished any pending actions that have already been queued. This is described in more detail in the *Handling Asynchronous Typesetting* section.

All of these promise-based functions return a promise that completes when the typesetting or conversion actions have completed. You can use that promise to synchronize your own code that relies on the results of those actions with the MathJax calls that produce those results. That can be done by using a .then() clause on the returned promise, or by using await in an async function to wait for the promise to resolve before going on.

38.2 MathJax Render Actions

A MathDocument instance contains a sequence of functions called *renderActions* that are used during the typesetting process to perform the actions needed to locate expressions in the page, compile them into the internal format, obtain font metrics for the surrounding text, typeset the mathematics, insert the typeset math back into the page, add menu actions, and so on. Loading some components will add new functions into this list. For example, the *a11y/semantic-enrich* component adds a render action that computes the semantic-enrichment for each expression. All the typesetting and conversion functions listed above use this collection of functions to perform their typesetting or conversion operations.

The *renderActions* configuration option provides a means of linking your own functions into this processing pipeline at any point, replacing existing steps with your own custom versions, or even removing them to trim down what MathJax does.

The renderActions configuration object consists of one or more name: value pairs (separated by commas), where the name gives an identifier for the action, and the value is an array consisting of a number followed by zero, one, or two functions, and an optional boolean, as described below.

The number gives the priority of the action in relation to the other actions that are already defined. The functions for the lowest numbers are performed first. The number may be negative, or have decimal places. The existing priorities are found in the STATE variable from the ts/core/MathItem.ts file or the MathJax._.core.MathItem object (when using MathJax Components). These include the following:

Name	Priority	Identifier
UNPROCESSED	0	
FINDMATH	10	find
COMPILED	20	compile
CONVERT	100	
METRICS	110	metrics
RERENDER	125	
TYPESET	150	typeset
INSERTED	200	update
STYLES	201	styles
LAST	10000	

Here, the *name* is the value's key in the STATE object that has the given *priority*. The *identifier* is the key in the renderAction object for the action with the given priority. Some states are for reference only and do not

have associated actions; e.g., CONVERT and RERENDER are only used to tell where a *mathDocument.convert()* or mathDocument.rerender() action should start in the list of render actions, while UNPROCESSED and LAST mark the usual start and end of the processing list, though values outside that range are allowed.

Some extensions add new render actions. These include

name	Priority	Identifier	Extension
CONTEXT_MENU	170	addMenu	ui/menu
	205	getMenu	ui/Menu
	1	checkLoading	ui/menu
LAZYALWAYS	13	lazyAlways	ui/lazy
ENRICHED	30	enrich	a11y/semantic-enrich
ATTACHSPEECH	210	attachSpeech	a11y/speech
COMPLEXITY	40	complexity	a11/complexity
EXPLORER	230	explorable	a11y/explorer

You may use these values to determine the appropriate priorities to use for your own actions. Actions with the same priority will be performed in the order in which they were added to the render action list for the document by you and the components you have loaded. The newState() function from the ts/core/MathItem.ts file can be used to create a new named state value like the ones above. Otherwise, you can give priorities as numeric values, or as an existing STATE, possibly value plus (or minus) a given number.

Following the priority in the action's definition array should be zero, one, or two functions. The first is called whenever the page is typeset via a call to *MathJax.typeset()*, mathDocument.render() or mathDocument.rerender(), or their promise-based versions, and is passed the MathDocument object in which it is running. Usually this function will loop through the MathItem objects stored in the mathDocument.math list and perform its action on each of those individually.

The second function is called whenever a math expression is updated individually via its rerender() method (e.g., when a subexpression is collapsed, or a toggle item is toggled, or the speech ruleset is changed), or during a *mathDocument.convert()* call is made, or any of the conversion functions based on it (e.g., *MathJax.tex2svg()*), or any of the promise-based versions of this is called. The function for this action is called with two arguments: the MathItem on which it is to operate, and the MathDocument that holds that item.

If either of these functions is given as an empty string, then that action is not performed (that way, you can define a typeset action without a rerender action, for example), and if it is a non-empty string rather than a function, that is taken to be the name of a method of the MathDocument (for the first function) or MathItem (for the second function) to be called for that action instead. If the function is missing, then the identifier used for the action in the renderAction object is taken as the method name to use.

Finally, the optional boolean value tells whether the second function is to be use for both convert() and rerender() actions (when true, the default), or only for rerender() actions (when false).

To remove a render action, set its identifier to an empty array. For example,

```
MathJax = {
  option: {
    renderActions: {
      addMenu: [],
      getMenu: [],
      checkLoading: [],
    }
  }
}
```

would disable all the MathJax menu actions. This would disable to the MathJax menu, though it is easier to set `enableMenu` to `false` to accomplish that.

To replace a render action, set its identifier to the definition you would like to use instead. For example, you could replace the `typeset` action with one that generates MathML output in the page rather than the CHTML or SVG output from MathJax. The *MathML Support* section for an example of how to do this.

38.2.1 A Render Action for Tooltips

Here is an example of defining your own render action that adds the original TeX notation as a tooltip on the MathJax output:

```
MathJax = {
  addTooltip(item) {
    const adaptor = MathJax.startup.adaptor;
    adaptor.setAttribute(math.typesetRoot, 'title', item.math);
  },
  options: {
    renderActions: {
      addTooltip: [175,
        (doc) => {
          for (const item of doc.math) {
            MathJax.config.addTooltip(item);
          }
        },
        (math, doc) => MathJax.config.addTooltip(math),
      ]
    }
  },
}
```

This uses a function `addTooltip()` that is stored in the MathJax configuration (and otherwise ignored by MathJax), and specified a new `addTooltip` render action at priority 175 (after typesetting by before inserting into the page). The first function of the action, which is called as part of a typeset action, loops through the math items in the document and call the `addTooltip()` function on each one. The second, called during a convert or rerender action, calls the `addTooltip()` function on the math item that it was passed.

38.2.2 A Render Action to Collapse Complex Subexpressions

This example loads the *a11y/complexity* component and automatically collapses any subexpressions with complexity greater than a given value (20 in this case).

```
MathJax = {
  options: {
    menuOptions: {
      settings: {
        collapsible: true
      }
    },
    renderActions: {
      collapse: [50,
```

(continues on next page)

(continued from previous page)

```
        (doc) => {
          for (const math of doc.math) {
            if (!math.root || math.state > 50) continue;
            math.root.walkTree((node) => {
              if (node.isKind('maction') && node.attributes.get('data-collapsible')) {
                if (node.childNodes[1].attributes.get('data-semantic-complexity') > 20) {
                  node.attributes.set('selection', 1);
                }
              }
            });
            math.state(50);
          }
        },
        '',
        false
      ]
    }
  }
};
```

Here we define an action that occurs after the math is compiled and its complexity values have been computed, but before the math is typeset. The action only operates for typeset actions (not convert or rerender calls), and it walks the MathML tree to find `maction` elements inserted by the *complexity* component that have complexity greater than 20. These get their selections set to 1, so they select the collapsed version instead of the expanded ones. Then the state is set to 50 so that if the item is processed again (either because the typesetting had to be restarted due to an asynchronous action later in the render actions, or by a later typesetting call), this math item will not be checked again.

38.2.3 A Render Action to use Tags for Math Delimiters

This example replaces the usual `find` action with a custom one. Instead of looking for the usual `\(...\)`, `\[...\]`, and `$$...$$` math delimiters, this render action will look for `<tex>...</tex>` and `<dtex>...</dtex>` elements for in-line and display style TeX expressions.

```
MathJax = {
  options: {
    renderActions: {
      find: [10,
        (doc) => {
          for (const node of document.querySelectorAll('tex, dtex')) {
            if (node.childNodes.length !== 1 || node.firstChild.nodeName !== '#text')␣
↪continue;
            const display = node.nodeName.toLowerCase() === 'dtex';
            const math = new doc.options.MathItem(node.textContent, doc.inputJax.tex,␣
↪display);
            const text = document.createTextNode('');
            node.parentNode.replaceChild(text, node);
            math.start = {node: text, delim: '', n: 0};
            math.end = {node: text, delim: '', n: 0};
            doc.math.push(math);
          }
        },
```

(continues on next page)

(continued from previous page)

```
        '',
        false
      ]
    }
  }
};
```

This render action replaces the usual `find` action with one that looks for all the <tex> and <dtex> elements in the document, and checks that they have only one child that is a text element. If so, it creates a new `MathItem` object with the node's text content, using the TeX input jax, and marked as a display expression if the tag was `dtex`. It then makes a new empty text node and replaces the original node with that, then hooks the `MathItem` to the empty text node. Finally, it pushes the new `MathItem` onto the document's math list.

The reason this render action replaces the usual one is because its identifier is `find`. If you want to look for both the original delimiters *and* these tags then you should use a different identifier, like `findTags`, instead.

38.3 MathJax Pre- and Post-Filters

Another means of hooking into MathJax's typesetting pipeline is via pre- and post-filters associated with MathJax's input and output jax. These are prioritized lists of functions that run either before or after the jax processes a `MathItem`, and they can be used to pre-process or post-process MathJax's compiling and typesetting functions. Input and output jax have both pre- and post-filters, and the MathML input jax has an extra set of filters for the parsed MathML as well.

When using *Mathjax Components framework*, you can use the MathJax configuration object to specify input and output jax filters. The `preFilter` and `postFilter` configuration options in the `tex`, `mathml`, `output`, `chtml`, or `svg` blocks allow you to specify arrays of filters (or filters together with their priorities). See the *MathJax Configuration Options* section for details.

When using direct access to the MathJax modules in node applications, to add a pre- or post-filter to an input jax use

`InputJax.preFilters.add`(*fn*, *priority*)

`InputJax.postFilters.add`(*fn*, *priority*)

Arguments

- **`(arg)=>boolean|void`** – The filter function to be called. The `arg` argument is an object with three keys: `math`, `document`, and `data`. The values for these keys are the `MathItem` being processed, the `MathDocument` containing that math item, and jax-specific additional data. If the function returns *false*, the any additional filters are cancelled.
- **`priority`** – The numeric priority of the filter, where lower numbers are executed first. This lets you insert functions anywhere in the filter list.

For the TeX input jax, the `data` item is the `ParseOptions` object for the input jax, which holds configuration data about the TeX input jax.

For the MathML input jax, the pre-filter only runs in the case that the MathML is a serialized MathML string, as it is when converting a MathML string, or when the *forceReparse* option is true. The post-filter's `data` is the root <math> element of the internal MathML tree of the MathML expression. For the MathML input jax, there is also a third filter:

`InputJax.mmlFilters.add`(*fn*, *priority*)

This runs on the MathML DOM tree, either from the document itself, or the one obtained by parsing a serialized MathML string, before the input jax converts the MathML into MathJax's internal format. The `data` in this case is the MathML DOM tree.

The AsciiMath input jax does not currently execute any pre- or post-filters.

For an output jax, the pre- and post-filters can be added via

`OutputJax.preFilters.add`(*fn*, *priority*)

`OutputJax.postFilters.add`(*fn*, *priority*)

with arguments as above. In this case, the `data` is the `mjx-container` node in which the output DOM elements have been placed. This will become the `MathItem.typesetRoot` value, but it has not yet been set when the filters run.

In an application that is using MathJax Components, the input jax can be obtained from `MathJax.startup.document.inputJax.tex` or `MathJax.startup.document.inputJax.mml`, and the output jax from `MathJax.startup.document.outputJax`. For applications using direct access to the MathJax modules, the input and output jax will have been instantiated by hand, so you should already have access to them; if not, then they can be obtained from the `MathDocument` instance returned by `mathjax.document()` by using that in place of `MathJax.startup.document` above.

38.3.1 Allowing Spaces in Numbers

Here is an example of using a TeX input filter to allow numbers to be entered that contain spaces, but where the spaces are removed in the output. That is, `$12 345$` will be parsed as a single number and displayed as `12345`.

```
MathJax = {
  tex: {
    numberPattern: /^(?:[0-9]+(?:(?: +|\{,\})[0-9]+)*(?:\.[0-9]*)?|\.[0-9]+)/,
    postFilters: [
      ({data}) => {
        for (const mn of data.getList('mn')) {
          const textNode = mn.childNodes[0];
          textNode.text = textNode.text.replace(/ /g, '');
        }
      }
    ],
  },
};
```

We set the `numberPattern` option to allow spaces within the number, and then use a post-filter to remove the spaces from the text of any `mn` elements that were produced during the TeX processing.

38.3.2 Converting Full-Width Characters to ASCII Equivalents

This filter converts any character in the Unicode Full-Width character range (U+FF01 – U+FF5F) to their ASCII equivalent versions, leading to better quality output.

```
MathJax = {
  tex: {
    preFilters: [
      ({math}) => {
        math.math = math.math.replace(/[\uFF01-\uFF5E]/g,
          (c) => String.fromCodePoint(c.codePointAt(0) - 0xFF00 + 0x20));
      }
    ]
```

(continues on next page)

(continued from previous page)

```
  }
};
```

This uses a pre-filter to replace characters in the full-width range by an equivalent one in the usual ASCII character range. This will allow numbers to be properly combined by TeX, for example, where the full-width versions would be treated as individual characters.

38.3.3 Converting Unicode Numeric Superscripts to TeX Ones

The following filter converts Unicode pseudo-script numbers (like those in the Superscript and Subscripts block) to actual TeX super- and subscripts.

```
MathJax = {
  //
  // The pseudoscript numbers 0 through 9, and a pattern for plus-or-minus a number
  //
  scripts: '\u2070\u00B9\u00B2\u00B3\u2074\u2075\u2076\u2077\u2078\u2079',
  scriptRE: /([\u207A\u207B])?([\u2070\u00B9\u00B2\u00B3\u2074-\u2079]+)/g,

  tex: {
    preFilters: [
      ({math}) => {
        math.math = math.math.replace(MathJax.config.scriptRE, (match, pm, n) => {
          const N = n.split('').map(c => MathJax.config.scripts.indexOf(c));  // convert␣
↪digits
          pm === '\u207A' && N.unshift('+');     // add plus, if given
          pm === '\u207B' && N.unshift('-');     // add minus, if given
          return '^{' + N.join('') + '}';        // make it an actual power
        });
      }
    ]
  }
};
```

This uses a TeX input jax pre-filter to scan the TeX expression for Unicode superscript numerals, with optional plus or minus signs, and replace them with ASCII numerals inside braces with a `^` to make them actual TeX superscripts.

The filter could be extended to process subscripts in a similar fashion.

38.3.4 Converting SVG Size from Ex to Px units

The SVG output jax sets the <svg> element `width` and `height` attributes using *ex* units, so the SVG will scale to the size of the surrounding font automatically. This filter converts those measurements to *px* units instead.

```
MathJax = {
  svg: {
    postFilters: [
      ({data}) => {
        const fixed = MathJax.startup.document.outputJax.fixed;
        const svg = data.querySelector('svg');
```

(continues on next page)

(continued from previous page)

```
        if (svg?.hasAttribute('viewBox')) {
          const [ , , w, h] = svg.getAttribute('viewBox').split(/ /);
          const em = MathJax.startup.document.outputJax.pxPerEm / 1000;
          svg.setAttribute('width', fixed(w * em) + 'px');
          svg.setAttribute('height', fixed(h * em) + 'px');
        }
      }
    ]
  }
};
```

We use an output jax post-filter to modify the `svg` element's attributes, taking advantage of the output jax's `fixed()` method to obtain a limited number of decimal places. The width and height are determined from the `viewBox` attribute, whose values correspond to `em` units in the SVG output.

38.3.5 An Autobold Filter

This configuration implements a substitute for the v2 *autobold* extension.

```
MathJax = {
  tex: {
    preFilters: [
      ({math}) => {
        const styles = window.getComputedStyle(math.start.node.parentNode);
        if (styles.fontWeight >= 700 && !math.inputData.bolded) {
          math.math = '\\boldsymbol{' + math.math + '}';
          math.inputData.bolded = true;
        }
      }
    ]
  }
};
```

It uses a TeX input jax pre-filter that tests if the parent element of the math string has CSS with `font-weight` of 700 or more (the usual `bold` value), and if so, it wraps the TeX code in `\boldsymbol{...}` to make it bold. Note, however, that if the expression itself includes bold notation, that does not become extra bold, so may not be distinguishable from the rest of the expression.

We track the fact that bolding has been added using the `inputData` object of the `math` object. That way, if the expression needs to be reparsed (e.g., for a `\require` command, or other dynamic data being loaded), we won't add `\boldsymbol` more than once.

38.3.6 Convert Mathvariant to Unicode

This example is more complex, and demonstrates a way to convert the use of the `mathvariant` attribute on the internal MathML token elements to their Unicode equivalents in the Mathematical Alphanumerics block. Because MathML-Core (the version of MathML implemented in most browsers) does not include support for `mathvariant` (except as `mathvariant="normal"` on single-character `mi` elements to prevent the automatic italicization of the character), this may be useful for cases where you want to produce MathML expressions for use with a browser's native MathML-Core support. Using this together with the *native MathML output* example would make that output more effective in browsers that implement MathML-Core.

```
MathJax = {
  startup: {
    ready() {
      //
      //  The numeric ranges for numbers, uppercase alphabet, lowercase alphabet,
      //  uppercase Greek, and lowercase Greek, with optional remapping of some
      //  characters into the (relative) positions used in the Math Alphanumeric block.
      //
      const ranges = [
        [0x30, 0x39],
        [0x41, 0x5A],
        [0x61, 0x7A],
        [0x391, 0x3A9, {0x3F4: 0x3A2, 0x2207: 0x3AA}],
        [0x3B1, 0x3C9, {0x2202: 0x3CA, 0x3F5: 0x3CB, 0x3D1: 0x3CC,
                        0x3F0: 0x3CD, 0x3D5: 0x3CE, 0x3F1: 0x3CF, 0x3D6: 0x3D0}],
      ];
      //
      //  The starting values for numbers, Alpha, alpha, Greek, and greek for the␣
↪variants
      //
      const variants = {
        bold: [0x1D7CE, 0x1D400, 0x1D41A, 0x1D6A8, 0x1D6C2],
        italic: [0, 0x1D434, 0x1D44E, 0x1D6E2, 0x1D6FC, {0x68: 0x210E}],
        'bold-italic': [0, 0x1D468, 0x1D482, 0x1D71C, 0x1D736],
        script: [0, 0x1D49C, 0x1D4B6, 0, 0, {
          0x42: 0x212C, 0x45: 0x2130, 0x46: 0x2131, 0x48: 0x210B, 0x49: 0x2110,
          0x4C: 0x2112, 0x4D: 0x2133, 0x52: 0x211B, 0x65: 0x212F, 0x67: 0x210A,
          0x6F: 0x2134,
        }],
        'bold-script': [0, 0x1D4D0, 0x1D4EA, 0, 0],
        fraktur: [0, 0x1D504, 0x1D51E, 0, 0, {
          0x43: 0x212D, 0x48: 0x210C, 0x49: 0x2111, 0x52: 0x211C, 0x5A: 0x2128,
        }],
        'bold-fraktur': [0, 0x1D56C, 0x1D586, 0, 0],
        'double-struck': [0x1D7D8, 0x1D538, 0x1D552, 0, 0, {
          0x43: 0x2102, 0x48: 0x210D, 0x4E: 0x2115, 0x50: 0x2119, 0x51: 0x211A,
          0x52: 0x211D, 0x5A: 0x2124,
          0x393: 0x213E, 0x3A0: 0x213F, 0x3B3: 0x213D, 0x3C0: 0x213C,
        }],
        'sans-serif': [0x1D7E2, 0x1D5A0, 0x1D5BA, 0, 0],
        'bold-sans-serif': [0x1D7EC, 0x1D5D4, 0x1D5EE, 0x1D756, 0x1D770],
        'sans-serif-italic': [0, 0x1D608, 0x1D622, 0, 0],
        'sans-serif-bold-italic': [0, 0x1D63C, 0x1D656, 0x1D790, 0x1D7AA],
        monospace: [0x1D7F6, 0x1D670, 0x1D68A, 0, 0],
        '-tex-calligraphic': [0, 0x1D49C, 0x1D4B6, 0, 0, {
          0x42: 0x212C, 0x45: 0x2130, 0x46: 0x2131, 0x48: 0x210B, 0x49: 0x2110,
          0x4C: 0x2112, 0x4D: 0x2133, 0x52: 0x211B, 0x65: 0x212F, 0x67: 0x210A,
          0x6F: 0x2134,
        }, '\uFE00'],
        '-tex-bold-calligraphic': [0, 0x1D4D0, 0x1D4EA, 0, 0, {}, '\uFE00'],
        '-tex-mathit': [0, 0x1D434, 0x1D44E, 0x1D6E2, 0x1D6FC, {0x68: 0x210E}],
      };
      //
```

(continues on next page)

(continued from previous page)

```
    // Styles to use for characters that can't be translated.
    //
    const variantStyles = {
      bold: 'font-weight: bold',
      italic: 'font-style: italic',
      'bold-italic': 'font-weight; bold; font-style: italic',
      'script': 'font-family: cursive',
      'bold-script': 'font-family: cursive; font-weight: bold',
      'sans-serif': 'font-family: sans-serif',
      'bold-sans-serif': 'font-family: sans-serif; font-weight: bold',
      'sans-serif-italic': 'font-family: sans-serif; font-style: italic',
      'sans-serif-bold-italic': 'font-family: sans-serif; font-weight: bold; font-
↪style: italic',
      'monospace': 'font-family: monospace',
      '-tex-mathit': 'font-style: italic',
    };
    //
    //  The filter function
    //
    function unicodeVariants(root) {
      //
      //  Walk the MathML tree for token nodes with mathvariant attributes
      //
      root.walkTree((node) => {
        if (!node.isToken || !node.attributes.isSet('mathvariant')) return;
        //
        //  Get the variant and the unicode characters of the element
        //
        const variant =
          node.attributes.get('data-mjx-variant') ?? node.attributes.get('mathvariant
↪');
        const text = [...node.getText()];
        //
        //  Skip the only valid case in MathML-Core and any invalid variants
        //
        if (variant === 'normal' && node.isKind('mi') && text.length === 1) return;
        node.attributes.unset('mathvariant');
        node.attributes.unset('data-mjx-mathvariant');
        if (!Object.hasOwn(variants, variant)) return;
        //
        //  Get the variant data
        //
        const start = variants[variant];
        const remap = start[5] || {};
        const modifier = start[6] || '';
        //
        //  Convert the text of the child nodes
        //
        let converted = true;
        for (const child of node.childNodes) {
          if (child.isKind('text')) {
            converted &= convertText(child, start, remap, modifier);
```

(continues on next page)

(continued from previous page)

```
            }
          }
          //
          // If not all characters were converted, add styles, if possible,
          // but not when it would already be in italics.
          //
          if (!converted &&
              !(['italic', '-tex-mathit'].includes(variant) && text.length === 1 && node.
↪isKind('mi'))) {
            addStyles(node, variant);
          }
        });
      }
      //
      //  Convert the content of a text node
      //
      function convertText(node, start, remap, modifier) {
        //
        //  Get the text
        //
        const text = [...node.getText()]
        //
        //  Loop through the characters in the text
        //
        let converted = 0;
        for (let i = 0; i < text.length; i++) {
          let C = text[i].codePointAt(0);
          //
          //  Check if the character is in one of the ranges
          //
          for (const j of [0, 1, 2, 3, 4]) {
            const [m, M, map = {}] = ranges[j];
            if (!start[j]) continue;
            if (C < m) break;
            //
            //  Set the new character based on the remappings and
            //  starting location for the range
            //
            if (map[C]) {
              text[i] = String.fromCodePoint(map[C] - m + start[j]) + modifier;
              converted++;
              break;
            } else if (remap[C] || C <= M) {
              text[i] = String.fromCodePoint(remap[C] || C - m + start[j]) + modifier;
              converted++;
              break;
            }
          }
        }
        //
        //  Put back the modified text content
        //
```

(continues on next page)

(continued from previous page)

```
        node.setText(text.join(''));
        //
        // Return true if all characters were converted, false otherwise.
        //
        return converted === text.length;
      }
      //
      // Add styles when conversion isn't possible.
      //
      function addStyles(node, variant) {
        let styles = variantStyles[variant];
        if (styles) {
          if (node.attributes.hasExplicit(styles)) {
            styles = node.attributes.get('style') + ' ' + styles;
          }
          node.attributes.set('style', styles);
        }
      }

      //
      //  Add the post-filters to all input jax
      //
      MathJax.startup.defaultReady();
      for (jax of MathJax.startup.document.inputJax) {
        jax.postFilters.add(({data}) => unicodeVariants(data.root || data));
      }
    }
  }
};
```

This example adds a post-filter to each of the input jax that are loaded (so it will work with both the MathML input as well as TeX input). The filter walks the internal MathML tree looking for token elements with `mathvariant` attributes, and then converts the content of the child text nodes of those token nodes to use the proper Unicode values for any alphabetic, numeric, or Greek characters that can be represented using the Mathematical Alphanumeric and Letterlike Symbols blocks. If any characters can't be converted to something in these blocks, we use a `style` attribute, when possible, to simulate the proper output.

The `ranges` variable gives the character ranges that will be converted, the `variants` object gives the data needed to make those ranges to the various Mathematical Alphanumerics characters for the different `mathvariant` values, and the `variantStyles` object to hold the styles that need to be applied for each variant.

The special `-tex-calligraphic` and `-tex-bold-calligraphic` variants are used internally in MathJax to produce the Chancery calligraphic variant (as opposed to the Roundhand script variant), but Unicode does not distinguish between these two, and the result of the `script` and `bold-script` variants is font dependent. The current mechanism to distinguish between these two in Unicode is to use the Unicode variant selector codes U+FE00 and U+FE01. The code here adds U+FE00 for the TeX calligraphic variants. You may wish to add U+FE01 to the script variants to explicitly request the Roundhand versions as well. Note, however, that not all fonts support these variant specifiers, so you may get the same characters in both cases, and which you get will depend on the font. Some browsers may also show unknown character glyphs for these select codes when they don't understand how to process them.

38.4 MathJax Startup and Load Hooks

When you use MathJax Components, there are several methods of synchronizing your code with the startup and loading actions used by MathJax. The first is through the `startup` block of the MathJax configuration object. You can use its *`ready()`* and *`pageReady()`* functions to link into MathJax's initialization sequence. There is also the *`MathJax.startup.promise`* can also be used to queue actions that need to take place after MathJax has performed its initial typesetting operation. These functions and promises are described in detail in the *Performing Actions During Startup* section.

The MathJax *loader* component also provides a number of mechanisms for hooking into its actions. The first is the *loader.ready()* and *loader.failed()* functions in the `loader` block of your MathJax configuration. The first is called when the loader has loaded all the components you have requested and is ready to call the startup component to set up MathJax and do the initial typesetting. You can replace it with your own function, and can call `MathJax.loader.defaultReady()` to perform the usual startup.

The `failed()` function is called if any of the components causes an error, and is passed the `Error` object containing the error (with an extra `package` property indicating the name of the package that caused the error, if known). You can use this method to obtain a full error stacktrace, for example, or to log load errors, or process them in some other way.

In addition to these general hooks, the loader provides ready and failed hooks on an individual-package basis. This is done by making a sub-block of the `loader` configuration block with its name being the name of the component and containing a *`ready()`* and/or `failed()` function. For example,

```
MathJax = {
  loader: {
    '[tex]/color': {
      ready() {console.log('The color extension has been loaded')},
      failed(err) {console.log('The color extension could not be loaded: ' + err.
↪message)},
    }
  }
};
```

would print a message when the color extension is loaded (for example, if it is autoloaded when the `\color` TeX macro is first used, or if loaded explicitly by `\require{color}`), or if it failed to load for some reason.

The package block can also contain a `checkReady()` function, which can be used to perform other asynchronous actions that must be completed when the package loads but before it is considered "ready". For example, if a package being loaded should trigger some additional JSON data to be loaded, the `checkReady()` function can start that loading and return a promise that is resolved when the data is available. In this case, MathJax will wait for that promise before indicating that the package is ready, and that typesetting should continue.

Note

MathJax uses the `checkReady()` mechanism internally (to load the font data when an output jax is loaded, for example), and so you may need to be a bit careful in setting this options for some packages.

CHAPTER

THIRTYNINE

MATHJAX FREQUENTLY ASKED QUESTIONS

39.1 Which license is MathJax distributed under?

MathJax is distributed under the Apache License, Version 2.0.

39.2 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 much of 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 *defer* attribute on the script that loads MathJax, that allows the browser to put off loading and running MathJax until the rest of the page is ready, so that can help speed up your initial page loading as well.

39.3 Mathematics is not rendering properly in IE. How do I fix that?

MathJax support for IE was dropped in MathJax v4. MathJax may still work in IE to some degree, but we no longer test in IE and will not make modifications to accommodate it. MathJax now uses ES6 features that may not not available in IE without using a polyfill or shim library.

MathJax version 3 only supports IE11, so if you are using an earlier version of IE, you will need to update your copy, or use a different browser.

If you are using IE11 with MathJax v3, 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 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*. 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.

39.4 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*.

Some mobile browsers change the page's font size after the page is loaded. Since MathJax tries to match the font size, of that happens after MathJax has run, it can mean that the math fonts are not the right size (usually they are too small). You can prevent the browser from changing the font size by using the following `<meta>` tag in the `<head>` section of your web page:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

This will tell the browser to keep the original scaling.

39.5 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.

39.6 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, while v2 offers an extension for Content MathML (see *the documentation on MathML support*), which has not been converted to version 3 or above.

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`.

39.7 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. The Mathematics StackExchange site has a MathJax-specific tutorial and quick reference that illustrates many of the features of MathJax.

MathML is an XML-based web format for mathematical expressions. MathML3, the latest version, has been an official W3C recommendation since October 2010. MathML4 is being developed, and MathML-Core, a limited version of MathML, is currently a working draft. As of 2023, all the major browsers support MathML-Core natively, though MathML-Core does not include all the features that MathJax uses, so it is not sufficient to render MathJax's MathML output in some cases. 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.

39.8 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.

39.9 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 only a limited amount of its text formatting capabilities. Most 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, or more general conversion tools like PreTeXT or pandoc. Note that TeX conversion tools may not produce results as good as controlling the HTML and CSS source yourself.

39.10 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 *Defining TeX macros* section.

CHAPTER

FORTY

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.

40.1 The MathJax Badges

Thanks to our friends at OER Glue for designing the last two badges.

40.2 The MathJax Logo

40.3 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

40.4 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):

40.4.1 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).

40.4.2 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.

40.4.3 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

FORTYONE

ARTICLES AND PRESENTATIONS

41.1 Articles

- MathJax: The present and the future by Davide Cervone and Volker Sorge, *MathJax*, *2020*
- Adaptable accessibility features for mathematics on the web (W4A 2019) by Davide Cervone and Volker Sorge, *MathJax*, *2019*
- 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*

41.2 Presentations

- Barrierereie 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

FORTYTWO

UPGRADING FROM V3 TO V4

Because version 3 was a complete rewrite of MathJax, the API had major changes from version 2 to version 3, making that upgrade one that could require significant work. That is not the case for upgrading from v3 to v4. The API for v4 is largely the same as v3, with most changes being additions rather than changes.

There are many new features in v4. See the *What's New in MathJax v4.0* section for complete details. There are some breaking changes, which are outlined in the *Breaking Changes in V4* section.

For those who are using MathJax in web pages by just configuring and loading MathJax without calling its API directly, you may not have to make any changes other than changing the URL to load version 4.

If you are calling the MathJax API, then you may need to change to using the promise-based calls like `MathJax.typesetPromise()` rather than the synchronous ones like `MathJax.typeset()`. See the *MathJax v4.0 and Promises* section for details and suggestions for how to proceed.

MathJax v4 now produces ES6 modules rather than the older ES5 CommonJS modules of v3. This means has lead to changes in the MathJax source directories and the build tools that are used to compile and pack MathJax and its extensions. See the sections on *MathJax ES6 Modules*, *Change to ES6 Modules*, and *Changes to the Build Tools* for more information.

There are significant changes to how MathJax handles speech generation in v4. See the *Explorer Technical Details* and *Changes for Speech Generation* sections for more information on these changes and how to address them.

If you have written extensions or other customizations for MathJax, then there are changes that you may need to take into account. See the *Breaking Changes in V4* section for more details about these. There are also many more examples available here for both the web-based and node-based applications of MathJax. See *Examples of MathJax in a Browser* and *Examples of MathJax in Node* for lists of the most important ones.

There are a number of changes to the TeX input jax and its extensions that may affect existing content. For example, the *textmacros* extension is no by default in the *combined components*, which has implications for `\text{}` and other text-mode macros if they contain text that will now be considered as macro references that where ignored in the past. There are also changes to the *mathtools* extension, and to several other extensions. See the *Input Improvements* section for details on this and the other such changes.

The `all-packages` extension has been removed, as have the combined components ending in `-full`. See *Removal of AllPackages* for more on this.

CHAPTER

FORTYTHREE

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.

43.1 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 *The Configuration Variable* 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 MathJax* section for some approaches to these issues.

43.2 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.

43.3 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};
          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.

43.4 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:

43.4.1 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.

43.4.2 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.

43.5 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.

43.6 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 *begingroup* and *mediawiki-texvc* TeX extensions haven't been ported to version 3 yet, but should be in the future.
- 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.

43.7 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).

43.8 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.

43.9 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;
      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: {
```

(continues on next page)

(continued from previous page)

```
    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));
    } 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);
    }
```

(continues on next page)

(continued from previous page)

```
    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

FORTYFOUR

WHAT'S NEW IN MATHJAX

44.1 What's New in MathJax v4.0

MathJax v4 is the culmination of several years of work, and includes significant new features, as well as improvements on existing features. There are some potentially breaking changes, and in order to not have these changes affect existing web sites, this release increments the major version number so that sites using the `mathjax@3` URLs will be protected from these until they update to version 4 explicitly. That is, this new release is an opt-in update.

44.1.1 Extended Font Support

MathJax v4 includes support for a number of new font sets for MathJax, and changes the default font to `mathjax-newcm`, one that is based on the New Computer Modern fonts; it offers support for a much larger range of characters than MathJax's original TeX font set, but is consistent with the look-and-feel of the original MathJax TeX fonts. The new set is slightly lighter, so will not seem so bold and will fit in better on Windows machines, without losing too much on linux, Mac OS, and iOS displays. The original MathJax TeX font set is also available as an option, for those who are unwilling to part with it.

There are 11 fonts available for MathJax v4:

Font Name	Original Source
mathjax-asana	A version of the Asana-Math font
mathjax-bonum	A version of the Gyre Bonum font
mathjax-dejavu	A version of the Gyre DejaVu font
mathjax-fira	A version of the Fira and Fira-Math fonts (a sans-serif font)
mathjax-modern	A version of Latin-Modern
mathjax-newcm	A version of New Computer Modern (the new default font in MathJax)
mathjax-pagella	A version of the Gyre Pagella font
mathjax-schola	A version of the Gyre Schola font
mathjax-stix2	A version of the STIX2 font
mathjax-termes	A version of the Gyre Termes font
mathjax-tex	The original MathJax TeX font

In addition, there is an extension that can be loaded on top of any of these fonts:

Font Name	Original Source
mathjax-euler	A version of the Neo Euler font as an extension

There are also several font extensions used internally to handle some LaTeX packages:

Font Name	TeX extension using this font extension
mathjax-bbm	The *bbm* TeX extension
mathjax-bboldx	The *bboldx* TeX extension
mathjax-dsfont	The *dsfont* TeX extension
mathjax-mhchem	The *mhchem* TeX extension

These are managed by the TeX packages that implement their associated macros, so you won't need to load these yourself. They are listed here only for completeness.

Specifying a Font to Use on the Web

You can specify the font you want to use by setting the `font` option in the new `output` block of your MathJax configuration (where options common to all output renders can be placed). For example,

```
MathJax = {
  output: {
    font: 'mathjax-stix2'
  }
};
```

will select the `mathjax-stix2` font. For in-browser use, this will obtain the font and its data from `cdn.jsdelivr.net` and no other configuration is necessary. It is also possible to configure the path to the fonts using the `fontPath` option of the `output` block of your MathJax configuration. This should be set to a string that indicates where the font can be found; that string should include `%%FONT%%` in any part of the path where the font name needs to appear. For example,

```
MathJax = {
  output: {
    fontPath: '@mathjax/%%FONT%%-font'
  }
};
```

is the default path in node applications.

It is also possible to specify an explicit URL as the font name in the configuration:

```
MathJax = {
  output: {
     font: 'https://cdn.jsdelivr.net/npm/@mathjax/mathjax-stix2-font'
  }
};
```

For those who wish to use the original MathJax font as it appears in version 3, specify the font as `mathjax-tex`.

Specifying a Font in Node Applications

For node applications, first install the font via something like

```
npm install @mathjax/mathjax-stix2-font
```

by adding `-font` to the name of whichever font you want and install that from the `@mathjax` scope. Here we selected the STIX2 font.

For node applications that use the MathJax Components framework, fonts are selected as described for web pages above; MathJax should find the font in your `node_modules/@mathjax` folder automatically.

For applications that use direct access to the MathJax modules, you should import the font you want and pass to the output jax using the `fontData` option when instantiating it. For example

```
import {MathJaxStix2Font} from '@mathjax/mathjax-stix2-font/js/chtml.js';
import {CHTML} from '@mathjax/src/js/output/chtml.js';

const chtml = new CHTML({fontData: MathJaxStix2Font});
```

creates a CommonHTML output jax with the STIX2 fonts. If you will be packaging your application for use on the web, you may need to specify the `fontPath` to point to the location on the web where MathJax should load additional font data, either a CDN, or your own server. For example,

```
import {MathJaxStix2Font} from '@mathjax/mathjax-stix2-font/js/chtml.js';
import {CHTML} from '@mathjax/src/js/output/chtml.js';

const chtml = new CHTML({
  fontData: MathJaxStix2Font,
  fontPath: 'https://cdn.jsdelivr.net/npm/@mathjax/mathjax-stix2-font'
});
```

uses the jsdelivr CDN to deliver the web fonts and dynamic character data for the STIX2 font.

Combined Components with Alternate Fonts

The combined component files, like `tex-chtml.js` and `mml-svg.js`, include the new `mathjax-newcm` font as part of the component so that only one file needs to be downloaded. But if you want to use a different font, you probably don't want to download `mathjax-newcm` first and then the font you actually want to use. Instead, you should use a component ending in `-nofont.js`, for example, `tex-chtml-nofont.js`, so that the initial download is smaller as it doesn't include `mathjax-newcm`. See the section on *MathJax v4.0 and Promises* for more details concerning the proper handling of typesetting with the new fonts.

In addition, the font packages themselves include versions of the `tex-mml-chtml.js` and `tex-mml-svg.js` combined components that include the given font instead of `mathjax-newcm`. See the section on *Combined Components with Fonts* for details on this.

The Font Tools

The tools for building the data needed by MathJax for your own font will be made available after version 4 is officially released. They were used to create these new fonts, but are not yet ready for public use, as they need cleaning up and documentation. But in the future, you will be able to generate an extension to an existing font (for example, to replace the letters and numbers with a different font while leaving all the rest of the characters unchanged), or a completely new font. So look for that functionality in the future.

44.1.2 Line-breaking Support

Version 4 includes the long-awaited implementation of automatic and explicit line breaking of math expressions. The support in v4 is an improvement over that in v2 in a number of ways. In particular, version 4 includes the option of breaking in-line expressions so that long expressions near the end of a line will automatically break and wrap to the next line. This is accomplished by allowing the browser to break the expressions where it needs to (following TeX's rules for what constitutes a valid in-line breakpoint). For display equations, version 4 provides support not only for automatic line breaking, but also for several other options for handling wide equations, including scaling the equation (to fit the container size), and scrolling if it is too wide. The page author can set the default, but there is also a new menu item where the viewer can switch the overflow handling to match their preferences.

Another improvement in v4 concerns line breaking within tables. MathJax now attempts to break cells within tables based on the size of table as a whole, whereas v2 broke cells only if they individually were too wide for the container, and broke them to the container width regardless of the size of the rest of the table. Version 4 also implements line-breaking of <mtext> elements (which are created by \text{} and other text-mode macros) , so long textual material can be broken automatically; this was not available in version 2.

Finally, as part of the line-breaking support, a number of new TeX macros have been made available to control line breaks (to make them more or less desirable, or force or prevent them entirely). Also, support has been added for additional array environment template patterns that can be used to control the width of and automatic line breaking within table cells, as well as insert text before or after every cell in a column, or adjust the spacing between columns. New macros for creating boxes of specific widths in which line breaking will occur are also available, and there are options for controlling justification and indentation of the text. Such boxes are also available in MathML via additional options for the <mpadded> element.

Display breaking

The algorithm used in version 4 for breaking displayed equations is based on the one from version 2, but is not identical to it. Unlike version 2, the results should be nearly identical between the CHTML and SVG output renders, and the code is set up so that the algorithm can be updated or even replaced much easier than in v2. We do have plans for improvements that we hope to make in the future.

The page author can control the method used to handle long expressions via the new displayOverflow output jax configuration option, which can be set to overflow, scale, scroll, truncate, linebreak, or elide, though the latter is not yet implemented. The reader can override the default using the MathJax contextual menu, which has a new item in the *Math Settings* submenu for handling *Wide Expressions*. For MathML input, MathJax version 4 now honors the overflow attribute of the math element, so you can mark a single long expression for line breaking, or for scrolling, for example.

When displayOverflow is set to linebreak, the breaking is controlled by the settings in the linebreaks sub-block of the output (or chtml or svg) block. The default settings are

```
linebreaks: {             // options for when overflow is linebreak
  inline: true,              // true for browser-based breaking of inline equations
  width: '100%',             // a fixed size or a percentage of the container width
  lineleading: .2,           // the default lineleading in em units
  LinebreakVisitor: null,    // The LinebreakVisitor to use
}
```

The last option is used to replace the line-breaking algorithm with a new one, so is a developer option, but the others are author-level settings that control things like how wide the lines are allowed to be, and how much extra space to put between lines.

In-line breaking

In version 4, in-line expressions can be allowed to break automatically by the browser. This is controlled via the inline option of the linebreaks block described above. When true (the default), MathJax will arrange for in-line expressions to be broken into pieces so that the browser can move parts of the equation onto the next line if they would otherwise extend beyond the width of the expression's container. (In version 2, in-line expressions are only broken when the expression by itself would be wider than the container, and in that case, the expression would essentially act like it was inside a element, so it badly disrupts the flow of the paragraph, and could cause misleading wrapping of text around the broken expression).

Note, however, that in order to do this, MathJax must make several separate elements containing math, and for SVG output in particular, several separate top-level <svg> elements. For this reason, node applications that are trying to create single SVG images for the mathematics would want to set linebreaks.inline to false to avoid that.

Finally, because the browser is doing the actual determination of the locations for in-line breaks, these breaks are chosen

purely by how much of the expression can fit at the end of the line before the break. That is, the parameters that mark breakpoints as good or bad (described below) are not taken into effect; however, forced breaks and no-break markers are respected.

New TeX Array Preamble Options

To help support line breaking within cells of wide tables, MathJax v4 includes support for the preamble column declarations defined in the array package for LaTeX. These include the traditional `c`, `l`, and `r` for alignment of the contents of the cell (centered, left, or right), but adds support for `p{width}`, `m{width}`, and `b{width}` for vertical alignment of a fixed-width column in which line-breaking will occur at the given width, as well as `w{align}{width}` and `W{align}{width}`. There is also new support for `>{...}` and `<{...}` for adding content that is put before or after every entry in a column, as well as `@{...}` for replacing the inter-column space with the given content, `!{...}` for replacing inter-column rules, and `*{n}{...}` for repeating a column specification *n* times. Support for `|` and the non-standard `:` are improved so multiple copies of `|` and `:` now produce multiple rules that are close together. Finally, non-standard `P{...}`, `M{...}`, and `B{...}` are defined that produce math-mode versions of their corresponding lower-case counterparts. The `\newcolumntype` macro for declaring new column specifications is also available.

Note that for `p`, `m`, `b`, `w`, `W`, `P`, `M`, and `B` columns, line-breaking will occur to the given column width only if line-breaking is the active overflow setting. Otherwise, wide content will overflow the width, as in actual LaTeX.

Line-breaking macros in TeX

In MathML, `<mo>` and `<mspace>` items can be marked as either good or bad breakpoint locations via the `linebreak="goodbreak"` or `linebreak="badbreak"` attributes, or linebreaks can be prevented via `linebreak="nobreak"` or forced with `linebreak="newline"`. In TeX, these can be controlled via the `\goodbreak`, `\badbreak`, `\nobreak`, and `\break` (or `\\`) macros. These will try to mark the operator that follows (or in some case precedes) the macro using the appropriate `linebreak` attribute. If there is no operator, then one will be introduced into the expression at that location. There is also `\allowbreak` that inserts a breakpoint that can be used if one is needed.

The `\parbox[align]{width}{text}` macro has been added in v4 to provide a line-breaking context of a given width and vertical alignment (`t`, `b`, `c` for top, bottom, center (the default), with `m` allowed as an alias for `c`) for text-mode material. Previous versions of MathJax include `\vcenter{}` for vertical centering, and v4 adds `\vtop{}` and `\vbox{}` for material to be aligned on the top line or bottom line of the contents. In LaTeX, their content is text-mode, but in MathJax, they are in math mode (since MathJax mainly does math-mode, and for backward compatibility with the original `\vcenter{}` implementation). The width of these boxes can be controlled using `\hsize=<dimen>` within the box, so `\vtop{\hsize=10em ...}` would make a box that is 10em wide whose content is line broken and aligned on the baseline of the first line. Finally, the `\makebox[width][align]{text}` macro can also be used to produce a line-breaking text box of a given width and vertical alignment. (This complements the `\mathmakebox[width][align]{math}` macro already in the *mathtools* package.)

While the new array preamble options allow you to control the cell widths in an array, they aren't available for other environments, like `align`. When line-breaking is enabled, you may want to have more control over how long lines of an alignment are broken. You can use `\hbox` or `\mbox` to avoid line breaks, but when you do allow breaks, you may want more control over indenting and alignment in such settings. For this reason, MathJax v4 introduces a non-standard `indentalign` environment that can be used within a cell of a table (or in any line-breaking context) to adjust the indentation amount and the horizontal alignment of any wrapped lines:

```
\begin{indentalign}[first][middle][last]{align}
   (long line of math)
\end{indentalign}
```

where `first`, `middle`, and `last` are optional dimensions that specify how much indentation to use for the first, middle, and last lines (where middle is any but the first or last lines). If only `first` and `middle` are provided, `last` will be the same as `middle`, and if only `first` is given, all three will use the same value. The `align` argument is one to three letters, each being one of `l`, `c`, or `r`, and these represent the alignments for the first, middle, and last lines. So

```
\begin{indentalign}[0em][1em]{l}
   (long line of math)
\end{indentalign}
```

would left align all lines, and indent the second and subsequent lines by 1em, when used in a context where line-breaking is in effect.

A subtle problem occurs within tables when breaks are needed in multiple columns. By default, the baseline of a cell that contains breakpoints is the baseline of the top line of the cell, and since the default row alignment is on the cell's baseline, this means that the rows align on the top lines' baselines. In the situation where the table is from an alignment environment, such as `\begin{align}...\end{align}`, if the first column requires breaks and the second has an equal sign at the beginning of it, then the equal sign would appear to be after the top line of the first column, as shown below:

$$\begin{aligned} a+b+c &= A+B+C+D+E+F \\ +d+e \qquad &+G+H+I+J \end{aligned}$$

which can cause confusing results.

To improve these situations, MathJax v4 introduces additional controls for how cells containing line breaks should be aligned, and sets the defaults for environments like `align` so that the first column aligns on its bottom line while the second is on the top line, producing more effective results:

$$\begin{aligned} a+b+c& \\ +d+e &= A+B+C+D+E+F \\ &\quad +G+H+I+J \end{aligned}$$

In addition, it introduces a new non-standard `\breakAlign` macro that can be used to set the vertical alignment for the various cells, rows, or columns in the alignment. The format is `\breakAlign{type}{align}`, where `type` is one of `c`, `r`, or `t`, indicating whether the alignment is for the single cell in which it occurs, the row in which it occurs, or for the entire table, and `align` is one of `t`, `c`, `m`, `b`, for top, center, middle, or bottom. The difference between `c` and `m` is that `c` always centers the cell regardless of line breaks, while `m` only centers if there are line breaks, and otherwise aligns on the cell baseline. When `type` is `r` or `t`, then `align` can be a sequence of these letters giving the alignments to use in each entry in the row, with the last one being repeated if there are more columns than letters. When `type` is `t` the alignments are applied as row alignments to each row in the table.

For example, `\breakAlign{t}{bt}` could be used at the beginning of an alignment to make the baseline of the bottom row of the first column align with that of the top row of the second column, as in the diagram above.

Linebreaking Control in MathML

The various line-breaking boxes described above are implemented via the MathML <mpadded> element. In order to facility that, MathJax v4 adds two non-standard attributes to the <mpadded> element: `data-overflow` and `data-align`. When `data-overflow="linebreak"` is used, the contents performs line-breaking to the width specified in the element's `width` attribute. (No other value for `data-linebreak` is implemented). The `data-align` attribute value can be `left`, `center` or `right`, to get the contents (line-broken or not) aligned to the left, center, or right of the specified width. You can use an <mstyle> element within the <mpadded> element in order to set the `indentshift`, `indentalign`, and similar attributes (for first and last lines) of the content, or can specify those attributes on the individual <mo> or <mspace> elements within the <mpadded> container.

Control over the alignment of cells with line breaking within an `mtable` can be accomplished in MathML input using the new `data-break-align` attribute on the `mtable`, `mtr`, or `mlabeledtr` elements, or the `data-vertical-align` attribute for `mtd` elements. These can have values of `top`, `center`, `middle`, or `bottom` (repeated and space-separated

for tables and rows). The difference between `center` and `middle` is that `center` always centers the cell regardless of line breaks, while `middle` only centers if there are line breaks, and otherwise aligns on the cell baseline.

The `data-vertical-align` attribute can be used on `msqrt`, `mroot`, and `mrow` elements as well to adjust how they are aligned when they contain line breaks. The default for roots is `bottom`, so that if line-breaks occur within a root, the root will align on its bottom line:

$$\sqrt{\begin{matrix}a+b+c\\+d+e\end{matrix}} = \begin{matrix}a+b\\+c\end{matrix}$$

In TeX there is no direct control over this attribute within roots.

44.1.3 Expression Explorer Updated and On by Default

In version 3 of MathJax, the expression explorer was not active by default, and so those with assistive needs had to turn it on explicitly, and it was loaded dynamically at that time. In its place, the *a11y/assistive-mml* extension was used to generate a hidden MathML expression that could be voiced by those screen readers that can process MathML. This was a stop-gap measure that was both clunky and somewhat fragile, and changes in screen readers often rendered it ineffective (e.g., the current VoiceOver version doesn't read the mathematics and skips it entirely, while Windows screen readers indicate clickable math, but then fail to read it).

Because of these problems, and because the Speech Rule Engine (SRE) that underlies MathJax's assistive support has been substantially rewritten to improve its performance (see the *Technical Details* section below), we are now packaging the expression explorer and SRE as part of all the combined configuration files, while the assistive MathML extension has been removed from them (though it can still be turned on in the `Options` sub-menu of the MathJax contextual menu). The SRE generates speech automatically for the expressions in your page and makes these speech strings available to screen readers using `aria-label` and `aria-braille-label` attributes so that a screen reader can voice the math naturally as part of the page. Moreover, the expressions are focusable so that they can be explored interactively. These features are available by default in version 4, and should provide smoother interaction with screen readers on all platforms.

Exploring Expressions

Using the explorer remains largely the same in v4 as it was in v3: tabbing to an expression activates the explorer, and arrow keys move down into an expression, up to a higher level of an expression, or left and right among neighboring terms. New in version 4 are the following features:

- Clicking on an expression will enter the explorer, highlight the clicked term, and cause a screen reader to speak that term; navigation via arrow keys can continue from that point.
- Double-clicking on an expression enters the explorer at the top level of the expression (just as tabbing to it would), rather than on the clicked term.
- In previous versions, the speech and/or braille subtitles were shown as well, but these have been turned off by default in this release, as they caused confusion for users not aware of the explorer features. They can be re-enabled using the MathJax contextual menu.
- While exploring an expression, pressing `h` will open a dialog box that describes the explorer's features. When an expression is first focused, a message informing the user of this feature is announced after the expression is read, but that can be disabled using the contextual menu.
- In v3, if an expression was explored and tabbing is used to leave the explorer, then when that expression is refocused, the selection would remain where it was when the expression was exited. In this version, tabbing to an expression always starts at the top level of the expression, not the last selected term. The reason for this is so

that if the page is read, whether as a whole or by smaller chunks, the expression will be read in full rather than just the previously selected term. There is a contextual menu item that can be used to select the older behavior, however, if a user wishes to be able to restart the explorer where they left off.

- MathJax now provides optional auto-voicing of expressions together with step-by-step highlighting of expression while a formula is spoken. This feature is primarily aimed at users who do not normally utilize a screen reader, and in particular, as support for dyslexic users. Currently, it has to be switched on explicitly, either in the *Speech* sub-item of the MathJax contextual menu, or using the `voicing` option in the `a11y` sub-block of the `options` configuration block. Speech is generated by providing SSML annotations to the browser's speechSynthesis API. While this makes use of the full range of prosody annotations available in SRE's speech rules, the feature is only available in browsers that come with an implementation of the `speechSynthesis` API and with built-in voices.

The MathJax team tests the explorer with 13 browser/os/screen-reader combinations: Chrome, Firefox, and Safari on MacOS with VoiceOver; Chrome, Firefox, and Edge on Windows with NVDA and JAWS; and Chrome and Firefox on Linux (Ubuntu 24 and 22) with Orca. The new explorer should give a much smoother experience to screen-reader users in all these settings.

Explorer Improvements

In addition to the performance enhancements and usage changes given above, the following are improvements found in v4:

- There is a new Korean and Afrikaans locales for speech output.
- The explorer now handles expressions with line breaks better.
- There are new text heuristics to distinguish genuine text elements from expressions that only use text to enforce font changes to `roman` or `mathvariant=normal`.
- The speech output for tensor expressions has been improved.
- A new Euro Braille output format is available that uses eight-dot Braille to output the original LaTeX for the whole expression, and during expression exploration, the LaTeX used in sub-expressions within the expression.
- Improvements in alphabet generation and symbol translations have reduced the size of the locale files in the distribution.

Technical Details

The generation of speech strings is a resource-intensive process. In the past, that was a source of performance slowdown for pages that contain a lot of mathematics. Version 4 has moved the speech generation into a web-worker, which runs in a separate thread from the main page, so the page will remain responsive to user interaction even while speech strings are being created.

Web-workers are only available in browsers, not in node applications, so MathJax is set up to use the worker-threads library in node, along with some shim code to give it a corresponding interface to the features that MathJax uses from web-workers in the browser. This allows the same code to be used in both browsers and node applications. Currently this requires the use of the LiteDOM in node, but we may provide support in other DOM implementations in the future.

Because node applications that include speech generation now use worker-threads behind the scenes, the node application won't exit until the MathJax worker thread is shut down, so applications may need to tell MathJax that they are finished. This is done by calling the new `MathJax.done()` method in applications that use MathJax components, or by calling the MathDocument's `done()` method for those that use direct calls to MathJax's modules.

Aside from moving the speech generation into a web-worker or node worker-thread, the speech-generation has been improved by adding more sophisticated semantic analysis for complex user defined structures, including improved disambiguation of function applications, ellipses, and unusual large operators. Additional locales supported are Afrikaans and Korean as well as 8-dot Euro Braille output generated from original LaTeX input formulas.

44.1.4 Support for HTML in MathML and TeX

HTML in MathML

The HTML5 specification allows for mixing HTML nodes inside MathML token nodes, and it is a long-standing request for MathJax to implement that as well. Version 4 finally does so. You can now use HTML nodes as children of token nodes, such as `<mtext>`. Thus

```
<mtext>a button <input type="button" value="Push Me"> to press</mtext>
```

is allowed, and would produce an `<mtext>` element containing a button surrounded by some plain text.

Because the HTML is not currently sanitized (something that could be added to the *ui/safe* extension), allowing HTML in token elements would be a security issue for sites that allow user-entered MathML. For this reason, the MathML input jax has a new option `allowHtmlInTokenNodes` to control whether to allow it, and it is `false` by default, so you have to opt into this new feature if you want to use it on your site.

HTML in TeX

HTML is now allowed in TeX and LaTeX input as well. This is handled through the special `<tex-html>` node, which (unlike most HTML nodes) can be included within the math delimiters. So, for example

```
$$3 + <tex-html><input type="text" id="answer" size="10"></tex-html> = 10$$
```

would present an expression with an input box where a student could fill in an answer. This feature is implemented via the *texhtml* extension package for the TeX input jax, so you would use a configuration like

```
MathJax = {
  loader: {load: ['[tex]/texhtml']},
  tex: {
      allowTexHTML: true,
      packages: {'[+]': ['texhtml']}
  }
};
```

to load it, add it to the packages to use, and enable it.

In its normal use case in the browser, the HTML will come from the DOM already, and so MathJax doesn't include HTML sanitization in this extension. Because of this, however, the *texhtml* extension does represent a security risk on sites that allow user content, if they don't sanitize the user input themselves. For this reason, there is an `allowTexHTML` option for the TeX input jax that must be enabled in order for the `<tex-html>` elements to be used. Note that `\require{}` is configured *not* to load the *texhtml* package, so unless you explicitly load it yourself, there should be no security issue.

Specifying the size of the HTML

In a browser, MathJax can measure the size of the HTML so that it can provide the proper amount of space for it within the equation, but in node applications, that is not possible, so MathJax provides a method for you to specify the size of the HTML explicitly. To specify the dimensions, add `data-mjx-hdw="H D W"` to the top-level HTML element inside the MathML token element, where `H`, `D`, and `W` are the height, depth, and width of the HTML. They can be in any units, but `em` units will work best.

How this attribute is used is handled via a new option to the output jax, `htmlHDW`, which can be set to `'auto'` (the default), `'ignore'`, `'use'`, or `'force'`. When set to `ignore`, the `data-mjx-hdw` attribute is ignored, and MathJax will try to measure the size of the HTML directly. This works well in the browser, but not in the liteDOM, jsdom, linkedom, or other non-browser adaptors. The `force` option means that MathJax will use the `data-mjx-hdw` values and will surround the HTML with additional nodes that force the HTML to have the given dimensions. This would

make the browser and node both have the same representation, not relying on the browser measurements. The value `use` means that MathJax will assume the `data-mjx-hdw` values are correct and will use them without forcing the HTML to have the given dimensions. Finally, `auto` means that MathJax will determine which to use; this will be `ignore` when in the browser and `force` when in node applications.

Having accurate values for the `data-mjx-hdw` attribute is crucial to the quality of the output. To that end, the following HTML file computes the needed values. These values depend on the surrounding font, so there is a place to enter that, as well. You can also specify the font in the MathML token element that holds the HTML, as in `<mtext fontfamily="Arial"><div>...</div></mtext>` or `<mo style="font-family: arial"><div>...</div></mo>`. Otherwise, MathJax will use the surrounding font. The page below gives you a place to enter the HTML you want to measure and the font to use. Press the `Compute HDW` button and the HTML is shown below together with modified HTML source that includes the needed `data-mjx-hdw` attribute. You can copy that and replace the original HTML will it.

```
<!DOCTYPE html>
<html>
<head>
<title>Compute HDW values for HTML in Token nodes</title>
<script>
function GetHDW() {
  const html = document.querySelector('mjx-html');
  const content = html.getBoundingClientRect();
  const baseline = document.querySelector('mjx-baseline').getBoundingClientRect();
  const em = parseFloat(window.getComputedStyle(html).fontSize);
  const h = baseline.top - content.top;
  const d = content.bottom - baseline.top;
  const w = content.right - content.left;
  return [h, d, w].map(x => (x / em).toFixed(3).replace(/\.?0+$/, '') + 'em').join(' ');
}
function ShowHDW() {
  const html = document.querySelector('#html').value;
  const content = document.querySelector('mjx-html');
  content.style.fontFamily = document.querySelector('#family').value;
  content.innerHTML = html
  const output = document.querySelector('#output');
  content.firstElementChild.setAttribute('data-mjx-hdw', GetHDW());
  output.innerHTML = content.innerHTML.replace(/</g, '&lt;');
}
</script>
<style>
mjx-measure {
  display: inline-block;
  border-left: 2px solid red;
  border-right: 2px solid red;
}
mjx-baseline {
  display: inline-block;
  height: 0;
  width: 0;
}
mjx-html {
  display: inline-block;
```

(continues on next page)

(continued from previous page)

```
}
mjx-line {
  display: inline-block;
  height: 0;
  width: 1em;
  border-top: 1px solid blue;
}
#input {
  display: inline-block;
}
#input textarea {
  margin-bottom: 3px;
}
#input input[type="button"] {
  float: right;
}
</style>
</head>
<body>

<h1>Compute HDW values for HTML in Token nodes</h1>

<p id="input">
<textarea id="html" cols="80" rows="10">
<div>HTML</div>
</textarea><br>
Font family: <input type="text" id="family">
<input type="button" value="Compute HDW" onclick="ShowHDW()">
</p>
<h2>The HTML:</h2>
<p>
<mjx-line></mjx-line><mjx-measure><mjx-baseline></mjx-baseline><mjx-html>
 
</mjx-html></mjx-measure><mjx-line></mjx-line>
</p>
<h2>The HTML with the HDW attribute:</h2>
<p id="output">
</p>

</body>
</html>
```

Of course, you can use the code above as a basis for automating the process using something like puppeteer, if you wish.

44.1.5 MathJax ES6 Modules

When MathJax was first released, the current version of JavaScript was ES5, so when the code base was moved to Typescript for v3, it was down-compiled to produce ES5 code. Modern browsers support ES6, which include many

new features, such as true object `class` creation and inheritance, proper `import` and `export` commands, `Set` and `Map` objects, promises, iterators, and many other features that make JavaScript programs faster and more reliable.

Along with new language features, ES6 introduced a new module structure that affects how individual javascript files obtain values from other files, and how they make their own definitions available to others. ES6 modules (which we will refer to as *MJS* modules) use the new `import` and `export` commands to do this, while the older CommonJS module format (which we will call *CJS*) used `require()` and the `module.exports` object to perform those functions.

MathJax v3 uses CommonJS modules with ES5 code (though this was not quite pure ES5, since one of its dependencies was actually ES6), but modern JavaScript applications are moving more and more to MJS format. Beginning with version 4, MathJax offers both MJS and CJS versions, with the MJS version being ES6, but the CJS version remaining ES5, as in past versions. The webpacked components for use in web pages are now based on the MJS versions.

Implications for MathJax in Web Pages

The webpacked MJS files are smaller than the earlier webpacked CJS files, so that should mean faster download and compile times, and the ES6 code is more efficient, so should run faster. But since this version is no longer ES5, the `es5` directory that was part of the URLs for accessing MathJax from a CDN is no longer correct. The details of how the directories have been adjusted are given in the next section, but for use on the web, the only important difference introduced by the change to ES6 is that you simply remove the `/es5` from the url. For example, you would use

```
https://cdn.jsdelivr.net/npm/mathjax@4/tex-mml-chtml.js
```

to load the `tex-mml-chtml.js` combined component.

Support for IE11 has been dropped in v4, as it does not support enough of the ES6 standards. (It is possible to webpack the CJS versions, so you can build your own ES5 version if that is necessary for you. This is described at the end of this section below.) In version 3, we recommended a link to `polyfill.io` in order to support IE11. This should now be removed since version 4 will not work with IE11 even with the polyfill, and especially since the `polyfill.io` has been bought by a Chinese entity that has used it to insert malignant code into web sites that use it.

If your only usage is in a web browser, you can skip the *next section*.

New Directory Structure

MathJax's new dual distribution of both MJS and CJS modules requires a new directory structure in the mathjax/MathJax-src repository and its associated `@mathjax/src` npm package in order to accommodate both versions. In the past, the compiled JavaScript code was found in the `js` directory, and the webpacked components were in the `es5` directory. Now that there are both CommonJS and ES-module versions of the compiled code, these are stored in the `cjs` and `mjs` directories, respectively.

The webpacked components are now based on the new `mjs` files, hence they are ES6 files, and so the `es5` directory has been removed, with the components now being placed in the new generically named `bundle` directory. That way, if there is a move to ES7 or higher, the directory name doesn't need to change again. The mathjax/MathJax repository and associated `mathjax` package have also eliminated the `es5` directory, and the combined components and component directories are now at the top level of the repository. That means you can access them without the need for `/es5` in the URL that was needed in v3. (See the *Accessing MathJax v4* section for more information on how to access v4.0 in a browser.)

Existing node applications that use MathJax (e.g., the v3 examples in the MathJax-demos-node repository) may have code that refers to the `mathjax-full/js`, `mathjax-full/es5`, or `mathjax/es5` directories that no longer exist. To accommodate these, v4 includes an `exports` section in its `package.json` file that maps these references to the proper new locations. In particular, references to `@mathjax/src/es5` are routed to `@mathjax/src/bundle` automatically, and similarly, `mathjax/es5` is routed to `mathjax`. For the `@mathjax/src/js` directory, references will be routed to `@mathjax/src/mjs` or `@mathjax/src/cjs` depending on whether the reference is from an `import` statement or a `require()` call. That means that ES modules (using `import`) will get the `mjs` versions, while CommonJS modules (using `require()`) will get the `cjs` ones.

The main `package.json` file now includes a `"type": "module"` line so that the `.js` files are considered to be MJS files automatically. The `cjs` directory (and other directories that need to be marked as CJS files) contain separate `package.json` files that set the type to `commonjs` so that the `.js` files they contain will be treated as CJS files.

Similarly, the font directories in the font packages have `mjs` and `cjs` directories, with `exports` sections in their `package.json` files to route `import` to the former and `require()` to the latter.

Changes to `components/src`

The files that are used to create the webpacked component files are ES6 modules, since they use `import` and `export`, and in previous versions of MathJax, you needed to use `node -r esm` to be able to `require()` these in your own programs as you can't use `require()` to load ES modules directly. Although you could load the webpacked versions of the component files in either MJS or CJS applications, the fact that the source component files are MJS modules made it difficult to use the source versions of the components in CJS applications.

Now that MathJax provides both MJS and CJS versions, the source component files are made available in both forms as well. Originally, the component files were found in `components/src`; with this beta version, those are now in `components/mjs`, since they are ES modules.

Prior to version 4, MathJax used Babel to convert these to ES5 during the webpack process, but since the webpacked versions are now ES6, that is no longer necessary, and Babel is no longer needed as a dependency for MathJax. Instead, for those who wish to use the components from source in a CommonJS node application, we use Typescript to down-compile the `components/mjs` files into the `components/cjs` directory as CJS modules of ES5 code.

In previous versions, `require('mathjax-full')` would load `components/src/node-main/node-main.js`, which would load the components from source rather than the webpacked versions. With the `mathjax` package, which only includes the webpacked versions, `require('mathjax')` would get `es5/node-main.js`, the webpacked version. In version 4, the two have been standardized so that they both load the webpacked version. When used with `require()`, you will get `bundle/node-main.cjs`, while `import` will load `bundle/node-main.mjs`. This is accomplished via the `exports` section of the `package.json` file. Of course, you use `@mathjax/src` in place of `mathjax-full`.

In order to get the source versions in `@mathjax/src`, use

```
const MathJax = require('@mathjax/src/source');
```

or

```
import MathJax  from '@mathjax/src/source';
```

and then call `MathJax.init(...)`. These load `components/mjs/node-main/node-main.mjs` or `components/cjs/node-main/node-main.cjs`, respectively.

For those who have been using `components/src` to load individual components from source, we map `components/js` to `components/mjs` when included via an `import` command, and to `components/cjs` when included via `require()`, so you can use `components/js` to get the correct files in either case. For backward compatibility, `components/src` has been mapped the same way.

The end result is that you should always get an appropriate version for your situation, whether you are importing MathJax into an MJS application or requiring it into a CJS one.

More MJS/CJS Issues

Since MathJax now needs to produce javascript files in two different formats, we use different typescript configuration files for the different setups. These are stored in the `tsconfig` directory. The MJS files are produced using `tsconfig/mjs.json` and the CJS one use `tsconfig/cjs.json`. Both of these call in `tsconfig/common.json` to set the parameters that are common to both, and then specify the `target` and `module` values to be correct for the desired JavaScript version and module format. The main `tsconfig.json` file simply calls in `tsconfig/mjs.def` and is there as a convenience for tools that expect a `tsconfig.json` file in the main directory.

In order to support both MJS and CJS versions, MathJax's dependencies also must provide both versions. The `speech-rule-engine`, `mj-context-menu`, and `mhchemparser` packages all now include both module formats using dual directories similar to MathJax itself. This means that the imports used by MathJax for these packages need to change depending on which module version is being created. In order to accomplish this, the references to those packages are handled using pseudo-package references that are remapped to the correct locations via the `tsconfig.json` and `package.json` files.

To this end, MathJax now uses the `#sre`, `#menu`, and `#mhchem` pseudo-package names to refer to these packages. The main `package.json` file uses the `imports` section to map these to the actual package directories that contain their MJS JavaScript files. E.g., `#mhchem/*` is mapped to `mhchemparser/esm/*`, to obtain the ES module versions of the parser. Conversely, the `package.json` file that is placed in the `cjs` directory maps `#mhchem/*` to `mhchemparser/js/*` to obtain the CommonJS versions. Similar mappings are done for the other two packages.

In addition to the mappings in the `package.json` file to let node (and webpack) know which directory to use, Typescript must also be told where to look for the `.d.ts` files for these packages. This is accomplished through the `tsconfig` json files via the `paths` array.

MathJax takes its default font from one of the MathJax font packages, and that also has separate MJS and CJS directories, so the MathJax code uses a pseudo-package, `#default-font`, to link to the proper `mjs` or `cjs` directory in the font package. This also provides a means of specifying what the default font is (`mathjax-newcm` by default), as changing the mappings in the `package.json` files and the `tsconfig` files would change the default font.

For the most part, MathJax's typescript source code can be used to produce either ES modules or CommonJS modules without alteration. But there are a few differences between MJS and CJS code that do need to be taken into account. For example, CommonJS code provides `__dirname` for the location of the file being compiled, but this is not available in MJS modules; meanwhile, MJS files must use `new URL(import.meta.url).pathname` to get that data, and `import` is not available in CJS modules. That means there is no common method that can be used for this in both cases, and so MathJax has some module-specific files to handle the few instances where module-specific code is needed.

To accommodate this, we introduce additional pseudo-package names that can be used to select between MJS and CJS files that export the needed data using the module-specific mechanisms. The `#root` and `#mml3` pseudo-package names are used for these two situations in the Typescript code, and `#js` and `#source` are used in the `components/mjs` definitions to link to the `mjs` or `cjs` JavaScript code and to module-specific code for obtaining the directory name. These are mapped to the proper locations in the `package.json` files and the `tsconfig` files. The module-specific code that these link to are stored in `mjs` and `cjs` directories where they are needed, so that `#root/root.js` gets mapped to `ts/components/mjs/root.js` when MJS files are being produced, but to `ts/components/cjs/root.js` for CJS files, and similarly for `#root/sre-root.js`. The `tsconfig` files exclude the directories for the other format so that they are not compiled when not needed.

Finally, since modern browsers can import MJS files, it is possible to load the MathJax files into a browser directly via a `<script type="module">` tag. To do so, however, you need to include a `<script type="importmap">` tag that tells the browser how to find the pseudo-packages described above. Something like

```
<script type="importmap">
{
  "imports": {
    "#js/": "./node_modules/@mathjax/src/mjs/",
    "#source/source.cjs": "./node_modules/@mathjax/src/components/mjs/source-lab.js",
    "#root/": "./node_modules/@mathjax/src/mjs/components/mjs/",
    "#mml3/": "./node_modules/@mathjax/src/mjs/input/mathml/mml3/mjs/",
    "#default-font/": "./node_modules/mathjax-modern-font/mjs/",
    "#sre/": "./node_modules/speech-rule-engine/js/",
    "#menu/": "./node_modules/mj-context-menu/js/",
    "#mhchem/": "./node_modules/mhchemparser/esm/",
    "@mathjax/src/components/js/a11y/util.js": "./node_modules/@mathjax/src/components/
↪mjs/a11y/util-lab.js",
```

(continues on next page)

(continued from previous page)

```
    "@mathjax/src/components/js/": "./node_modules/@mathjax/mjs/components/mjs/",
    "@mathjax/src/js/": "./node_modules/@mathjax/src/mjs/",
    "@mathjax/src/": "./node_modules/@mathjax/src/"
  }
}
</script>
```

should do the trick. Then you can set up your MathJax configuration in a file `mathjax-config.js` as in

```
import {source} from '@mathjax/src/components/js/source.js';

window.MathJax = {
  loader: {
    load: ['input/tex', 'output/chtml'],
    source: source
  }
};
```

and then use

```
<script type="module">
  import './mathjax-config.js';
  import '@mathjax/src/components/js/startup/startup.js';
</script>
```

to load MathJax components via their source code rather than webpacked files.

This is useful for testing changes to MathJax, but should not be used in production, as the number of files that will be loaded can be quite large, and each file will need to be retrieved separately, making for unneeded network overhead.

Component JSON files

In past versions of MathJax, the `components/src` directory contained files that control the production of the webpacked component files in the `es5` directory. Each component had a subdirectory that contained files that told the MathJax build tools how to construct the component. These included at least one `.js` file that imported the needed MathJax modules and did any setup needed for the component, along with one or more of `build.json`, `copy.json`, and `webpack.config.js` that contain the data needed for the build tools to process the component.

In version 4, these three `.json` files have been combined into a single `config.json` file that contains sections for each of the three original ones. The `build` property of `config.json` contains the data that used to be in `build.json`, the `copy` property holds what was in `copy.json`, and the `webpack` property holds the data needed to pack the component.

The `webpack` data primarily consists of the data that used to be passed to the `PACKAGE()` function in `webpack.config.js`, but as named properties, and only those that differ from the defaults need to be included (unlike the calls to `PACKAGE()` where all arguments were needed). The defaults are set up so that most properties don't need to be specified, just the name of the component and the `libs` array, in most cases. There are some additional properties that control whether the default font should be included in the packed file, or whether a function from an external file should be called to modify the default webpack configuration after it has been constructed. See the `config.json` files in the various `components/mjs` subdirectories for examples.

Building MJS and CJS versions

There are a large number of new package scripts used for building the files used by MathJax. The main changes are that there are separate commands for building the MJS and CJS files, along with new commands to make everything all in one step, and for making single components individually.

The `pnpm -s compile` and `pnpm -s make-components` scripts perform these steps for the MJS versions, and there is a new

```
pnpm -s build
```

command that does both of these at once. The command

```
pnpm -s build-all
```

not only compiles and packs the MJS versions, but also compiles the CJS versions.

Additional commands and details are given in the section on *Changes to the Build Tools*.

Reproducing the Old ES5 Webpack Files

The webpacked components in the `bundle` directory are based on the MJS files, which are ES6 JavaScript files. In contrast, the earlier versions of MathJax had CJS versions of ES5 code. If you need to support an ES5 environment, it is possible to build that using

```
npm run -s compile-cjs
npm run -s make-cjs-components
```

which will create a `bundle-cjs` directory that contains ES5 versions of the webpacked components, comparable to the old `es5` directory.

The `tsconfig.json` file has the settings for the MJS versions of the JavaScript files, and if you use an editor like `emacs`, the Typescript editing mode automatically compiles the `.ts` files based on `tsconfig.json` into the `mjs` directory. If you need to make modifications to the typescript files and your application links to the CJS versions of the compiled files, it may be convenient to switch the `tsconfig.json` file to produce the CJS versions instead. To do that, use

```
npm run -s use-cjs
```

which will point the `tsconfig.json` file to the parameters for compiling into the `cjs` directory instead. When you are done,

```
npm run -s use-mjs
```

will set things back to the original arrangement.

44.1.6 Accessing MathJax v4

MathJax version 4.0 can be accessed via CDN as

```
https://cdn.jsdelivr.net/npm/mathjax@4/tex-mml-chtml.js
```

using any one of the *combined components*:

- `tex-html.js`
- `tex-svg.js`

- tex-mml-chtml.js
- tex-mml-svg.js
- mml-html.js
- mml-svg.js

Note that the /es5 directory is no longer needed in the URL. See the *MathJax ES6 Modules* section for more details on this change.

Each of the combined component files includes the mathjax-newcm font, but also comes in a version ending in -nofont.js (e.g., tex-mml-chtml-nofont.js) that does not include it, where you are expected to specify the font using the output.font configuration option. This saves your readers from having to download the mathjax-newcm font that is not going to be used. See the section *Extended Font Support* for details about the available fonts and how to access them.

The other combined configurations from v3 have been removed, as they were either redundant (now that the explorer is already included in all combined components), or where the ones that included tex-full. The latter have been removed because you will need to use the promise-based calls anyway, and that will handle the autoloading of extensions as well, and since the all-packages extension doesn't include the newer packages, so isn't really "all" packages anyway, it has also been removed. See the section on *Removal of AllPackages* for more information, and for an example of how to implement an all-packages work-around.

MathJax Scoped NPM Packages

With version 4, MathJax has moved to scoped packages for the source and font npm packages. The mathjax-full package is now @mathjax/src, and the font packages are @mathjax/mathjax-stix2-font, @mathjax/mathjax-fira-font, and so on. Future extensions and other packages will be in the @mathjax scope as well. The only exception is that the mathjax package remains un-scoped. Since the use of MathJax in browsers is primarily through the mathjax package, that means the URL for loading MathJax will remain the same, with only the version number needing to be changed and the /es5 directory removed. That is, you can use

```
https://cdn.jsdelivr.net/npm/mathjax@4/tex-mml-chtml.js
```

to load the latest version 4 of MathJax. To obtain the source version to include in a node project, you would use

```
pnpm install @mathjax/src@4
pnpm install @mathjax/mathjax-newcm-font
```

and if you want a different font, a command like

```
pnpm install @mathjax/mathjax-stix2-font
```

to get the latest *mathjax-stx2* font package. The other fonts can be obtained similarly.

In your code, you will need to change any references to the mathjax-full package to @mathjax/src, and any references to mathjax-[fontname]-font to @mathjax/mathjax-[fontname]-font.

Combined Components with Fonts

In a browser, when you specify the font option in the output (or chtml or svg) block of the MathJax configuration, MathJax should access the fonts from cdn.jsdelivr.net automatically. But if you load a *combined component* like tex-mml-chtml.js, it will include the mathjax-newcm font data, even if you are loading another font.

You can overcome this by loading the -nofont version of the combined configuration, but there is also another approach. The font packages include combined configuration files that are equivalent to tex-mml-chtml.js and tex-mml-svg.js, but that include that package's font rather than mathjax-newcm.

For example, the `mathjax-stix2-font` package includes `tex-mml-chtml-mathjax-stix2.js` and `tex-mml-svg-mathjax-stix2.js`, so you can use

```
https://cdn.jsdelivr.net/npm/@mathjax/mathjax-stix2-font/tex-mml-chtml-mathjax-stix2.js
```

in order to get a single-file MathJax component that includes the `mathjax-stix2` font rather than `mathjax-newcm`.

In particular, you can get the equivalent of the `tex-mml-html.js` file with the original MathJax TeX font all in one file using

```
https://cdn.jsdelivr.net/npm/@mathjax/mathjax-tex-font/tex-mml-chtml-mathjax-tex.js
```

This font does not have dynamic ranges (all the font data is in one file), so it should operate much the same as MathJax v3 in that respect.

Similarly, you could use the SVG versions to get MathJax with a specific font with SVG output.

44.1.7 Input Improvements

MathJax v4 includes several new features for the TeX input jax, including updated Unicode positions for some macros, new macros for characters that are now available in the fonts, updates to some TeX extensions packages, new configuration parameters, the ability to embed HTML within a TeX expression, and new macros for controlling vertical alignment and line breaks.

The macros for line breaking and alignment are discussed in the *Line-breaking macros in TeX* section, while embedding HTML in TeX expressions is described in the *HTML in TeX* section. Support for additional `array` environment preamble column types is discussed in the section on *New TeX Array Preamble Options*. The other additions and changes are described below.

Textmacros Enabled by Default

In previous versions of MathJax, the macros in text-mode material (in `\text{}` and similar macros) were not processed unless the *textmacros* extension was loaded and enabled. This version now includes the *textmacros* extension in all the combined components that contain the TeX input jax (i.e., all the ones starting with `tex-`). This means that macros inside text-mode will be processed. Because only a limited number of text-mode-macros are defined, this can lead to errors in cases where the literal macro name would be displayed in the past. For example, prior to v4,

```
\text{The \item macro is used in lists}
```

would produce the literal string `The \item macro is used in lists`, but in v4 and above, it will lead to an error message about `\item` being undefined. So this may be a breaking change in some pages that take advantage of the old behavior.

You can disable the *textmacros* extension in combined components that include it by merging

```
MathJax = {
  tex: {
    packages: {'[-]': ['textmacros']}
  }
};
```

into your MathJax configuration.

New and Updated Macros

The Unicode characters produced by \vdash, \models, and \backslash have been adjusted to produce better results. The \iddots, \dddot, \ddddot, \oiint, \oiiint, \ointop, and \AA macros have been added, as have the displaymath, math, and darray environments.

The non-standard \bbFont and \scr macros have been removed, and the \frak macro has been made compatible with its usual LaTeX version.

The \underline, \llap, \rlap, \phantom, \vphantom, \hphantom, \smash, \mmlToken macros have been added to the textmacros package for use in text mode.

The \char macro is now available for inserting characters by their Unicode character positions. It produces an internal mn, mi, mo, or mtext element depending on the character specified. E.g., \char"61 produces <mi>a</mi> internally.

A new non-standard macro \U is now available for inserting a Unicode character into the TeX input string to be processed as though it had been in the input stream originally. It takes on argument, which is the Unicode code point in hexadecimal notation. For example, \U{229E} would produce the character U+229E, a plus sign in a square. Note in particular that these macros can be used in the second argument to \mmlToken, as in \mmlToken{mi}{\U{213C}}.

A new non-standard macro \breakAlign has been added to control the vertical alignment of blocks that contain line breaks. This and several other new line breaking macros are discussed in the previous section on *Line-breaking macros in TeX* above.

New TeX Packages

The *units* package has been added, which makes the \units, \unitfrac, and \nicefrac macros available, along with new tex.units.loose and tex.units.ugly configuration options. Both are boolean values, and the first controls how large the space is before units (true is a large space, false a smaller one), while the second determines whether \nicefrac produces bevelled fractions (false) or stacked fractions (true).

MathJax v4 includes three new TeX packages that provide alternative double-struck (i.e., blackboard bold) character sets: *dsfont*, *bbm*, and *bboldx*. New font extensions are now available for these packages, and these are loaded automatically when the TeX package is loaded. These extensions work in cunjunction with any of the fonts available in v4.

The *dsfont* package defines a macro \mathds that provides access to its double-struck characters. There is a configuration option that controls whether the sans-serif version of these fonts is used, or the roman versions:

```
MathJax = {
  tex: {
    dsfont: {
      sans: true   // default is false
    }
  }
}
```

The *bbm* package defines macros \mathbbm, \mathmmbss, and \mathbbmtt to generate its double-struck characters, as well as a \mathversion macro that can be used to select the version of the double-struck fonts to use (this is a global setting). Here, \mathversion{bold} selects the bold versions of the double-struck characters, while any argument other than bold will select the normal versions of the fonts.

The *bboldx* package redefines \mathbb to use the bboldx double-struck characters, and adds \mathbfbb to access their bold-face versions, plus \imathbb, \jmathbb, \imathbfbb, and \jmathbfbb for dotless i and j characters in these fonts. In addition, there are macros for upper- and lower-case Greek letters, e.g., \bbGamma, \bfbbsigma, etc., and text-based versions of these for use in \text{}, e.g., \txtbbGamma. The bold delimiters \bbLparen, \bbRparen, \bbLbrack, \bbRbrack, \bbLangle, \bbRangle, and the bfbb versions of these, are defined as well.

One of the last remaining extensions from version 2 that was not ported to v3 is now available in v4: the *begingroup* extension that allows you to create temporary macro definitions. The v4 version of *begingroup* defines two new non-standard macros: `\begingroupReset` and `\begingroupSandbox`. The first one ends any open `\begingroup` macros, removing any of their temporary macros or environments. The second can be used to isolate the definitions in one section of the page from those in another, so that sites like StackExchange can use this between user posts to make sure that one user doesn't redefine things to mess up another user. The `\begingroupSandbox` macro can't be redefined, and its action is essentially to do `\begingroupReset\begingroup`. This removes any previous user definitions and makes a new group for the next user's definition. It also directs any global definitions to this new group so that a user can create global macros in their own sandbox, but they are removed at the next `\begingroupSandbox` call. Any macros or environments created before the first `\begingroupSandbox` call are shared definitions that are available in every sandbox. Once `\begingroupSandbox` is performed, however, there is no going back; no new shared definitions can be made.

Note that macros loaded by `\require{}` or by the *autoload* extension are not managed by the *begingroup* extension, so are global and remain in effect even after an `\endgroup` or `\begingroupSandbox` call.

Updates to `mathtools`

The *mathtools* extension has been updated to reflect the changes to the actual LaTeX package that were made in 2022 and 2024. In particular, there are some breaking changes to `\coloneq` and three other colon macros, several new colon-like commands, and several new extensible arrow macros, as described below.

The `\coloneq`, `\Coloneq`, `\eqcolon` and `\Eqcolon` macros now use the 2022 and later definitions (they use = rather than -, so `\coloneq` produces `:=` not `:-` as in the past). A new `legacycolonsymbols` option controls which set to use (just as in actual *mathtools*). This can be set in the `mathools` section of the `tex` block of your MathJax configuration, or via the `\mathtoolsset` macro. The new colon macros `\approxcolon`, `\Approxcolon`, `\simcolon`, `\Simcolon`, `\colondash`, `\Colondash`, `\dashcolon`, and `\Dashcolon` are now defined, and are available regardless of the setting of `legacycolonsymbols`.

The new extensible arrow macros are `\xlongrightarrow`, `\xlongleftarrow`, `\xLongrightarrow`, and `\xLongleftarrow`.

Support for `\MakeAboxedCommand`, which was missing in the past, has been added in this release. This includes a non-standard starred version that handles box commands whose contents are in math-mode rather than text-mode, like `\bbox` and `\boxed` versus `\fbox` and `\fcolorbox`.

The `\vcentercolon` macro was incorrectly named `\centercolon` in previous versions, and has been corrected here. This is a breaking change for pages that used the incorrect name, but you can always define `\centercolon` to be `\vcentercolon` if that is an issue.

The `mutlinedgap` configuration option has been renamed to `multlined-gap` to correspond better with other option names (that all use dashes), and there is a new `multlined-width` option has been added to give the default width for `multlined` environments.

Updates to Other Packages

For the *mhchem* extension, several of the arrows were not previously stretchy. This release adds a new mhchem-specific font that includes the characters needed to stretch all the arrows available in *mhchem*, improving its output in both the CHTML and SVG renderers. Note, however, that these fonts match the `mathjax-newcm` font set, and are used no matter what font is selected, so the arrows may not match other arrows used in the font if you are using one other than `mathjax-newcm`.

The *configmacros* package now allows you to create active characters that are bound to macros, so that

```
MathJax = {
  tex: {
    active: {
```

(continues on next page)

(continued from previous page)

```
      'x': '\\mmlToken{mi}[mathvariant="bold"]{x}'
    }
  }
}
```

defines `x` to always produce a boldface "x".

Note that you need to take care not to cause a loop by using the character you are making active in its own definition. In the example above, since the argument to `\mmlToken` is not further processed as TeX commands (except for instances of `\U`), that is not the case here.

A new `formatRef` configuration option has been added to the *tagformat* package that allows you to specify how `\eqref` is formatted. It should be a function that takes one argument, the tag associated with the specified label, and returns the string that should be used in place of the `\eqref`. The default is to use the result of `formatTag`, which is the string that will be used for the equation number on the equation itself. The returned string will be used as the link text for the link that targets the specified expression.

New or Updated Configuration Options

A new `tex.tagAlign` configuration option is now available that specifies how tags should be vertically aligned compared to their equations. The default is to align on the baseline, but you can specify `top`, `center`, `bottom`, `baseline`, or `axis`. One use case for this is when the equation is likely to have automatic line breaks inserted, in which case the baseline will be the baseline of the top line of the equation (in most cases), but you may want to have the alignment be the center of the broken equation rather than the baseline of the top line. Setting `tagAlign` to `center` would make sense in this case, without harming the usual placement for most equations.

A new `tex.mathStyle` configuration parameter has been added to control the italicization of variables in TeX expressions, as can be done in LaTeX via the math-style document setting. This can be set to one of `TeX`, `ISO`, `French`, or `upright`. The setting affects how upper- and lower-case Latin and Greek letters are italicized. `TeX` uses italics for all but upper-case Greek, whereas `ISO` makes everything italic, `upright` makes them all upright, and `French` makes everything upright except lower-case Latin letters.

When converting TeX to MathJax's internal MathML format, the TeX input jax will put multi-letter sequences into a single `mi` element when they appear inside `\mathrm`, `\mathbf`, and related macros. What constitutes a "letter" in this setting is now configurable via the `tex.identifierPattern` configuration option, which is a regular expression that indicates what characters should be combined into one identifier. The default value is `/^[a-zA-Z]+/`, but it can be extended to include other characters (e.g., numbers or accented characters) via this configuration option. Note that the pattern must begin with `^` to tie it to the beginning of the string.

Similarly, there is now a configuration option `tex.ams.operatornamePattern` to specify what should be put into a single `mi` within the argument to `\operatorname`. Because LaTeX treats `-` and `*` as text characters (rather than mathematical operators) within `\operatorname`, the default for this pattern is `/^[-*a-zA-Z]+/`. Again, the pattern should always begin with `^`.

The `tex.digits` option has been renamed `numberPattern` to be more in line with the options above. The `tex.digits` option is retained for backward compatibility, though it will likely be removed in a future release. A new `tex.initialDigit` pattern tells MathJax when to apply the number pattern. This makes it easier to change the number pattern to include other formats, like Persian numerals, for example.

Similarly, a new `tex.initialLetter` pattern has been added that is used to trigger when the the identifier pattern is used. This make using accented characters or other non-Latin characters for multi-letter identifiers easier to configure.

Other TeX Input Changes

In the past, if an array environment had lines around the outside of the array, and there were mixed solid and dotted lines used, then MathJax might change some of them so that they are all the same style. This has been fixed in this version, so the boundary lines should now have the correct style in all cases.

The checking for proper nesting of AMS environments has been improved. This may affect existing expressions that are improperly nested but were not flagged by MathJax in the past. Previously, there was no check that these environments appeared at the top level of the expression, so an `align` environment could be used inside an `array`, for example; this now generates an error. On the other hand, `gather` should be allowed within `align` (but not another `gather`), but was being flagged as erroneous nesting; this is now allowed.

The TeX input jax now attaches `data-latex` attributes to the MathML elements that it produces, indicating the TeX command from which the element originated. This information can be used by the assistive tools to produce Braille output of the original LaTeX, for example. Since `data` attributes are transferred to the CHTML and SVG output nodes, this information is available in MathJax's output in the page, not just the internal MathML notation.

Because the MathML specification indicates that any `mtext` element is "space-like", and since an operator in an `mrow` whose only other elements are space-like is considered to be an "embellished operator" that should be treated as an unbreakable unit, this can lead to unexpected results. When the operator is used for line breaking, the line break must occur before or after the embellished operator as a whole. That is, `{\text{A} + \text{B}}` produces `<mrow><mtext>A</mtext><mo>+</mo><mtext>B</mtext></mrow>`, making the `<mo>+</mo>` an embellished operator; if a linebreak is to occur at this +, it will be done before the `A` or after the `B`, not at the + itself. This is not what is usually intended for this LaTeX expression. Although the MathML specification is not clear about why `mtext` elements are space-like, it is likely because these are sometimes used to insert explicit spaces into the expression via space characters, but *any* `mtext` is considered space-like regardless of its content, leading to awkward situations like the one described above.

In version 4, MathJax has parted from the specification in making an `mtext` element be space-like only if its contents consists solely of space characters or is empty and it doesn't have a `mathbackground` or `style` attribute. Similarly, an `mspace` element is considered space-like only if it does not have an explicit `linebreak`, `height`, `depth`, `mathbackground` or `style` attribute. With these changes, TeX expressions will not generate unexpected embellished operators that will affect their line breaking.

44.1.8 Output Improvements

A new `output` section has been added to the MathJax configuration object that allows you to specify output options that are common to both output renderers. That way, if the user switches the renderer via the MathJax contextual menu, the same settings will be used in the new renderer without needing additional configuration. The individual `chtml` and `svg` blocks then should only include options that are specific to those output formats.

CHTML Improvements

An important improvement has been made in v4 to the way the CHTML output renderer handles stretchy delimiters. In the past, the CHTML output would use CSS transforms to stretch the extender to the needed size. This lead to alignment issues between the extenders and the end pieces in some browsers, and in some cases, thin extenders disappeared entirely. In this version of MathJax, the extenders are made from repeated copies of the extender piece, slightly overlapping, and clipped to the proper size. This makes for cleaner and more reliable assemblies for stretched characters, both vertically and horizontally, and should eliminate the rendering problems seen in v3.

The MathJax layout uses a font that has larger depth below the baseline than most standard fonts. When the MathJax characters are drag selected, this leads to the selection bounding box being larger than expected. With this release, the CHTML output now uses clip paths to restrict the bounding boxes, making for more accurate selection backgrounds, and preventing unwanted vertical scroll bars in some displayed equations.

An improvement has been made in a long-standing issue with the CHTML output in WebKit-based browsers, like Safari, where characters (particularly in runs of text) would not line up on the baseline properly. This is a bug in WebKit, but

this version of MathJax includes a work-around that should help with the alignment in `\text{}`, `\mathrm{}` and similar macros, and other situations where the characters are grouped into a single MathML element.

SVG Improvements

The SVG output jax has two new configuration options. The first is `svg.blacker`, which is a number that indicates the stroke width (in thousandths of an em) to use for the character paths. Because some parts of some characters are very thin, the default is 3 in an attempt to help prevent those sections from disappearing at small sizes. Some page authors may wish to increase or decrease this in order to help the weight of the MathJax fonts better match the surrounding text font. A value of up to 15 may make sense in those situations. Values above 30 will likely cause some characters to render poorly.

The second is `svg.useXlink`, which is either true or false, and specifies whether the SVG elements should use `xlink` namespaces for their `href` attributes. In HTML5, the `xlink` namespace is no longer necessary, but older systems may still require it, so the option is available. The default is `true`.

The SVG output jax now groups characters that are not in the MathJax fonts into a single <text> element. That allows combining characters to combine, so that languages and emojis that use multiple characters to form a single glyph will be handled properly. So use `\text{}` around such characters to allow them to combine properly. Note that this was already being done for CHTML output, so this brings the two in line with each other.

Assistive MathML Size

In the past, the hidden MathML that is produced by the *a11y/assistive-mml* extension could be rendered by the browser larger than the math typeset by MathJax, which could interfere with the size of the container for the expression. Version 4 includes additional CSS to resolve this problem. Note, however, that the *assistive-mml* extension is off by default in v4, and is only active if the user enables it in the MathJax contextual menu, or if the page author enables it in the MathJax configuration object.

44.1.9 MathJax v4.0 and Promises

Because the new MathJax fonts include more extensive character coverage, meaning much more data is required, the fonts have been broken down into smaller pieces that can be loaded dynamically, rather than being one big data file, as was the case with version 3. This allows the initial download of MathJax to be smaller, while still accommodating rarely used glyphs for those who need them.

As a result, however, when the data for one of these ranges is needed, MathJax will pause and wait for the data to arrive from the CDN. That means that producing MathJax output is now potentially an asynchronous process, which was not the case in v3 (where only the input processing could be asynchronous). In the past, as long as you pre-loaded all the TeX extensions that you needed (e.g., with one of the `-full` components), you could use synchronous calls to *MathJax.tex2svg()* or the other similar functions. With the new (larger) dynamic fonts, that is no longer guaranteed. That means you should instead use the promise-based versions of these calls, like *MathJax.tex2svgPromise()*, in order to properly handle the potential for dynamically loaded font data.

When using the synchronous calls, you may get errors indicating a "MathJax retry", which is what MathJax uses to mediate its asynchronous loading actions. The promise-based functions handle these retry errors automatically, but the synchronous functions expect you to trap them and handle them yourself, usually through the *mathjax.handleRetriesFor()* function.

Properly handling promises may mean reorganizing how your own code works. If you can not avoid using synchronous calls, then you may need to load all the font dynamic data up front using a single promise-based call before you start using MathJax synchronously. This can be done using

```
MathJax.startup.document.outputJax.font.loadDynamicFiles();
```

to load all the font dynamic data. This function returns a promise, and you should wait for it to resolve before calling any MathJax conversion functions. Note, however, that there can be a *lot* of font data, and these fonts may include many characters that will never get used, so only do this if you absolutely have to. It is better to use the promise-based conversion functions if you can.

For node applications, you can use

```
MathJax.startup.document.outputJax.font.loadDynamicFilesSync();
```

to load the font data synchronously, provided you have defined the MathJax loading mechanism by importing `@mathjax/src/js/util/asyncLoad/node.ts` before hand.

44.1.10 MathJax User-Interface Updates

Since MathJax v4 includes a significant rewrite of the expression explorer, this has lead to a reorganization of the MathJax contextual menu that moves the accessibility options to a more prominent position for easier access and better control. The top-level menu now includes an *Accessibility* section with four submenus — *Speech*, *Braille*, *Explorer*, and *Options* — rather than an accessibility submenu as in previous versions. The Speech menu allows you to enable/disable speech generation and its associated visual output, and to turn on or off auto voicing. It also provides control over the speech rule-set to use, the verbosity of the set in use, and the language to use for the speech. Similarly, the Braille menu allows you to enable/disable Braille generation and display, as well as to select the type of Braille to generate.

The explorer controls for magnification and highlighting have been moved to the *Explorer* menu, and other accessibility options have been moved from the *Math Settings* and old *Accessibility* submenus to the *Options* menu. A new *Semantic Enrichment* option controls whether the accessibility features are available or not (unchecking it disables speech and Braille generation and the explorer).

Several new items have been added to the *Show Math As* and *Copy to Clipboard* submenus of the MathJax contextual menu. These include:

- *Speech Text*, which is the generated speech string for the mathematical expression.
- *Braille Code*, which is the Braille string for the mathematical expression.
- *SVG Image*, which is a serialized SVG object representing the expression, which can be pasted into a stand-alone image file for use elsewhere.
- *Error Message*, which is the full error message when there is a TeX or MathML input error, or an internal MathJax error. In particular, when the TeX *noerrors* extension is used (so that error messages are not displayed within the page), this can give you the actual error message for an expression that doesn't typeset.

Note that *Speech Text* and *Braille Code* are only available when their associated menu items in the accessibility section are enabled (as is the case for the default combined components). Similarly, *SVG Image* is only available when the SVG output jax is available (either in a configuration that loads it, or if the user changes to SVG output in the contextual menu).

There is also a new *MathML/SVG has* entry in the *Math Settings* submenu that controls what attributes are included in the MathML and SVG produced by the *Show Math As* and *Copy to Clipboard* menu items. The *TeX hints* and *Original as annotation* items have been moved there, and there are two new items: *Semantic attributes* and *LaTeX attributes*. The first controls whether to include the attributes that have been added by the semantic enhancement; there are a lot of these, and they can make the MathML hard to read, and generally are not necessary for use outside of MathJax, so the default is to filter these attributes, but you can uncheck that item if you want to include them in the MathML output. The second controls whether to include the *data-latex* attributes that the TeX input jax adds to the internal MathML to indicate the LaTeX commands that generated the given MathML. These are included by default, but can be turned off with this menu item.

44.1.11 MathJax API Changes

There are several MathJax API changes in this release, though most should not be breaking changes, as described below.

New Promise-Based Functions

Some actions in MathJax require loading extra code from an extension, which is an asynchronous action, as the browser must wait for the file to be loaded before it can use it. In MathJax v3, such asynchronous actions were mostly associated with loading TeX extension packages, and that could be avoided by pre-loading the extensions that are needed, so that typesetting could be performed synchronously. In v4, with fonts that have much greater coverage than in v3, some font data may need to be loaded asynchronously as well, and that means that typesetting may be asynchronous even if all the needed TeX extensions are pre-loaded. As a result, the *MathJax.typesetPromise()* function is more likely to be needed, and *MathJax.typeset()* will only work if no font data needs to be loaded. This is discussed in more detail in the *MathJax v4.0 and Promises* section.

Because of this greater need to handle asynchronous file loading, several new functions have been added to the `MathDocument` class to provide promise-based versions of the corresponding synchronous calls. These include *mathDocument.convertPromise()*, `mathDocument.renderPromise()`, and `mathDocument.rerenderPromise()`, which wrap the *mathDocument.convert()*, `mathDocument.render()`, and `mathDocument.rerender()` methods in the needed *mathjax.handleRetriesFor()* call and return its promise. This makes it easier to perform these actions when font data or TeX extensions need to be loaded than having to use `mathjax.handlerRetriesFor()` yourself.

In the past, promise-based functions, like *MathJax.typesetPromise()*, *MathJax.tex2chtmlPromise()*, etc., could not be called while another one was currently in effect. That is, you needed to wait for the promise from one such call to resolve before you could do the next call, and the documentation encouraged you to use *MathJax.startup.promise* to help chain these calls together. In v4, these functions now use an internal promise (associated with the `MathDocument`) to prevent more than one from running concurrently, so these calls chain automatically. In particular, you should no longer use *MathJax.startup.promise* yourself to serialize your calls to these functions.

You may wish to use the new `MathDocument` promise to synchronize other code with MathJax's typesetting operations without having to keep track of the promises returned by the various promise-based functions. For this reason, MathJax provides a new `mathDocument.whenReady()` method of the `MathDocument` class. It takes a function as its argument, and performs that action when its internal promise is resolved; that is, when any previous promise-based typesetting or conversion actions complete. You can think of `mathDocument.whenReady()` as queuing your action to be performed whenever MathJax has finished anything that has been queued previously.

The function you pass to `mathDocument.whenReady()` can return a promise (if it starts any asynchronous actions of its own, for example), in which case that promise must be fulfilled before any further `mathDocument.whenReady()` actions will be performed. For example

```
const doc = mathjax.document('', {});
doc.whenReady(() => console.log('A'));
doc.whenReady(() => {
  return new Promise((ok, fail) => {
    setTimeout(() => {
      console.log('B');
      ok();
    }, 1000);
  });
});
doc.whenReady(() => console.log('C'));
```

would print `A` to the developer console, then a second later print `B` followed by `C`.

Changes to Speech Generation

In v3, the speech generation was performed within the *a11y/semantic-enrich* component along with the semantic enrichment of the internal MathML representation of the mathematical expressions that it processes. In v4, these two functions have been separated from each other, and the speech-generation functionality is performed in a new *a11y/speech* component. This is included in all the combined components, but can be loaded individually by including `a11y/speech` in the `load` array of the `loader` block of your MathJax configuration.

The section on *Technical Details* already mentions the new `MathJax.done()` function that is used to shut down the web-worker or node worker-thread that is created for speech production. There is a corresponding new `mathDocument.done()` method for the `MathDocument` class that can be used in applications that don't use the MathJax Component framework, but rather call on MathJax modules directly.

Named Access to Input Jax

A MathDocument's `inputJax` array included any input jax that you have loaded. E.g., in the `tex-mml-svg.js` combined component, it would contain entries for both the TeX and MathML input jax. Because this is an array, it was not obvious in v3 which of the two entries was which (you would need to check each entry's `name` property to see if it is the one you want). In this release, the `inputJax` array also includes properties that point to the input jax by name. That is, `mathDocument.inputJax.tex` will point to the TeX input jax, if any, and similarly for `mathDocument.inputJax.mathml`.

Change to ES6 Modules

The fact that the webpacked components are now ES6 files (see the section on *MathJax ES6 Modules*) means that MathJax will no longer run in IE11, so you should no longer include the `polyfill.io` script that was recommended in the documentation for IE11 support.

The `es5` directory has been removed from the MathJax distribution, so the `/es5` should be removed from the URL used to access MathJax's components. In the `mathjax` npm package, the files from the `es5` directory are now in the main directory, and for `mathjax-full` (now called `@mathjax/src`), they are in the generic `bundle` directory.

Changes to Configuration Options

The `tex.skipHtmlTags` configuration property now includes `select` and `option` tags, since pop-up menu items can only contain textual content, not other HTML tags.

In addition to the new configuration options discussed in other sections, there are several additional options available in this release:

- Two new settings in the `options.menuOptions.settings` configuration object: `showSRE` and `showLatex`, which control whether to include the data attributes generated by the speech-rule-engine or the `data-latex` attributes in MathML and SVG output in the *Show Math As* and *Copy to Clipboard* menus.
- `mathml.verify.checkMathvariants`, which controls whether the MathML input jax will check that `mathvariant` attribute values are valid math variants and report an error if not. Invalid `mathvariant` values can cause MathJax to crash under some circumstances, so the default value of this option is `true`, but this may cause current expressions with invalid math variant values that used to render to now show those nodes as having errors.

The `lineWidth` property of the `Metrics` object (used to store information about the font metrics of the container surrounding an expression) has been removed, as the line-breaking algorithm ended up using the `containerWidth` property directly. That affects functions that accept metric data as their inputs (such as *`mathDocument.convert()`* and *`MathJax.tex2chtml()`*), as these will no longer accept `lineWidth` in the options passed to them.

Some backward-compatibility code in v3 has been removed; e.g., when the `tex.multlineWidth` configuration option was moved to `tex.ams.multlineWidth` in an earlier version, there was code to move the old value to the new location, but that code has been removed in v4.

Changes to the Code Base

The MathJax code base has undergone a major cleanup effort for this release, using `eslint` and `prettier` to format the code consistently, and new life-cycle scripts to perform these actions have been added to the `package.json` file. Other modernizations, like moving from `String.substr()` to `String.substring()` were also performed.

A number of object name changes are listed in the *Breaking Changes in V4* section.

Finally, MathJax's test suite has been expanded to include more than 3,000 tests. We have full coverage for the TeX input jax and the `ts/util` directories, but more tests need to be written for other sections of the code base. This is an ongoing project that will take time to complete.

44.1.12 Changes to the Build Tools

With the v4 release, MathJax has switched to the `pnpm` package manager rather than `npm`. This speeds up installation and improves script handling. Although you can still use `npm`, some of the scripts in `package.json` call `pnpm`, so you will need to have `pnpm` installed to use those scripts. Fortunately, this only affects those who are compiling and packaging MathJax, so unless you are working with the MathJax source files, you should not be affected by this change. If you are only using MathJax in web pages via a CDN, for example, you will not need to worry about `pnpm` (or `npm`).

To install `pnpm` you can use

```
npm install -g pnpm
```

The tools used for creating the webpacked component files have been significantly updated for v4. Partly this is to accommodate the changes needed for the production of the dual MJS/CJS files, and partly in order to make it easier to do individual steps of the build process in isolation. There are now package scripts for performing most of the development tasks that you might need to do if you are modifying MathJax or building your own components.

In the past, to compile the typescript files into javascript, you would use `pnpm -s compile`, and to build the webpacked component files, you did `pnpm -s make-components`. These commands are still available, but there is a new

```
pnpm -s build
```

command that does both at once.

Because we now make both MJS and CJS versions of the MathJax files, there are commands to make each type. The commands above make the MJS versions, but there are also module-specific commands that make the MJS and CJS versions separately.

```
pnpm -s compile-cjs
pnpm -s compile-mjs

pnpm -s make-cjs-components
pnpm -s make-mjs-components

pnpm -s build-cjs
pnpm -s build-mjs
```

The generic versions are just aliases for the MJS-specific ones. Note that because the webpacked versions use the MJS JavaScript files, the `make-cjs-components` script is never run, but if you want to make ES5-based versions of the webpacked files,

```
pnpm -s compile-cjs
pnpm -s make-cjs-components
```

will create a `bundle-cjs` directory containing the ES5 webpacked files comparable to the ones that used to be in the `es5` directory.

There is also

```
pnpm -s build-all
```

that does both the `build-mjs` and `build-cjs` actions, creating all the files in the `mjs`, `cjs`, `bundle` and `components/cjs` directories.

Because the `make-components` action webpacks *all* the components, a time consuming process, there is a `make-one` script that webpacks only one component. The format for this is

```
pnpm -s make-one <component> <module-type>
```

where `<component>` is the name of the directory in `components/mjs` that defines the component, and `<module-type>` is either `mjs` or `cjs`. For example

```
pnpm -s make-one input/tex mjs
```

would pack only the TeX input component.

These commands all rely on the `components/bin/makeAll` script, which has been enhanced for version 4. It now has a number of command-line options to control its functions, including:

- `--no-subdirs` to prevent it from processing all the subdirectories of the given directory.
- `--cjs` to process using CJS rules.
- `--mjs` to process using MJS rules (the default).
- `--terse` to only print the main headings rather than the file details like the files included in a webpacked version.
- `--build` to only perform the build steps (i.e., creating the `lib` directories used for shared imports).
- `--copy` to only perform the copy steps (e.g., copying the CHTML woff files into place).
- `--pack` to only do the webpack steps.
- `--bundle-cjs` to webpack into the `bundle-cjs` directory rather than the `bundle` directory.

These can also be passed to `pnpm -s -- make-one` if you want to restrict the steps performed (it already uses `--no-subdirs` and one of `--cjs` or `--mjs`).

These changes mean you only need to use `makeAll`, since it handles calling `components/bin/build`, `components/bin/copy` and `components/bin/pack` itself. The arguments for these sub-programs have changed in this version, particularly for `components/bin/pack`, so you should not call these by hand yourself unless you have looked at the internals of them carefully.

44.1.13 Breaking Changes in V4

A number of breaking changes have already been mentioned elsewhere:

- *MathJax Scoped NPM Packages* discusses the change to scoped npm packages, and in particular, moving from `mathjax-full` to `@mathjax/src`.
- *MathJax v4.0 and Promises* discusses the greater need to use promises in v4.
- *MathJax ES6 Modules* discusses the changes to the directory structure and the removal of the `/es5` from the URLs used to obtain MathJax from a CDN.

- *Input Improvements* discusses some potentially breaking changes for some pages, including the fact that the the *textmacros* extension is now included in all combined components, the update to the *mathtools* extension to include the changes in the corresponding LaTeX package from 2022 and 2024, and the change from `digits` to `numberPattern` for configuring the pattern used to identify a number in TeX input.

In addition to those, there are a number of other potentially breaking changes, as described below.

Removal of AllPackages

The `AllPackages.ts` file and the `all-packages` extension were intended as a means of loading most of the TeX extensions up front so that you did not need to worry about asynchronous loading of extensions via the `autoload` package. But now that MathJax's output is also asynchronous, using `AllPackages` is no longer sufficient to allow for synchronous processing. As more extensions have been created, they have not all been added to `AllPackages.ts`, and as the library of extensions, both core and third-party, grows, it is impractical to keep all of them in one package. So in this release, these files have been removed. For those employing MathJax in node applications, you can use

```
import {source} from '@mathjax/src/components/js/source.js';
const AllPackages = Object.keys(source).filter((name) => name.substring(0,6) === '[tex]/
↪').sort();
```

to get a list of the main TeX packages. For use on the web, you could use

```
<script type="importmap">
{
  "imports": {
    "#source/source.cjs": "https://cdn.jsdelivr.net/npm/@mathjax/src@4/components/mjs/
↪source-lab.js"
  }
}
</script>
<script type="module">
import {source} from 'https://cdn.jsdelivr.net/npm/@mathjax/src@4/components/mjs/source.
↪js';
const load = Object.keys(source).filter((name) => name.substring(0,6) === '[tex]/').
↪sort();
const packages = ['base'].concat(load.map((name) => name.substring(6)));
window.MathJax = {
  loader: {load},
  tex: {packages}
};
</script>
```

to load all the extensions. Add any other configuration options that you need to the `window.MathJax` variable.

Changes for Speech Generation

The *MathJax API Changes* section describes the separation of the semantic enrichment from the speech generation and the introduction of a new *speech* component. Because the speech is now generated within a web worker or node worker thread, the speech-generation code is no longer in the main MathJax components, but is in a separate file that is run in the worker. That means there is no more access to speech generation directly within MathJax (it is only available in the worker). In particular, the `ts/a11y/sre.ts` file now only includes the semantic-enrichment methods of the speech-rule engine, and `Sre.toSpeech()` is no longer available. In a node application, you can load this function directly from the speech-rule-engine's API, however. Here is an example command-line script that takes a TeX expression and returns its speech string using this approach:

```
//
//  Load the modules needed for MathJax
//
import {mathjax} from '@mathjax/src/js/mathjax.js';
import {TeX} from '@mathjax/src/js/input/tex.js';
import {liteAdaptor} from '@mathjax/src/js/adaptors/liteAdaptor.js';
import {RegisterHTMLHandler} from '@mathjax/src/js/handlers/html.js';
import {SerializedMmlVisitor} from '@mathjax/src/js/core/MmlTree/SerializedMmlVisitor.js
↪';
import {STATE} from '@mathjax/src/js/core/MathItem.js';

//
// Import the speech-rule-engine
//
import '@mathjax/src/components/require.mjs';
import {setupEngine, engineReady, toSpeech} from 'speech-rule-engine/js/common/system.js
↪';

//
// Import the needed TeX packages
//
import '@mathjax/src/js/input/tex/base/BaseConfiguration.js';
import '@mathjax/src/js/input/tex/ams/AmsConfiguration.js';
import '@mathjax/src/js/input/tex/newcommand/NewcommandConfiguration.js';
import '@mathjax/src/js/input/tex/noundefined/NoUndefinedConfiguration.js';

//
// The em and ex sizes and container width to use during the conversion
//
const EM = 16;          // size of an em in pixels
const EX = 8;           // size of an ex in pixels
const WIDTH = 80 * EM;  // width of container for linebreaking

//
// Create DOM adaptor and register it for HTML documents
//
const adaptor = liteAdaptor({fontSize: EM});
RegisterHTMLHandler(adaptor);

//
// Create input jax and a (blank) document using it
//
const tex = new TeX({
  packages: ['base', 'ams', 'newcommand', 'noundefined'],
  formatError(jax, err) {console.error(err.message); process.exit(1)},
  //
  // Other TeX configuration goes here
  //
});
const html = mathjax.document('', {
  InputJax: tex,
  //
  // Other document options go here
```

(continues on next page)

(continued from previous page)

```
  //
});

//
//  Create a MathML serializer
//
const visitor = new SerializedMmlVisitor();
const toMathML = (node => visitor.visitTree(node, html));

//
// Convert the math from the command line
//
const mml = html.convert(process.argv[2] || '', {
  display: true,
  em: EM,
  ex: EX,
  containerWidth: WIDTH,
  end: STATE.CONVERT         // stop after conversion to MathML
});

//
// Set up the speech engine to use English
//
const locale = process.argv[3] || 'en';
const modality = locale === 'nemeth' || locale === 'euro' ? 'braille' : 'speech';
await setupEngine({locale, modality}).then(() => engineReady());

//
// Produce the speech for the converted MathML
//
console.log(toSpeech(toMathML(mml)));
```

With the speech generation being performed in a worker, the process is now inherently asynchronous, as the communication between the main thread and the worker thread is mediated by promises. That means that speech generation can't be done synchronously, and you must use the promise-based functions for handling typeset and conversion operations that involve speech, unless you use a technique like the one above.

The speech-generation process now applies the speech attributes to the DOM nodes themselves, rather than to the internal MathML structure (as was done in v3), so serialized versions of the internal MathML will not include the speech as they did in the past.

Object and Type Name Changes

Some name changes have occurred within the code to help clarify the purpose of some objects or methods. In particular, the `Loader.preLoad()` method has been renamed `Loader.preLoaded()` in order to make it clear that this does not itself load the given components, but that your code has done that and you are telling MathJax that they have already been loaded.

Another change involves the objects used to handle CSS styles. MathJax has two object classes that deal with CSS definitions, one that specifies CSS styles via object literals (essentially JSON structures), and one that parses a CSS string into an object structure that acts like a DOM element's `style` attribute. Moreover both modules declared a `StyleList` type, and these were not compatible. That both caused confusion and complicated their use together in the same module, so the names for these types and objects have been changed in this release in order to make it clearer which is which.

To accomplish this, the `ts/util/CssStyles.js` file was renamed to `ts/util/StyleJson.js`, and with that, `StyleList` is changed to `StyleJson`, `StyleData` to `StyleJsonData`, and `CssStyles` to `StyleJsonSheet`. This more accurately describes what this object does (it is the one that takes JSON data), and what the objects represent, while the `ts/util/Style.ts` file implements (a subset of) the DOM object `style` attribute.

The move to ESM modules and compiling to ES6 rather than ES5 lead to an issue with the webpacked versions of some component files that would cause errors when they are loaded. The source of the problem was due to the use of a custom `Symbol` class in MathJax's TeX input jax that conflicts with the native javascript `Symbol` object. This was not an issue in previous versions of MathJax, but due to differences between how webpack handles CommonJS and ESM modules, it caused problems with some TeX extension packages that use the `Symbol` class. This has lead us to rename the custom `Symbol` class to `Token`, and rename the `Symbol.ts` and `SymbolMap.ts` files to `Token.ts` and `TokenMap.ts`. This is a potential breaking change to those who have created their own TeX extension packages that load one of these files.

The names of a number of internal objects have been normalized to be Pascal case (like camel case, but with an initial upper-case letter). For example, the internal MathML items like `CHTMLmath` have been renamed as `ChtmlMath`. Such changes would only affect those writing their own extensions, but for those who do, you may need to adjust the names of classes like this.

Finally, the `latest.js` file has been removed, as `jsdelivr.net` and other CDNs handle providing the latest version automatically, and with more granularity than this file did.

See also the MathJax v4.0 release notes for a list of the commits and bug fixes in this release.

44.2 What's New in MathJax v3.2

Version 3.2 includes a number of new features, as well as bug fixes for several issues with version 3.1. The new features are described below.

See also the release notes for the list of bugs that have been fixed in version 3.2.

44.2.1 Lazy Typesetting

Although MathJax version 3 is already an order of magnitude faster than version 2, with version 3.2 we offer a new extension that is designed to make pages with large numbers of equations perform even better. It implements a "lazy typesetting" approach that only typesets an expression when it comes into view, which means that expressions will not be typeset when they are not visible. Your readers will not have to wait for the entire document to typeset, which can speed up their initial view of the page. Furthermore, any expressions that are never seen will not be typeset. This also helps with the situation where you may link to a particular location in your page (via a URL with a hash); in version 2, typesetting the material above that point can cause the browser to change the scroll position, and so the user may not end up at the proper location in the page. With the lazy extension, the material above that point is not typeset until the user scrolls upwards, and so there is no position change.

Lazy typesetting works best with SVG output, but changes (discussed below) with the way the CommonHTML output handles its stylesheet updates make the CHTML output nearly as fast. With TeX input, the lazy extension makes sure that previous expressions are processed by TeX (though not output to the page) so that any macro definitions or automatic equation numbers are in place when the visible expressions are processed. Currently, documents that contain `\ref` or `\eqref` links may not yet work properly, since target equations may not have been typeset, and so the link location may not be marked in the document. In particular, forward references are unlikely to work, and backward references will work only if the target expression has already been typeset. We hope to improve this situation in a future release.

See the *Lazy Typesetting* documentation for information on how to configure MathJax to use this new feature.

44.2.2 CSS Updates

MathJax's CHTML output handles the characters that appear in the math on the page by storing information about their bounding boxes and text content in a CSS stylesheet. When additional math is typeset, this stylesheet may need to be updated, and in previous versions, MathJax would replace the entire stylesheet with a new one. This can cause visual flashing, and can be expensive as the browser must re-evaluate all the rules and apply them again. In version 3.2, the CHTML output now adds rules to the stylesheet individually, so the older rules are not replaced, and only the new rules must be evaluated and applied. This makes updates must faster, and is of particular benefit to the lazy-typesetting extension described above, as the page can be updated many times as equations scroll into view. This change makes the CHTML output work almost as smoothly as SVG output with the lazy extension.

44.2.3 New TeX Packages

Version 3.2 includes nine new TeX extension packages:

- *cases* — provides environments for individually numbered cases.
- *centernot* — implements a centered *not* command (and a non-standard *centerOver* that places one symbol centered on top of another).
- *colortbl* — provides macros for coloring cells of an array or alignment.
- *empheq* — an environment for placing material to the left or right of an alignment that has individual equation numbers.
- *gensymb* — provides macros for some specific units.
- *mathtools* — offers a range of macros and environments for advanced mathematical typesetting.
- *setoptions* — provides the ability to change some TeX input jax options from within an expression (e.g., to change the tag side).
- *textcomp* — provides a range of macros for specifying various text characters.

- *upgreek* — provides macros for upright Greek characters.

These are all included in the components that end in `-full` (and include the TeX input jax), and you can load individual ones as you would other tex packages. Note, however, that none of these are autoloaded, though you can configure the *autoload* extension to do so, if you wish. See the *autoload* documentation for details.

In addition to these new packages, some of the older packages have been updated:

- The *ams* package now includes `flalign`, `xalign`, and `xxalign` environments. In addition, the `multline` extension has been made more compatible with actual LaTeX. In the past, `multline` was set to be 85% of the container width, but now it is set to 100%, but with a 1em indent on both sides; when there is a tag, the indent on the tag side is increased by the width of the tag, as is the case in LaTeX. The width was stored in the `multlineWidth` configuration option in the `tex` configuration block. That has now been moved to the `ams` block in the `tex` configuration, and there is a new `multlineIndent` value. These are set to `100%` and `1em` respectively. To obtain the old behavior, set them to `85%` and `0`. Currently, if `multlineWidth` is found in the main `tex` option block, it will be moved to the `ams` block, but that backward-compatibility code will be removed in a future release.
- The *physics* package now implements all macros, even those that are not officially documented, but are nevertheless available in LaTeX. In addition, it now implements the `italicdiff` and `arrowdel` options.
- **The following macros have been added to the indicated package:**
  - `\overunderset` (ams) — a combination of `\overset` and `\underset`.
  - `\stackbin` (ams) — similar to `\stackrel` but produces a symbol with the spacing of a binary operator.
  - `\nonscript` (base) — apply the following spacing only when in display and text styles.
  - `\boxed` (base) — puts a frame around an expression.
  - `\framebox` (base) — puts a frame around a text argument.
  - `\ip`, `\Bqty`, `\qsince`, `\Residue` (physics) — originally missing from the physics package.

44.2.4 MathML Extensions

The MML3 extension from version 2 has been ported to version 3 and is available to be included when you load the MathML input jax. This extension implements the MathML3 elementary math tags (like `<mstack>` and `<mlongdiv>`) using an XSLT transform to convert these tags into other presentation MathML tags that MathJax has implemented. This does a reasonable job for some constructs, and a poorer job for others, but it does make it possible to process elementary math within MathJax v3. This is an experimental extension as a stop-gap measure until these tags are fully implemented within core MathJax.

See the *Experimental mml3 extension* documentation for information on how to configure MathJax to use this new feature.

44.2.5 Explorer Update

The Speech-Rule Engine (SRE) that underlies MathJax's assistive technology support has been updated to the most recent version (3.3.3). This includes support for the Hindi language, so that the expression explorer can generate speech in Hindi (as well as its other languages: English, French, German, Italian, Spanish, together with Braille support in Nemeth).

See the SRE release notes for details.

This release also ports the remaining missing features for the explorer to v3. This includes summarising expressions and navigation of tabular expressions, like matrices or equation systems. See the *keyboard command* documentation for details.

As of v3.2.1 MathJax pulls in SRE v4 which supports as Catalan, Danish, Norwegian (Bokmal and Nynorsk) and Swedish as additional languages. It also integrates SRE code directly making use of its promise structure, instead of loading it as an external package. Consequently the old *sreReady* method will be deprecated and the loophole to use speech rule engine directly via the *SRE* namespace in the browser, is closed.

See the SRE release notes as well as the MathJax v3.2.1 release notes for details.

44.2.6 Other New Features

In addition to the major features listed above, there are some minor new features as well:

- Packages can now be specified for the *textmacros* extension to the TeX input jax. This allows you to configure additional macros that can be processed within text mode. See the *textmacros* documentation for details.
- Processing of raw Unicode characters in TeX input has been improved. In the past, nearly all non-ASCII characters would be placed within an <mo> element, which is not always the best tag to use. In version 3.2, processing of raw Unicode characters is more nuanced, so that letters are placed in <mi> elements and other symbols in <mo>. For example, a literal Greek alpha (U+03B1) will produce <mi>α</mi> (which is what is generated by \alpha) rather than <mo>α</mo> as in earlier versions. This should provide better results, though perhaps still not perfect in all cases.
- In the past, errors in the MathJax configuration options (such as an unknown option) would produce a fatal error and MathJax would not run. In version 3.2, such errors now produce non-fatal warnings instead, and MathJax will continue to process the remaining options (and then typeset the page). This means that changes to the options (like those described in the breaking changes section below) will not cause your pages to fail outright (though the old options will have no effect). You can configure MathJax to make such errors fatal again, if you wish, and you can provide a function that will be called when there is an option error so that you can more easily trap such errors and handle them yourself. See the *Startup Options* for more details.
- The component loader uses a set of filters to convert a component specification (like `[tex]/physics`) to the full URL for loading the extension. In the past, it was difficult to hook into that filtering mechanism, but in version 3.2, you can now configure additional filters for the loader. See the *Loader Options* documentation for more details.

44.2.7 Breaking Changes in this Release

Some of the changes made to the options to accommodate the updated speech-rule engine are potentially breaking changes, in that the previous options (`enrichSpeech`, `a11y.locale`, `a11y.speechRules`) are no longer valid options. Version 3.1.4 includes code to transfer the old options to their new locations, but that code has been removed in version 3.2. As errors in options are no longer fatal (unless you configure them to be), this change will no longer cause MathJax to fail, but will cause warning messages in the browser console, so look there for such error reports.

Similarly, the code that automatically renames the older TeX package names to their current all-lower-case versions (e.g., `configMacros` to `configmacros` and `colorV2` to `colorv2`) has been removed from version 3.2. If you are using old package names, you will need to update your configuration. This applies to `\require{}` macros that refer to the older names as well as their names in the `loader` section, the `tex.packages` array, and the `tex.autoload` block.

Version 3.2 removes the `matchFontHeight` option for the SVG output jax, since it only applies to the CommonHTML output, but was previously allowed in the `svg` configuration block, while doing nothing.

Version 3.2 removes of the `toArray()` method from the `LinkedList` class (and its subclasses), so any custom code that uses that should switch to using `Array.from(...)` around the list instead.

Finally, the `Box.ts` and `CssStyles.ts` (and their associated `.js` files) have been moved from the `output` directories to the `util` directory. Compatibility files were placed in the original locations so that older code would continue to work, but these have been removed in v3.2, so you should modify any custom code that loads these files to obtain them from the `util` directory instead.

44.3 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.

44.3.1 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.

44.3.2 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* documentation.

44.3.3 Noundefined Package Options

The `noundefined` package now has configuration options similar to the ones available in the ones available in version 2. These include the ability to set the text color, background color, and size of the text to use for displaying undefined macro names within TeX formulas. See the *noundefined options* for details.

44.3.4 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.

44.3.5 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.

44.3.6 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 language. 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 Extensions* for more details.

44.3.7 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.

44.3.8 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 elements. 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.

44.3.9 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.

44.3.10 New API for Clearing Typeset Content

If you are dynamically adding and removing content from your page, you need to tell MathJax about 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.

44.3.11 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 location 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.

44.3.12 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()`.

44.3.13 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).

44.3.14 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.

44.4 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.

44.4.1 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.

44.4.2 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.

44.4.3 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 Mathematics* for details.

44.4.4 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.

44.4.5 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.

44.4.6 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 of MathJax 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).

44.4.7 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 Mathematics* for more details.

44.4.8 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.

44.4.9 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.

44.4.10 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.

44.5 What's New inEarlier Versions

44.5.1 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.

44.5.2 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 *screen-reader support* for details on the expected behavior as well as background on the limitations due to lack of web standards and browser/OS technology.
  - *Accessible 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 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.

44.5.3 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 lambda, 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 conversion 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 degradation.
- #1016 Fix bug related to <`script`> elements with namespace prefix, e.g., in xHTML.

44.5.4 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).

44.5.5 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)

44.5.6 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 environment closes with unbalanced braces. (Issue #454)
- Allow spaces in the RGB, rgb, and greyscale color specifications. (Issue #446)
- Process `\$` in `\text` arguments. (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 absolutely 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 disappear 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 preceding 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)

44.5.7 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 typeset 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 performance 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 interpreted 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 their 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.

44.5.8 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 typesetting

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 typesetting. (Since the size of the page may have changed due to the mathematical typesetting, 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 occurred 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 parameters 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 typesetting. Since the size of the page may have changed due to the mathematical typesetting, 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 ⁡ 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 details.)
- 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 appropriate 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).

44.5.9 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.

44.5.10 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: [ ['$','$'], ['\\(','\\)'] ],
      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.

44.5.11 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"]
  });
</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, Inc., is a non-profit organization, registered with the IRS as a public charity under section 501(c)(3), with EIN 88-1669159, and incorporated in West Virginia. Any contributions of support you make to MathJax, Inc., are tax deductible.

This version of the documentation was built May 04, 2026.

INDEX