MathJax Documentation

Release 2.3

Davide Cervone, Casey Stark, Robert Miner, Paul Topping

Apr 10, 2017

Contents

MathJax is an open-source JavaScript display engine for LaTeX, MathML, and AsciiMath notation that works in all modern browsers.

CHAPTER 1

Basic Usage

What is MathJax?

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

MathJax uses web-based fonts (in those browsers that support it) to produce high-quality typesetting that scales and prints at full resolution (unlike mathematics included as images). MathJax can be used with screen readers, providing accessibility for the visually impaired. 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 will even convert TeX notation into MathML, so that it can be rendered more quickly by those browsers that support MathML natively, or so that you can copy and paste it into other programs.

MathJax is modular, so it loads 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. Finally, MathJax has a rich application programming interface (API) that can be used to make the mathematics on your web pages interactive and dynamic.

Getting Started

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, SVG or MathML equations for viewing in any modern browser.

There are two ways to access MathJax: the easiest way is to use the copy of MathJax available from a distributed network service such as `cdnjs.com`, but you can also download and install a copy of MathJax on your own server,

or use it locally on your hard disk (with no need for network access). All three of these are described below, with links to more detailed explanations. 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 order to customize the setup for your pages.

Using 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 two things:

1. Link to MathJax in the web pages that are to include mathematics.
2. Put mathematics into your web pages so that MathJax can display it.

Warning: We retired our self-hosted CDN at *cdn.mathjax.org* in April, 2017. We recommend using cdnjs.com which uses the same provider. The use of `cdn.mathjax.org` was governed by its terms of service.

To jump start using *cdnjs*, you accomplish the first step by putting

```
<script type="text/javascript" async
  src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.3.0/MathJax.js?config=TeX-MML-
↪AM_CHTML">
</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 version of MathJax from the distributed server, and configure it to recognize mathematics in both TeX, MathML, and AsciiMath notation, and ask it to generate its output using HTML with CSS to display the mathematics.

Warning: The `TeX-MML-AM_CHTML` configuration is one of the most general (and thus largest) combined configuration files. We list it here because it will quickly get you started using MathJax. It is probably not the most efficient configuration for your purposes and other *combined configuration files* are available. You can also provide additional configuration parameters to tailor one of the combined configurations to your needs or use our development tools to generate your own combined configuration file.

More details about the configuration process can be found in the *Loading and Configuring MathJax* instructions.

Note: To see how to enter mathematics in your web pages, see *Putting mathematics in a web page* below.

Installing 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. To do so you will need to do the following things:

1. Obtain a copy of MathJax and make it available on your server or hard disk.
2. Configure MathJax to suit the needs of your site.

3. Link MathJax into the web pages that are to include mathematics.
4. Put mathematics into your web pages so that MathJax can display it.

Obtaining and Installing MathJax

The easiest way to set up MathJax is to obtain the v2.1 archive from the MathJax download page (you should obtain a file named something like `mathjax-MathJax-v2.1-X-XXXXXXXX.zip` where the X's are random looking numbers and letters). This archive includes both the MathJax code and the MathJax webfonts, so it is the only file you need. Note that this is different from v1.0 and earlier releases, which had the fonts separate from the rest of the code.

Unpack the archive and place the resulting MathJax folder onto your web server at a convenient location where you can include it into your web pages. For example, making `MathJax` a top-level directory on your server would be one natural way to do this. That would let you refer to the main MathJax file via the URL `/MathJax/MathJax.js` from within any page on your server.

Note: While this is the easiest way to set up MathJax initially, there is a better way to do it if you want to be able to keep your copy of MathJax up-to-date. That uses the Git version control system, and is described in the *Installing MathJax* document. If you prefer using Subversion, you can also use that to get a copy of MathJax (see *Installing MathJax via SVN*).

Once you have MathJax set up on your server, you can test it using the files in the `MathJax/test` directory. If you are putting MathJax on a server, load them in your browser using their web addresses rather than opening them locally (i.e., use an `http://` URL rather than a `file://` URL). When you view the `index.html` file, after a few moments you should see a message indicating that MathJax appears to be working. If not, check that the files have been transferred to the server completely and that the permissions allow the server to access the files and folders that are part of the MathJax directory. (Be sure to verify the MathJax folder's permissions as well.) Check the server log files for any errors that pertain to the MathJax installation; this may help locate problems in the permission or locations of files.

Configuring your copy of MathJax

When you include MathJax into your web pages as described below, it will load the file `config/TeX-AMS-MML_HTMLorMML.js` (i.e., the file named `TeX-AMS-MML_HTMLorMML.js` in the `config` folder of the main `MathJax` folder). This file preloads all the most commonly-used components of MathJax, allowing it to process mathematics that is in the TeX or LaTeX format, or in MathML notation. It will produce output in MathML form if the user's browser supports that sufficiently, and will use HTML-with-CSS to render the mathematics otherwise.

There are a number of other prebuilt configuration files that you can choose from as well, or you could use the `config/default.js` file and customize the settings yourself. The combined configuration files are described more fully in *Common Configurations*, and the configuration options are described in *Configuration Options*.

Note: The configuration process changed between MathJax v1.0 and v1.1, so if you have existing pages that use MathJax v1.0, you may need to modify the tag that loads MathJax so that it conforms with the new configuration process. See *Installing and Configuring MathJax* for more details.

Linking your copy of MathJax into a web page

You can include MathJax in your web page by putting

```
<script type="text/javascript" src="path-to-MathJax/MathJax.js?config=TeX-AMS-MML_
↪HTMLorMML"></script>
```

in your document's <head> block. Here, 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, you could use

```
<script type="text/javascript" src="/MathJax/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
↪</script>
```

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

```
<html>
    <head>
        ...
        <script type="text/javascript" src="/MathJax/MathJax.js?config=TeX-AMS-MML_
↪HTMLorMML"></script>
    </head>
    <body>
        ...
    </body>
</html>
```

If you have installed MathJax on a server that is in a different domain from the one serving the page that loads MathJax, be sure to read the *Notes About Shared Servers* for more details. In that case, you may wish to consider using the *MathJax CDN* rather than installing your own copy of MathJax.

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 notation. The configuration file used in the examples above tells MathJax to look for both TeX and MathML notation within your pages. Other configuration files tell MathJax to use AsciiMath input. These three formats are described in more detail below.

TeX and LaTeX input

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

The default math delimiters are $$...$$ and \[...\] for displayed mathematics, and \(...\) for in-line mathematics. Note in particular that the $...$ in-line delimiters are **not** used by default. That is because dollar signs appear too often in non-mathematical settings, which could cause some text to be treated as mathematics unexpectedly. For example, with single-dollar delimiters, "... the cost is $2.50 for the first one, and $2.00 for each additional one ..." would cause the phrase "2.50 for the first one, and" to be treated as mathematics since it falls between dollar signs. For this reason, if you want to use single-dollars for in-line math mode, you must enable that explicitly in your configuration:

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  tex2jax: {inlineMath: [['$','$'], ['\\(','\\)']]}
});
</script>
<script type="text/javascript" src="path-to-mathjax/MathJax.js?config=TeX-AMS-MML_
↪HTMLorMML"></script>
```

See the `config/default.js` file, or the *tex2jax configuration options* page, for additional configuration parameters that you can specify for the *tex2jax* preprocessor, which is the component of MathJax that identifies TeX notation within the page. See the *TeX and LaTeX* page for more on MathJax's support for TeX, and in particular how to deal with single dollar signs in your text when you have enabled single dollar-sign delimiters.

Here is a complete sample page containing TeX mathematics (also available in the test/sample-tex.html file):

```
<!DOCTYPE html>
<html>
<head>
<title>MathJax TeX Test Page</title>
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({tex2jax: {inlineMath: [['$','$'], ['\\(','\\)']]}});
</script>
<script type="text/javascript"
  src="https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</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 the page, there are some caveats that you must keep in mind when you enter your mathematics. In particular, you need to be careful about the use of less-than signs, since those are what the browser uses to indicate the start of a tag in HTML. Putting a space on both sides of the less-than sign should be sufficient, but see *TeX and LaTeX support* for 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 the `TeX-AMS-MML_HTMLorMML` configuration. These include:

- *TeX/AMSmath.js*, which defines the AMS math environments and macros,
- *TeX/AMSsymbols.js*, which defines the macros for the symbols in the *msam10* and *msbm10* fonts,
- *TeX/noErrors.js*, which shows the original TeX code rather than an error message when there is a problem processing the TeX, and
- *TeX/noUndefined.js*, which prevents undefined macros from producing an error message, and instead shows the macro name in red.

Other extensions may be loaded automatically when needed. See *TeX and LaTeX support* for details on the other TeX extensions that are available.

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.

Note that this will work in HTML files, not just XHTML files (MathJax works with both), and that the web page need not be served with any special MIME-type. Also note that, unless you are using XHTML rather than HTML, you

should not include a namespace prefix for your <math> tags; for example, you should not use <m:math> except in a file where you have tied the m namespace to the MathML DTD by adding the xmlns:m="http://www.w3.org/1998/Math/MathML" attribtue to your file's <html> tag.

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

Here is a complete sample page containing MathML mathematics (also available in the test/sample-mml.html file):

```
<!DOCTYPE html>
<html>
<head>
<title>MathJax MathML Test Page</title>
<script type="text/javascript"
  src="https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</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>
        <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, but should use explicit open and close tags for all your math elements. For example, you should use

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

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

The component of MathJax that recognizes MathML notation within the page is called the *mml2jax* extension, and it has only a few configuration options; see the `config/default.js` file or the *mml2jax configuration options* page for more details. See the *MathML* page for more on MathJax's MathML support.

AsciiMath input

MathJax v2.0 introduced a new input format: *AsciiMath* notation. For mathematics written in this form, you mark your mathematical expressions by surrounding them in "back-ticks", i.e., `` `...` ``.

Here is a complete sample page containing AsciiMath notation (also available in the test/sample-asciimath.html file):

```
<!DOCTYPE html>
<html>
<head>
<title>MathJax AsciiMath Test Page</title>
<script type="text/javascript"
  src="https://example.com/MathJax.js?config=AM_HTMLorMML-full"></script>
</head>
<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>
```

The component of MathJax that recognizes asciimath notation within the page is called the *asciimath2jax* extension, and it has only a few configuration options; see the `config/default.js` file or the *asciimath2jax configuration options* page for more details. See the *AsciiMath support* page for more on MathJax's AsciiMath support.

Where to go from here?

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

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

If you are trying to use MathJax in blog or wiki software or in some other content-management system, you might want to read about *using MathJax in popular platforms*.

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

If you are having trouble getting MathJax to work, you can read more about *installing MathJax*, or *loading and configuring MathJax*.

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*.

Installing and Testing MathJax

The easiest way to use MathJax is to link directly to the MathJax distributed network service (see *Using a CDN*). In that case, there is no need to install MathJax yourself, and you can begin using MathJax right away; skip this document on installation and go directly to *Configuring MathJax*.

MathJax can be loaded from a public web server or privately from your hard drive or other local media. To use MathJax in either way, you will need to obtain a copy of MathJax. There are three ways to do this: via `git`, `svn`, or via a pre-packaged archive. We recommend `git` or `svn`, as it is easier to keep your installation up to date with these tools.

Obtaining MathJax via Git

The easiest way to get MathJax and keep it up to date is to use the Git version control system to access our GitHub repository. Use the command

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

to obtain and set up a copy of MathJax. (Note that there is no longer a `fonts.zip` file, as there was in v1.0, and that the `fonts` directory is now part of the repository itself.)

Whenever you want to update MathJax, you can now use

```
cd MathJax
git remote show origin
```

to check if there are updates to MathJax (this will print several lines of data, but the last line should tell you if your copy is up to date or out of date). If MathJax needs updating, use

```
cd MathJax
git pull origin
```

to update your copy of MathJax to the current release version. If you keep MathJax updated in this way, you will be sure that you have the latest bug fixes and new features as they become available.

This gets you the current development copy of MathJax, which is the version that contains all the latest changes to MathJax. Although we try to make sure this version is a stable and usable version of MathJax, it is under active development, and at times it may be less stable than the "release" version. If you prefer to use the most stable version (that may not include all the latest patches and features), you will want to get one of the tagged releases. Use

```
cd MathJax
git tag -l
```

to see all tagged versions, and use

```
cd MathJax
git checkout <tag_name>
```

to checkout the indicated version of MathJax, where `<tag_name>` is the name of the tagged version you want to use. When you want to upgrade to a new release, you will need to repeat this for the latest release tag.

Each of the main releases also has a branch in which critical updates are applied (we try hard not to patch the stable releases, but sometimes there is a crucial change that needs to be made). If you want to use the patched version of a release, then check out the branch rather than the tag. Use

```
cd MathJax
git branch
```

to get a list of the available branches. There are separate branches for the main releases, but with `-latest` appended. These contain all the patches for that particular release. You can check out one of the branches just as you would a tagged copy. For example, the branch for the `v2.1` tagged release is `v2.1-latest`. To get this release, use

```
cd MathJax
git checkout v2.1-latest
```

and to update it when changes occur, use

```
cd MathJax
git pull origin v2.1-latest
```

Obtaining MathJax via SVN

If you are more comfortable with the subversion source control system, you may want to use GitHub's `svn` service to obtain MathJax. If you want to get the latest revision using `svn`, use the command

```
svn checkout http://github.com/mathjax/MathJax/trunk MathJax
```

to obtain and set up a copy of MathJax. (Note that there is no longer a `fonts.zip` file as of v1.1, and that the `fonts` directory is now part of the repository itself.)

Whenever you want to update MathJax, you can now use

```
cd MathJax
svn status -u
```

to check if there are updates to MathJax. If MathJax needs updating, use

```
cd MathJax
svn update
```

to update your copy of MathJax to the current release version. If you keep MathJax updated in this way, you will be sure that you have the latest bug fixes and new features as they become available.

This gets you the current development copy of MathJax, which is the version that contains all the latest changes to MathJax. Although we try to make sure this version is a stable and usable version of MathJax, it is under active development, and at times it may be less stable than the "release" version. If you prefer to use one of the tagged releases instead, then use

```
svn checkout https://github.com/mathjax/MathJax/branches/[name] MathJax
```

where `[name]` is replaced by the name of the branch you want to check out; e.g., `2.1-latest`. The branch names can be found on the GitHub MathJax page under the branches tab.

Obtaining MathJax via an archive

Release versions of MathJax are available in archive files from the MathJax download page or the MathJax GitHub page (via the "zip" button, or the "downloads" tab), where you can download the archive that you need.

You should download the v2.1 archive (which will get you a file with a name like `mathjax-MathJax-v2.1-X-XXXXXXXX.zip`, where the X's are some sequence of random-looking letters and numbers), then simply unzip it. Once the MathJax directory is unpacked, you should move it to the desired location on your server (or your hard disk, if you are using it locally rather then through a web server). One natural location is to put it at the top level of your web server's hierarchy. That would let you refer to the main MathJax file as `/MathJax/MathJax.js` from within any page on your server.

From the MathJax GitHub download link, you can also select the `Download .tar.gz` or `Download .zip` buttons to get a copy of the current development version of MathJax that contains all the latest changes and bug-fixes.

If a packaged release receives any important updates, then those updates will be part of the *branch* for that version. The link to the `.zip` file in the download list will be the original release version, not the patched version. To obtain the patched version, use the *Branches* drop down menu (at the far left of the menus within the page) to select the release branch that you want (for example `v2.1-latest`), and then use the "zip" button just above it to get the latest patched version of that release.

Obtaining MathJax via Bower

Starting with version 2.3, it is possible to use Bower to install MathJax. Assuming Bower is installed on your system, just execute the following command:

```
bower install MathJax
```

Testing your installation

Use the HTML files in the `test` directory to see if your installation is working properly:

```
test/
    index.html          # Tests default configuration
    index-images.html   # Tests image-font fallback display
    sample.html         # Sample page with lots of pretty equations
    examples.html       # Page with links to all sample pages
```

Open these files in your browser to see that they appear to be working properly. If you have installed MathJax on a server, use the web address for those files rather than opening them locally. When you view the `index.html` file, you should see (after a few moments) a message that MathJax appears to be working. If not, you should check that the files have been transferred to the server completely, and that the permissions allow the server to access the files and folders that are part of the MathJax directory (be sure to verify the MathJax folder's permissions as well). Checking the server logs may help locate problems with the installation.

Notes about shared installations

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 — Firefox's and IE9's same-origin security policy for cross-domain scripting.

Firefox's interpretation of the same-origin policy is more strict than most other browsers, and it affects how fonts are loaded with the *@font-face* CSS directive. MathJax uses this directive to load web-based math fonts into a page when the user doesn't have them installed locally on their own computer. Firefox's security policy, however, only allows 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, Firefox won't be able to access those web fonts. In this case, MathJax will pause while waiting for the font to download (which will never happen); it will time out after about 5 seconds and switch to image fonts as a fallback. Similarly, IE9 has a similar same-origin policy in its *IE9 standards mode*, so it exhibits this same behavior.

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 server. In the remote server's `MathJax/fonts/` folder, create a file called `.htaccess` that contains the following lines:

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

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

This file should make it possible for pages at other sites to load MathJax from this server in such a way that Firefox and IE9 will be able to download the web-based fonts. If you want to restrict the sites that can access the web fonts, change the `Access-Control-Allow-Origin` line to something like:

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

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

Firefox and local fonts

Firefox's same-origin security policy affects its ability to load web-based fonts, as described above. This has implications not only to cross-domain loading of MathJax, but also to using MathJax locally from your hard disk. Firefox's interpretation of the same-origin policy for local files is that the "same domain" for a page is the directory where that page exists, or any of its subdirectories. So if you use MathJax in a page with a `file://` URL, and if MathJax is loaded from a directory other than the one containing the original page, then MathJax will not be able to access the web-based fonts in Firefox. In that case, MathJax will fall back on image fonts to display the mathematics.

In order for Firefox to be able to load the fonts properly for a local file, your MathJax installation must be in a subdirectory of the one containing the page that uses MathJax. This is an unfortunate restriction, but it is a limitiation imposed by Firefox's security model that MathJax can not circumvent. Currently, this is not a problem for other browsers.

One solution to this problem is to install the MathJax fonts locally, so that Firefox will not have to use web-based fonts in the first place. To do that, either install the STIX fonts, or copy the fonts from `MathJax/fonts/HTML-CSS/TeX/otf` into your systems fonts directory and restart your browser (see the MathJax fonts help page for details).

IE9 and remote fonts

IE9's same-origin policy affects its ability to load web-based fonts, as described above. This has implications not ony to cross-domain loading of MathJax, but also to the case where you view a local page (with a `file://` URL) that accesses MathJax from a remote site such as a CDN service. In this case, IE9 does **not** honor the `Access-Control-Allow-Origin` setting of the remote server (as it would if the web page came from an `http://` URL), and so it **never** allows the font to be accessed.

One solution to this problem is to install the MathJax fonts locally so that MathJax doesn't have to use web-based fonts in the first place. Your best bet is to install the STIX fonts on your system (see the MathJax fonts help page for details).

Loading and Configuring MathJax

You load MathJax into a web page by including its main JavaScript file into the page. That is done via a `<script>` tag that links to the `MathJax.js` file. To do that, place the following line in the `<head>` section of your document:

```
<script type="text/javascript" src="path-to-MathJax/MathJax.js"></script>
```

where `path-to-MathJax` is replaced by the URL of the copy of MathJax that you are loading. For example, if you are using cdnjs as a distributed network service, the tag might be

```
<script type="text/javascript"
   src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.3.0/MathJax.js">
</script>
```

If you have installed MathJax yourself, `path-to-MathJax` will be the location of MathJax on your server, or (if you are using MathJax locally rather than through a server) the location of that directory on your hard disk. For example, if the MathJax directory is at the top level of your web server's directory hierarchy, you might use

```
<script type="text/javascript" src="/MathJax/MathJax.js"></script>
```

to load MathJax.

If you install MathJax on a server in a domain that is different from the one containing the page that will load MathJax, then there are issues involved in doing so that you need to take into consideration. See the *Notes About Shared Servers* for more details.

When you load MathJax, it is common to request a specific configuration file as discussed in the section on *Using a Configuration File* below, and in more detail in the *Common Configurations* section. A typical invocation of MathJax would be

```
<script type="text/javascript"
   src="https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>
```

which loads MathJax with a configuration file that includes everything you need in order to enter mathematics in either TeX, LaTeX, or MathML notation, and produces output using MathML if the browser supports that well enough, or HTML-with-CSS otherwise. If you **don't** load an explicit configuration file, you will need to include an in-line configuration block in order to tell MathJax how to read and display the mathematics on your pages. See the section below on *Using In-line Configuration Options* for details.

It is best to load MathJax in the document's `<head>` block, but it is also possible to load MathJax into the `<body>` section, if needed. If you do this, load it as early as possible, as MathJax will begin to load its components as soon as it is included in the page, and that will help speed up the processing of the mathematics on your page. MathJax does expect there to be a `<head>` section to the document, however, so be sure there is one if you are loading MathJax in the `<body>`.

It is also possible to load MathJax dynamically after the page has been prepared, for example, via a GreaseMonkey script, or using a specially prepared bookmarklet. This is an advanced topic, however; see *Loading MathJax Dynamically* for more details.

Loading MathJax from a CDN

MathJax is available as a web service from various free CDN providers, so you can obtain MathJax from there without needing to install it on your own server.

Warning: We retired our self-hosted CDN at *cdn.mathjax.org* in April, 2017. We recommend using cdnjs.com which uses the same provider. The use of `cdn.mathjax.org` was governed by its terms of service.

A CDN is part of a distributed "cloud" network, so it is handled by servers around the world. That means that you should get access to a server geographically near you, for a fast, reliable connection.

Most CDN services offer several versions of MathJax. For example, *cdnjs* hosts all tagged versions since v1.1 so you can link to the version you prefer.

Note: There is currently no provider who offers a rolling release link, i.e, a link that updates to each newer version of MathJax upon release.

The URL that you use to obtain MathJax determines the version that you get. For example, *cdnjs* uses a URL that includes the version tag so you can load the current version via

```
https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.3.0/MathJax.js   # the 2.3.0 release
```

Pre-releases are also available on *cdnjs*.

Note: If you wish to use the development version of MathJax, you will need to install your own copy; see *Installing and Testing MathJax* for information on how to do that.

If you wish to use a CDN but use your own configuration file rather than one of the pre-defined ones, see the information at the end of the *Using a Local Configuration File* section below.

Configuring MathJax

There are two ways to configure MathJax: via a configuration file, or by including configuration commands within the web page itself. These can be used independently, or in combination. For example, you can load a main pre-defined configuration file, but include in-line commands to adjust the configuration to your needs.

Note that you must use at least one of these two forms of configuration. Unlike MathJax v1.0, version 1.1 and higher does not load a default configuration file. If you have been using version 1.0's `config/MathJax.js` for your configuration, you will need to load that configuration file explicitly via a `config` parameter, as described below.

Using a configuration file

The first way to configure MathJax is to use a configuration file. MathJax comes with a number of pre-defined configuration files, which are stored in the `MathJax/config` directory. Among these are the following

`default.js`
: A file that contains nearly all the configuration options with comments describing them, which you can edit to suit your needs.

TeX-AMS-MML_HTMLorMML.js
: Allows math to be specified in *TeX*, *LaTeX*, or *MathML* notation, with the *AMSmath* and *AMSsymbols* packages included, producing output using MathML if the browser supports it sufficiently, and HTML-with-CSS otherwise.

TeX-AMS_HTML.js
: Allows math to be specified in *TeX* or *LaTeX* notation, with the *AMSmath* and *AMSsymbols* packages included, and produces output using the HTML-CSS output processor.

MML_HTMLorMML.js
: Allows math to be specified using *MathML* notation, and produces MathML output if the browser supports it sufficiently, or HTML-CSS output otherwise.

AM_HTMLorMML.js
: Allows math to be specified using *AsciiMath* notation, producing output in MathML if the browser supports it sufficiently, or as HTML-with-CSS otherwise.

TeX-AMS-MML_SVG.js
: Allows math to be specified in *TeX*, *LaTeX*, or *MathML* notation, with the *AMSmath* and *AMSsymbols* packages included, producing output using SVG.

TeX-MML-AM_HTMLorMML.js
: Allows math to be specified in *TeX*, *LaTeX*, *MathML*, or *AsciiMath* notation, with the *AMSmath* and *AMSsymbols* packages included, producing output using MathML if the browser supports it sufficiently, and HTML-with-CSS otherwise.

The first of these is a file that you can edit to suit your needs. It contains nearly all the configuration options that MathJax allows, and has comments explaining them. The others are what are called *combined configuration files*, which not only configure MathJax, but also pre-load the various files that the configuration requires. (The contents of these files are explained in more detail in the *Common Configurations* section.)

Usually, MathJax loads its components only when they are needed, but each component will require a separate file to be loaded, and that can cause delays before the mathematics is displayed. The combined configuration files load the majority of the needed files all as one large file, reducing the number of network requests that are needed. That means you will probably be getting the components that MathJax needs faster than you would without the combined file, but you may be loading components that are never actually used; that is the trade off.

Each of the combined configuration files comes in two flavors: the ones listed above, which only configure the output processors but don't include the main code, and a "full" version, that also includes the complete output processors. For example, with `TeX-AMS_HTML.js` and `TeX-AMS_HTML-full.js`, the latter includes the complete HTML-CSS output processor. The "full" configuration files are substantially larger (on the order of 70KB more), so you need to decide whether it is worth loading the full configuration for your pages.

If most of your pages include mathematics, then it is to your advantage to load the full version, but if you are including MathJax in a theme file for a blog or wiki that only includes mathematics occasionally, then perhaps it is better to use the standard configuration instead, in which case the output processors are only loaded when they are actually needed, saving the loading of 70KB for pages that don't. Of course, if your server is configured to compress the files it sends, the difference between the two is considerably reduced. Furthermore, most browsers will cache the javascript they receive, so the download cost should only occur on the first page a user views, so it may be best to use the "full" version after all. Note, however, that mobile devices sometimes have limits on the size of files that they cache, so they may be forced to download the configuration on every page. You need to keep these issues in mind as you decide on which configuration to use.

To load a configuration file, use `config=filename` (where `filename` is one of the names above without the `.js`) as a parameter to the URL of the `MathJax.js` file. For example

```
<script type="text/javascript"
   src="https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>
```

loads the `config/TeX-AMS-MML_HTMLorMML.js` configuration file from the MathJax distributed network service.

You can include more than one configuration file by separating them with commas. For example, if you have a locally defined configuration file called `MathJax/config/local/local.js` that modifies the settings for the `TeX-AMS_HML` configuration, defines some new TeX macros, and so on, you can use

```
<script type="text/javascript"
   src="path-to-MathJax/MathJax.js?config=TeX-AMS_HTML,local/local">
</script>
```

to first load the main configuration, then the local modifications.

Using a local configuration file with a CDN

You can load MathJax from a CDN server but still use a configuration from your own local server. For example, suppose you have a configuration file called `local.js` on your own server, in a directory called `MathJax/config/local`. Then you can load MathJax from a CDN and still use your configuration file as follows:

```
<script type="text/javascript"
   src="https://example.com/MathJax.js?config=TeX-AMS_HTML,http://myserver.com/
↪MathJax/config/local/local.js">
</script>
```

Because the `local.js` file is not on a CDN server, you must give the complete URL to the local configuration file. Note that you also have to edit the *`loadComplete()`* call that is at the bottom of the configuration file to change it from `[MathJax]/config/local/local.js` to the complete URL as you give it in the `config` parameter. In the example above, it would be

```
MathJax.Ajax.loadComplete("http://myserver.com/MathJax/config/local/local.js");
```

That is because the `[MathJax]` in the original URL refers to the root directory where `MathJax.js` was loaded, which is on a CDN, not your local server, and so you need to tell MathJax the actual location of your configuration file.

Using in-line configuration options

The second way to configure MathJax is through *in-line configuration*, which puts the configuration options within the web page itself. The use of in-line configuration with MathJax requires two separate `<script>` tags: one for specifying the configuration settings and one for loading of MathJax. Because MathJax starts its configuration process as soon as it is loaded, the configuration script must come **before** the script tag that loads `MathJax.js` itself. You do this by including a `<script>` with `type="text/x-mathjax-config"` whose content will be run when MathJax performs its configuration. Generally, this script will include a `MathJax.Hub.Config()` call to perform MathJax configuration, but it can also include other MathJax commands, such as registering signal actions, or any JavaScript commands that you want. You can have as many such script tags as you need, and MathJax will process them in the order in which they appear in the document.

For instance,

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    extensions: ["tex2jax.js"],
    jax: ["input/TeX", "output/HTML-CSS"],
    tex2jax: {
      inlineMath: [ ['$','$'], ["\\(","\\)"] ],
```

```
      displayMath: [ ['$$','$$'], ["\\[","\\]"] ],
      processEscapes: true
    },
    "HTML-CSS": { availableFonts: ["TeX"] }
  });
</script>
<script type="text/javascript" src="path-to-MathJax/MathJax.js">
```

This example includes the *tex2jax* preprocessor and configures it to use both the standard *TeX* and *LaTeX* math delimiters. It uses the *TeX* input processor and the *HTML-CSS* output processor, and forces the HTML-CSS processor to use the TeX fonts rather than other locally installed fonts (e.g., *STIX* fonts). See the *configuration options* section (or the comments in the `config/default.js` file) for more information about the configuration options that you can include in the `MathJax.Hub.Config()` call. This configuration does **not** load any pre-defined configuration file.

Note that you can combine in-line configuration with file-based configuration; simply include `text/x-mathjax-config` scripts as above, but also include `config=filename` when you load the `MathJax.js` file. For example, the *tex2jax* preprocessor does **not** enable the TeX single-dollar in-line math delimiters by default. You can load one of the pre-defined configuration files that includes the TeX preprocessor, and use an in-line configuration block to enable the single-dollar signs, as in this example:

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    tex2jax: {
      inlineMath: [ ['$','$'], ["\\(","\\)"] ],
      processEscapes: true
    }
  });
</script>
<script type="text/javascript" src="path-to-MathJax/MathJax.js?config=TeX-AMS_HTML">
</script>
```

Starting with MathJax version 2.3, it is possible to set `window.MathJax` to a configuration object in any Javascript code before MathJax's startup. MathJax will then use that object for its initial configuration. For instance the previous example becomes:

```
<script type="text/javascript">
  window.MathJax = {
    tex2jax: {
      inlineMath: [ ['$','$'], ["\\(","\\)"] ],
      processEscapes: true
    }
  };
</script>
<script type="text/javascript" src="path-to-MathJax/MathJax.js?config=TeX-AMS_HTML">
</script>
```

Similarly to `text/x-mathjax-config`, you can enter arbitrary code to execute during the configuration phase. You just need to put that code in an `AuthorInit` function:

```
<script type="text/javascript">
  window.MathJax = {
    AuthorInit: function () {
      ... initialization code ...
    }
  };
</script>
```

Note that this initialization code runs before the `MathJax.Hub.queue` is set up, so if you want to queue additional actions during the *AuthorInit* function, use

```
<script type="text/javascript">
  window.MathJax = {
    AuthorInit: function () {
      MathJax.Hub.Register.StartupHook("Begin",function () {
        MathJax.Hub.Queue(
          ... your actions here ...
        )
      });
    }
  };
</script>
```

Configuring MathJax after it is loaded

Because MathJax begins its configuration process immediately after it is loaded (so that it can start loading files as quickly as it can), the configuration blocks for MathJax must come before `MathJax.js` is loaded, so they will be available to MathJax when it starts up. There are situations, however, when you might want to put off configuring MathJax until later in the page.

One such situation is when you have a site that loads MathJax as part of a theme or template, but want to be able to modify the configuration on specific pages of the site. To accomplish this, you need to ask MathJax to delay its startup configuration until some later time. MathJax uses the `delayStartupUntil` parameter to control the timing of the startup sequence. By default, it is set to `none`, meaning there is no delay and MathJax starts configuration right away.

You can set `delayStartupUntil=onload` in order to prevent MathJax from continuing its startup process until the page's onLoad handler fires. This allows MathJax to find the `text/x-mathjax-config` blocks that occur anywhere on the page, not just the ones that appear above the `<script>` that loads `MathJax.js`. It also means that MathJax will not begin loading any of the files that it needs until then as well, which may delay the displaying of your mathematics, since the onLoad handler doesn't execute until all the images and other media are available. (If you have used a combined configuration file, however, it already includes all the main files that MathJax needs, so there is not much loss in delaying the startup.)

You can set `delayStartupUntil=configured` in order to delay the startup configuration until the `MathJax.Hub.Configured()` method is called. This allows you to delay startup until later on the page, but then restart the MathJax configuration process as soon as possible rather than waiting for the entire page to load. For example, you could use

```
<script type="text/javascript"
   src="path-to-MathJax/MathJax.js?config=TeX-AMS-MML_HTMLorMML&
↪delayStartupUntil=configured">
</script>
```

in your theme's header file, and

```
<script type="text/javascript">
  MathJax.Hub.Configured()
</script>
```

in its footer, so that MathJax will delay setting up until the footer is reached, but will not have to wait until images and other files are loaded. In this way, if you have `text/x-mathjax-config` script tags within the main body of the document, MathJax will read and process those before continuing its startup. In this way you can use a default configuration that can be modified on a page-by-page basis.

Note that `MathJax.Hub.Configured()` is not called by MathJax; you must make that call somewhere within the page yourself after the configuration blocks are set up. If you do not execute this function, MathJax will not process any of the math on the page.

Details of the MathJax configuration process

Since there are a number of different ways to configure MathJax, it is important to know how they interact. The configuration actions are the following:

1. Execute `AuthorInit()` from in-line `MathJax = {...}`.
2. Process any configuration file explicitly specified as a script parameter via *config*=.
3. Perform author configuration from in-line `MathJax = {...}`
4. Process the in-line script body (deprecated), if present.
5. If delayed startup is requested, wait for the indicated signal.
6. Process `text/x-mathjax-config` config blocks.
7. Process any config files queued in the configuration's *config* array by earlier config code.

Note that `text/x-mathjax-config` script blocks must either precede the `MathJax.js` script element, or you must request a delayed startup. Otherwise, blocks that follow the `MathJax.js` script element may or may not be available when MathJax runs, and browser-dependent erratic behavior will result. Similarly, `window.MathJax` must be created before `MathJax.js` is loaded. If you set the `MathJax` variable afterward, you will disable MathJax entirely!

Common Configurations

MathJax comes with a number of pre-defined configuration files in the `MathJax/config` directory. The `default.js` file contains nearly all the possible configuration options together with comments explaining them, so you can use that file to customize MathJax to your needs. Simply load it via

```
<script type="text/javascript" src="path-to-MathJax/MathJax.js?config=default"></
↪script>
```

where `path-to-MathJax` is the URL to the MathJax directory on your server or hard disk. If you are using MathJax from a CDN, you can view the contents of default.js as a reference, but you will not be able to edit a CDN copy. It is possible to use a CDN copy of MathJax with your own configuration file, however; see *Using a Local Configuration File with a CDN* for details.

The remaining files in the MathJax/config directory are combined configuration files that include not just configuration parameters but also the files that MathJax would need to load for those configurations. This means MathJax will have to load fewer files, and since each file access requires establishing connections over the network, it can be faster to load one larger file than several smaller ones. See *Loading and Configuring MathJax* for more details about how to load configurations, and how to modify the parameters for a configuration file.

The following sections describe the contents of the combined configuration files. Each comes in two flavors: a standard version and a "full" version. The standard version simply defines the output processor(s) that are part of the configuration, but doesn't load the code that implements the output processor. The full version loads the complete output processors, so everything that MathJax needs for the page should be loaded up front, and there will be no delay once the page is ready to be processed. To obtain the "full" version, add `-full` to the end of the configuration file name.

The `TeX-MML-AM_HTMLorMML` configuration file

This configuration file is the most general of the pre-defined configurations. It loads all the main MathJax components, including the TeX, MathML, and AsciiMath preprocessors and input processors, the AMSmath, AMSsymbols, noErrors, and noUndefined TeX extensions, both the native MathML and HTML-with-CSS output processor definitions, and the MathMenu and MathZoom extensions. It is equivalent to the following configuration:

```
MathJax.Hub.Config({
  config: ["MMLorHTML.js"],
  jax: ["input/TeX","input/MathML","input/AsciiMath","output/HTML-CSS","output/
↪NativeMML"],
  extensions: ["tex2jax.js","mml2jax.js","asciimath2jax.js","MathMenu.js","MathZoom.js
↪"],
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  }
});
```

In addition, it loads the mml Element Jax, the TeX, MathML, and AsciiMath input jax main code (not just the definition files), as well as the *toMathML* extension, which is used by the Show Source option in the MathJax contextual menu. The *-full* version also loads both the HTML-CSS and NativeMML output jax main code, plus the HTML-CSS *mtable* extension, which is normally loaded on demand.

See the *tex2jax configuration* section for other configuration options for the `tex2jax` preprocessor, and the *TeX input jax configuration* section for options that control the TeX input processor. See the *mml2jax configuration* section for other configuration options for the `mml2jax` preprocessor, and the *MathML input jax configuration* section for options that control the MathML input processor. See the *asciimath2jax configuration* section for other configuration options for the `asciimath2jax` preprocessor, and the *AsciiMath input jax configuration* section for options that control the AsciiMath input processor. See *MathJax Output Formats* for more information on the NativeMML and HTML-CSS output processors. See the *MMLorHTML configuration* section for details on the options that control the `MMLorHTML` configuration.

The `TeX-AMS-MML_HTMLorMML` configuration file

This configuration file is the most commonly used of the pre-defined configurations. It loads all the main MathJax components, including the TeX and MathML preprocessors and input processors, the AMSmath, AMSsymbols, noErrors, and noUndefined TeX extensions, both the native MathML and HTML-with-CSS output processor definitions, and the MathMenu and MathZoom extensions. It is equivalent to the following configuration:

```
MathJax.Hub.Config({
  config: ["MMLorHTML.js"],
  jax: ["input/TeX","input/MathML","output/HTML-CSS","output/NativeMML"],
  extensions: ["tex2jax.js","mml2jax.js","MathMenu.js","MathZoom.js"],
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  }
});
```

In addition, it loads the mml Element Jax, the TeX and MathML input jax main code (not just the definition files), as well as the *toMathML* extension, which is used by the Show Source option in the MathJax contextual menu. The `-full` version also loads both the HTML-CSS and NativeMML output jax main code, plus the HTML-CSS *mtable* extension, which is normally loaded on demand.

See the *tex2jax configuration* section for other configuration options for the `tex2jax` preprocessor, and the *TeX input jax configuration* section for options that control the TeX input processor. See the *mml2jax configuration* section for other configuration options for the `mml2jax` preprocessor, and the *MathML input jax configuration* section for options

that control the MathML input processor. See *MathJax Output Formats* for more information on the NativeMML and HTML-CSS output processors. See the *MMLorHTML configuration* section for details on the options that control the `MMLorHTML` configuration.

The `TeX-AMS_HTML` configuration file

This configuration file is for sites that only use TeX format for their mathematics, and that want the output to be as close to TeX output as possible. This uses the HTML-CSS output jax (even when the user's browser understands MathML). The user can still use the MathJax contextual menu to select the NativeMML output jax if they desire.

This file includes all the important MathJax components for TeX input and output, including the *tex2jax* preprocessor and TeX input jax, the AMSmath, AMSsymbols, noErrors, and noUndefined TeX extensions, the HTML-with-CSS output processor definition, and the MathMenu and MathZoom extensions. It is equivalent to the following configuration:

```
MathJax.Hub.Config({
  jax: ["input/TeX","output/HTML-CSS"],
  extensions: ["tex2jax.js","MathMenu.js","MathZoom.js"],
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  }
});
```

In addition, it loads the mml Element Jax and the TeX input jax main code (not just the definition file), as well as the *toMathML* extension, which is used by the Show Source option in the MathJax contextual menu. The `-full` version also loads the HTML-CSS output jax main code, plus the HTML-CSS *mtable* extension, which is normally loaded on demand.

See the *tex2jax configuration* section for other configuration options for the `tex2jax` preprocessor, and the *TeX input jax configuration* section for options that control the TeX input processor. See *MathJax Output Formats* for more information on the HTML-CSS output processor.

The `MML_HTMLorMML` configuration file

This configuration file is for sites that only use MathML format for their mathematics. It will use MathML output in browsers where that is supported well, and HTML-CSS output otherwise. The user can still use the MathJax contextual menu to select the other output format if they desire.

This file includes all the important MathJax components for MathML input and output, including the *mml2jax* preprocessor and MathML input jax, the NativeMML and HTML-CSS output processor definition files, and the MathMenu and MathZoom extensions. It is equivalent to the following configuration:

```
MathJax.Hub.Config({
  config: ["MMLorHTML.js"],
  jax: ["input/MathML","output/HTML-CSS","output/NativeMML"],
  extensions: ["mml2jax.js","MathMenu.js","MathZoom.js"]
});
```

In addition, it loads the mml Element Jax and the MathML input jax main code (not just the definition file), as well as the *toMathML* extension, which is used by the Show Source option in the MathJax contextual menu. The `-full` version also loads both the HTML-CSS and NativeMML output jax main code files, plus the HTML-CSS *mtable* extension, which is normally loaded on demand.

See the *mml2jax configuration* section for other configuration options for the `mml2jax` preprocessor, and the *MathML input jax configuration* section for options that control the MathML input processor. See *MathJax Output Formats* for

more information on the NativeMML and HTML-CSS output processors. See the *MMLorHTML configuration* section for details on the options that control the `MMLorHTML` configuration.

The `AM_HTMLorMML` configuration file

This configuration file is for sites that only use AsciiMath format for their mathematics. It will use MathML output in browsers where that is supported well, and HTML-CSS output otherwise. The user can still use the MathJax contextual menu to select the other output format if they desire.

This file includes all the important MathJax components for AsciiMath input and output, including the *asciimath2jax* preprocessor and AsciiMath input jax, the NativeMML and HTML-CSS output processor definition files, and the MathMenu and MathZoom extensions. It is equivalent to the following configuration:

```
MathJax.Hub.Config({
  config: ["MMLorHTML.js"],
  jax: ["input/AsciiMath","output/HTML-CSS","output/NativeMML"],
  extensions: ["asciimath2jax.js","MathMenu.js","MathZoom.js"]
});
```

In addition, it loads the mml Element Jax and the TeX input jax main code (not just the definition file), as well as the *toMathML* extension, which is used by the Show Source option in the MathJax contextual menu. The `-full` version also loads the HTML-CSS output jax main code, plus the HTML-CSS *mtable* extension, which is normally loaded on demand.

See the *asciimath2jax configuration* section for other configuration options for the `asciimath2jax` preprocessor, and the *AsciiMath input jax configuration* section for options that control the AsciiMath input processor. See *MathJax Output Formats* for more information on the HTML-CSS and NativeMML output processors. See the *MMLorHTML configuration* section for details on the options that control the `MMLorHTML` configuration.

The `TeX-AMS-MML_SVG` configuration file

This configuration file is the same as *TeX-AMS-MML_HTMLorMML* except that it uses the SVG output renderer rather than the NativeMML or HTML-CSS ones. It loads all the main MathJax components, including the TeX and MathML preprocessors and input processors, the AMSmath, AMSsymbols, noErrors, and noUndefined TeX extensions, the SVG output processor definitions, and the MathMenu and MathZoom extensions. It is equivalent to the following configuration:

```
MathJax.Hub.Config({
  jax: ["input/TeX","input/MathML","output/SVG"],
  extensions: ["tex2jax.js","mml2jax.js","MathMenu.js","MathZoom.js"],
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  }
});
```

In addition, it loads the mml Element Jax, the TeX and MathML input jax main code (not just the definition files), as well as the *toMathML* extension, which is used by the Show Source option in the MathJax contextual menu. The `-full` version also loads both the SVG output jax main code, plus the SVG *mtable* extension, which is normally loaded on demand.

See the *tex2jax configuration* section for other configuration options for the `tex2jax` preprocessor, and the *TeX input jax configuration* section for options that control the TeX input processor. See the *mml2jax configuration* section for other configuration options for the `mml2jax` preprocessor, and the *MathML input jax configuration* section for options that control the MathML input processor. See *MathJax Output Formats* for more information on the SVG output processor.

The `Accessible` configuration file

This configuration file is essentially the same as `TeX-AMS-MML_HTMLorMML` except that it includes options that are designed for assistive technology, particularly for those with visual challenges. *This file is deprecated* since the controls that make MathJax work with screen readers are now available in the MathJax contextual menu, and so there is no need to set them in the configuration file any longer. So you can use any of the other pre-defined configurations and readers with special needs should be able to change the MathJax settings themselves to be appropriate for their software.

The Accessible configuration is equivalent to the following:

```
MathJax.Hub.Config({
  config: ["MMLorHTML.js"],
  jax: ["input/TeX","input/MathML","output/HTML-CSS","output/NativeMML"],
  extensions: ["tex2jax.js","mml2jax.js","MathMenu.js","MathZoom.js"],
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  },
  menuSettings: {
    zoom: "Double-Click",
    mpContext: true,
    mpMouse: true
  },
  errorSettings: { message: ["[Math Error]"] }
});
```

This turns off the MathJax contextual menu for IE when MathPlayer is active, and passes mouse events on to MathPlayer to allow screen readers full access to MathPlayer. It also sets the zoom trigger to double-click, so that readers can see a larger version of the mathematics by double-clicking on any equation.

In addition, it loads the mml Element Jax, the TeX and MathML input jax main code (not just the definition files), as well as the *toMathML* extension, which is used by the Show Source option in the MathJax contextual menu. The `-full` version also loads both the HTML-CSS and NativeMML output jax main code, plus the HTML-CSS *mtable* extension, which is normally loaded on demand.

Using MathJax in popular web platforms

MathJax plugins are available for a growing number of wikis, blogs, and other content-management systems.

- MathJax-LaTeX, Simple-MathJax plug-ins for WordPress.
- MathJax plugin for Drupal.
- Concrete5 MathJax plugin.
- MathJax plugins for Joomla.
- Sphinx extension: MathJax
- MathJax plugin for DokuWiki
- MediaWiki math extension used on Wikipedia, using MathJax since v1.20.
- Tiddlywiki plugin, PluginMathJax for TiddlyWiki.
- WikidPad, a plugin for the personal wiki platform.
- MathJax Extension for the webbased SVG editor SVG edit.
- Instantbird Extension adds MathJax to the Mozilla-based chat client.

If the program you are using is not one of these, you might be able to use MathJax by modifying the theme or template for your wiki or blog, as explained below.

Using MathJax in a Theme File

Most web-based content-management systems include a theme or template layer that determines how the pages look, and that loads information common to all pages. Such theme files provide a way to include MathJax in your web templates in the absence of MathJax-specific plugins for the system you are using. To take advantage of this approach, you will need access to your theme files, which probably means you need to be an administrator for the site; if you are not, you may need to have an administrator do these steps for you. You will also have to identify the right file if the theme consists of multiple files.

To enable MathJax in your web platform, add the line:

```
<script type="text/javascript"
   src="https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML"></script>
```

either just before the `</head>` tag in your theme file, or at the end of the file if it contains no `</head>`.

Keep in mind that this will enable MathJax for your current theme/template only. If you change themes or update your theme, you will have to repeat these steps. We strongly suggest to use a plugin or help the community of your favorite software by writing a plugin.

MathJax TeX and LaTeX Support

The support for *TeX* and *LaTeX* in MathJax consists of two parts: the *tex2jax* preprocessor, and the *TeX* input processor. The first of these looks for mathematics within your web page (indicated by math delimiters like `$$...$$`) and marks the mathematics for later processing by MathJax. The TeX input processor 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.

The *tex2jax* preprocessor can be configured to look for whatever markers you want to use for your math delimiters. See the *tex2jax configuration options* section for details on how to customize the action of *tex2jax*.

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 has few configuration options (see the *TeX options* section for details), but it can also be customized through the use of extensions that define additional functionality (see the *TeX and LaTeX extensions* below).

Note that 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; it only handles the mathematics. So, for example, MathJax does not implement `\emph` or `\begin{enumerate}...\end{enumerate}` or other text-mode macros or environments. You must use HTML to handle such formatting tasks. If you need a LaTeX-to-HTML converter, you should consider other options.

TeX and LaTeX math delimiters

By default, the *tex2jax* preprocessor defines the LaTeX math delimiters, which are `\(...\)` for in-line math, and `\[...\]` for displayed equations. It also defines 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-dollars for in-line math mode, you must enable that explicitly in your configuration:

```
MathJax.Hub.Config({
  tex2jax: {
    inlineMath: [['$','$'], ['\\(','\\)']],
    processEscapes: true
  }
});
```

Note that if you do this, you may want to also set `processEscapes` to `true`, as in the example above, so that you can use `\$` to prevent a dollar sign from being treated as a math delimiter within the text of your web page. (Note that within TeX mathematics, `\$` always has this meaning; `processEscapes` only affects the treatment of the *opening* math delimiter.)

See the `config/default.js` file, or the *tex2jax configuration options* page, for additional configuration parameters that you can specify for the *tex2jax* preprocessor, which is the component of MathJax that identifies TeX notation within the page.

TeX and LaTeX in HTML documents

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>`) as TeX-formatted math does not include HTML tags. Also, since the mathematics is initially given as text on 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 the browsers. For example,

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

will cause a problem, because the brower 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 "`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 to simply put spaces around these symbols to cause the browser to avoid them, so

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

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

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

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 ...
```

Keep in mind that the browser interprets your text before MathJax does.

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 you 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 will convert your subscripts markers into italics (inserting `<i>` tags into your mathematics, which will cause MathJax to ignore the math).

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

If there is no plugin for your system, or if it doesn't handle the subtleties of issolating the mathematics from the other markup that it supports, then you may have to "trick" it 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 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.

Some content-management systems use the backslash (\) as a special character for "escaping" other characters, but TeX uses this character to indicate a macro name. 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*. Note in particular that if you want `\\` you will have to double *both* backslashes, giving `\\\\`.

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, then you can use `\$` in the text of your page to get a dollar sign (without the backslash) in the end. Alternatively, you use something like `<span>$</span>` to isolate the dollar sign so that MathJax will not use it as a delimiter.

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 these, they must be enclosed in math delimiters (since MathJax only processes macros in math-mode). For example

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

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

You can include macro definitions in the *Macros* section of the *TeX* blocks of your configuration, but they must be represented as JavaScript objects. For example, the two macros above can be pre-defined in the configuraiton by

```
MathJax.Hub.Config({
  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.

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.

If you have many such definitions that you want to use on more than one page, you could put them into a configuration file that you can load along with the main configuration file. For example, you could create a file in `MathJax/config/local` called `local.js` that contains your macro definitions:

```
MathJax.Hub.Config({
  TeX: {
    Macros: {
      RR: "{\\bf R}",
      bold: ["{\\bf #1}",1]
    }
  }
});

MathJax.Ajax.loadComplete("[MathJax]/config/local/local.js");
```

and then load it along with your main configuration file on the script that loads `MathJax.js`:

```
<script src="/MathJax/MathJax.js?config=TeX-AMS_HTML,local/local.js"></script>
```

If you are using a CDN, you can make a local configuration file on your own server, and load MathJax itself from a CDN and your configuration file from your server. See *Using a Local Configuration File with a CDN* for details.

Automatic Equation Numbering

New in MathJax v2.0 is the ability to have equations be numbered automatically. This functionality is turned off by default, so that pages don't change when you update from v1.1 to v2.0, but it is easy to configure MathJax to produce automatic equation numbers by adding:

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  TeX: { equationNumbers: { autoNumber: "AMS" } }
});
</script>
```

to your page just before the `<script>` tag that loads `MathJax.js` itself.

Equations can be numbered in two ways: either number the AMSmath environments as LaTeX would, or number all displayed equations (the example above uses AMS-style numbering). Set `autoNumber` to `"all"` if you want every displayed equation to be numbered. 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.

Note that the AMS environments come in two forms: starred and unstarred. The unstarred versions produce equation numbers (when `autoNumber` 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*}
```

won't be numbered (when `autoNumber` 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. See the equation numbering links in the MathJax examples page for more examples.

You can configure the way that numbers are displayed and how the references to them are made using parameters in the `equationNumbers` block of your `TeX` configuration. See the *TeX configuration options* page for more details.

TeX and LaTeX extensions

While MathJax includes nearly all of the Plain TeX math macros, and many of the LaTeX macros and environments, not everything is implemented in the core TeX input processor. Some less-used commands are defined in extensions to the TeX processor. MathJax will load some extensions automatically when you first use the commands they implement (for example, the `\def` and `\newcommand` macros are implemented in the `newcommand.js` extension, but MathJax loads this extension itself when you use those macros). Not all extensions are set up to load automatically, however, so you may need to request some extensions explicitly yourself.

To enable any of the TeX extensions, simply add the appropriate string (e.g., `"AMSmath.js"`) to the *extensions* array in the `TeX` block of your configuration. If you use one of the combined configuration files, like `TeX-AMS_HTML`, this will already include several of the extensions automatically, but you can include others using a mathjax configuration script prior to loading MathJax. For example

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({ TeX: { extensions: ["autobold.js"] }});
</script>
<script type="text/javascript"
    src="https://example.com/MathJax.js?config=TeX-AMS_HTML">
</script>
```

will load the *autobold* TeX extension in addition to those already included in the `TeX-AMS_HTML` configuration file.

You can also load these extensions from within a math expresion 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).

It is also possible to create a macro that will autoload an extension when it is first used (under the assumption that the extension will redefine it to perform its true function). For example

```
<script type="text/x-mathjax-config">
MathJax.Hub.Register.StartupHook("TeX Jax Ready",function () {
  MathJax.Hub.Insert(MathJax.InputJax.TeX.Definitions.macros,{
    cancel: ["Extension","cancel"],
    bcancel: ["Extension","cancel"],
    xcancel: ["Extension","cancel"],
    cancelto: ["Extension","cancel"]
  });
});
</script>
```

would declare the `\cancel`, `\bcancel`, `\xcancel`, and `\cancelto` macros to load the *cancel* extension (where they are actually defined). Whichever is used first will cause the extension to be loaded, redefining all four to their proper values. Note that this may be better than loading the extension explicitly, since it avoids loading the extra file on pages where these macros are *not* used. The sample autoloading macros example page shows this in action. The *autoload-all* extension below defines such macros for *all* the extensions so that if you include it, MathJax will have access to all the macros it knows about.

The main extensions are described below.

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` (in text mode) 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.

To use this extension in your own configurations, add it to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["action.js"]
}
```

This extension is **not** included in any of the combined configurations, and will not be loaded automatically, so you must include it explicitly in your configuration if you wish to use these commands.

AMSmath and AMSsymbols

The *AMSmath* extension implements AMS math environments and macros, and the *AMSsymbols* extension implements macros for accessing the AMS symbol fonts. These are already included in the combined configuration files that load the TeX input processor. To use these extensions in your own configurations, add them to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["AMSmath.js", "AMSsymbols.js", ...]
}
```

See the list of control sequences at the end of this document for details about what commands are implemented in these extensions.

If you are not using one of the combined configuration files, the *AMSmath* extension will be loaded automatically when you first use one of the math environments it defines, but you will have to load it explicitly if you want to use the other macros that it defines. The *AMSsymbols* extension is not loaded automatically, so you must include it explicitly if you want to use the macros it defines.

Both extensions are included in all the combined configuration files that load the TeX input processor.

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.

To use this extension in your own configurations, add it to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["AMScd.js"]
}
```

Alternatively, if the extension hasn't been loaded in the configuration, you can use `\require{AMScd}` to load it from within a TeX expression. Note that you only need to include this once on the page, not every time the *CD* environment is used.

This extension is **not** included in any of the combined configurations, and will not be loaded automatically, so you must include it explicitly in your configuration if you wish to use these commands.

Autobold

The *autobold* extension adds `\boldsymbol{...}` around mathematics that appears in a section of an HTML page that is in bold.

```
TeX: {
  extensions: ["autobold.js"]
}
```

This extension is **not** loaded by the combined configuration files.

BBox

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

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

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

Here are some examples:

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

This extension is **not** included in any of the combined configurations, but it will be loaded automatically, so you do not need to include it in your *extensions* array.

Begingroup

The *begingroup* extension implements commands that provide a mechanism for localizing macro defintions 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.

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 core TeX input jax.)

To use this extension in your own configurations, add it to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["begingroup.js"]
}
```

This extension is **not** included in any of the combined configurations, and will not be loaded automatically, so you must include it explicitly in your configuration if you wish to use these commands.

Cancel

The *cancel* extension defines the following macros:

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

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

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

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

To use this extension in your own configurations, add it to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["cancel.js"]
}
```

This extension is **not** included in any of the combined configurations, and will not be loaded automatically, so you must include it explicitly in your configuration if you wish to use these commands.

Color

The `\color` command in the core TeX input jax is not standard in that it takes the mathematics to be colored as one of its parameters, whereas the LaTeX `\color` command is a switch that changes the color of everything that follows it.

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

To use this extension in your own configurations, add it to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["color.js"]
}
```

This extension is **not** included in any of the combined configurations, and will not be loaded automatically, so you must include it explicitly in your configuration if you wish to use these commands, and have `\color` be compatible with LaTeX usage.

Enclose

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

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

For example

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

To use this extension in your own configurations, add it to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["enclose.js"]
}
```

This extension is **not** included in any of the combined configurations, and will not be loaded automatically, so you must include it explicitly in your configuration if you wish to use these commands.

Extpfeil

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

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

For example

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

defines an extensible right harpoon with barb up. Note that MathJax knows how to stretch only a limited number of characters, so you may not actually get a stretchy character this way.

To use this extension in your own configurations, add it to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["extpfeil.js"]
}
```

This extension is **not** included in any of the combined configurations, and will not be loaded automatically, so you must include it explicitly in your configuration if you wish to use these commands.

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`.

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

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

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

For example:

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

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

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

This extension is **not** included in any of the combined configurations, but it will be loaded automatically when any of these macros is used, so you do not need to include it explicitly in your configuration.

mhchem

The *mhchem* extensions implements the `\ce`, `\cf`, and `\cee` chemical equation macros of the LaTeX *mhchem* package. See the mhchem CPAN page for more information and a link to the documentation for *mhchem*.

For example

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

To use this extension in your own configurations, add it to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["mhchem.js"]
}
```

This extension is **not** included in any of the combined configurations, and will not be loaded automatically, so you must include it explicitly in your configuration if you wish to use these commands.

noErrors

The *noErrors* extension prevents TeX error messages from being displayed and shows the original TeX code instead. You can configure whether the dollar signs are shown or not for in-line math, and whether to put all the TeX on one line or use multiple lines (if the original text contained line breaks).

This extension is loaded by all the combined configuration files that include the TeX input processor. To enable the *noErrors* extension in your own configuration, or to modify its parameters, add something like the following to your `MathJax.Hub.Config()` call:

```
TeX: {
  extensions: ["noErrors.js"],
  noErrors: {
    inlineDelimiters: ["",""],    // or ["$","$"] or ["\\(","\\)"]
    multiLine: true,              // false for TeX on all one line
    style: {
      "font-size":   "90%",
      "text-align":  "left",
      "color":       "black",
      "padding":     "1px 3px",
      "border":      "1px solid"
      // add any additional CSS styles that you want
      //  (be sure there is no extra comma at the end of the last item)
    }
  }
}
```

Display-style math is always shown in multi-line format, and without delimiters, as it will already be set off in its own centered paragraph, like standard display mathematics.

The default settings place the invalid TeX in a multi-line box with a black border. If you want it to look as though the TeX is just part of the paragraph, use

```
TeX: {
  noErrors: {
    inlineDelimiters: ["$","$"],    // or ["",""] or ["\\(","\\)"]
    multiLine: false,
    style: {
      "font-size": "normal",
      "border": ""
    }
  }
}
```

You may also wish to set the font family or other CSS values here.

If you are using a combined configuration file that loads the TeX input processor, it will also load the *noErrors* extension automatically. If you want to disable the *noErrors* extension so that you receive the normal TeX error messages, use the following configuration:

```
TeX: { noErrors: { disabled: true } }
```

Any math that includes errors will be replaced by an error message indicating what went wrong.

noUndefined

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

This extension is loaded by all the combined configuration files that include the TeX input processor. To enable the *noUndefined* extension in your own configuration, or to modify its parameters, add something like the following to your `MathJax.Hub.Config()` call:

```
TeX: {
  extensions: ["noUndefined.js"],
  noUndefined: {
    attributes: {
      mathcolor: "red",
      mathbackground: "#FFEEEE",
      mathsize: "90%"
    }
  }
}
```

The `attributes` setting specifies attributes to apply to the `mtext` element that encodes the name of the undefined macro. The default values set `mathcolor` to `"red"`, but do not set any other attributes. This example sets the background to a light pink, and reduces the font size slightly.

If you are using a combined configuration file that loads the TeX input processor, it will also load the *noUndefined* extension automatically. If you want to disable the *noUndefined* extension so that you receive the normal TeX error messages for undefined macros, use the following configuration:

```
TeX: { noUndefined: { disabled: true } }
```

Any math that includes an undefined control sequence name will be replaced by an error message indicating what name was undefined.

Unicode support

The *unicode* extension implements a `\unicode{}` extension to TeX that allows arbitrary unicode code points to be entered in your mathematics. You can specify the height and depth of the character (the width is determined by the browser), and the default font from which to take the character.

Examples:

```
\unicode{65}                          % the character 'A'
\unicode{x41}                         % the character 'A'
\unicode[.55,0.05]{x22D6}             % less-than with dot, with height .55em and depth␣
↪0.05em
```

```
\unicode[.55,0.05][Geramond]{x22D6} % same taken from Geramond font
\unicode[Garamond]{x22D6}           % same, but with default height, depth of .8em,.
↪2em
```

Once a size and font are provided for a given unicode point, they need not be specified again in subsequent `\unicode{}` calls for that character.

The result of `\unicode{...}` will have TeX class *ORD* (i.e., it will act like a variable). Use `\mathbin{...}`, `\mathrel{...}`, etc., to specify a different class.

Note that a font list can be given in the `\unicode{}` macro, but Internet Explorer has a buggy implementation of the `font-family` CSS attribute where it only looks in the first font in the list that is actually installed on the system, and if the required glyph is not in that font, it does not look at later fonts, but goes directly to the default font as set in the *Internet-Options/Font* panel. For this reason, the default font list for the `\unicode{}` macro is `STIXGeneral, 'Arial Unicode MS'`, so if the user has *STIX* fonts, the symbol will be taken from that (almost all the symbols are in *STIXGeneral*), otherwise MathJax tries *Arial Unicode MS*.

The *unicode* extension is loaded automatically when you first use the `\unicode{}` macro, so you do not need to add it to the *extensions* array. You can configure the extension as follows:

```
TeX: {
  unicode: {
    fonts: "STIXGeneral, 'Arial Unicode MS'"
  }
}
```

Autoload-all

The *autoload-all* extension predefines all the macros from the extensions above so that they autoload the extensions when first used. A number of macros already do this, e.g., `\unicode`, but this extension defines the others to do the same. That way MathJax will have access to all the macros that it knows about.

To use this extension in your own configurations, add it to the *extensions* array in the TeX block.

```
TeX: {
  extensions: ["autoload-all.js"]
}
```

This extension is **not** included in any of the combined configurations, and will not be loaded automatically, so you must include it explicitly in your configuration if you wish to use these commands.

Note that *autoload-all* redefines `\color` to be the one from the *color* extension (the LaTeX-compatible one rather than the non-standard MathJax version). This is because `\colorbox` and `\fcolorbox` autoload the *color* extension, which will cause `\color` to be redefined, and so for consistency, `\color` is redefined immediately.

If you wish to retain the original definition of `\color`, then use the following

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  TeX: { extensions: ["autoload-all.js"] }
});
MathJax.Hub.Register.StartupHook("TeX autoload-all Ready", function () {
  var MACROS = MathJax.InputJax.TeX.Definitions.macros;
  MACROS.color = "Color";
  delete MACROS.colorbox;
  delete MACROS.fcolorbox;
```

```
});
</script>
```

Supported LaTeX commands

This is a long list of the TeX macros supported by MathJax. If the macro is defined in an extension, the name of the extension follows the macro name. If the extension is in brackets, the extension will be loaded automatically when the macro or environment is first used.

More complete details about how to use these macros, with examples and explanations, is available at Carol Fisher's TeX Commands Available in MathJax page.

Symbols

```
#
%
&
^

_
{
}
~
'

\    (backslash-space)
\!
\#
\$
\%
\&
\,
\:
\;
\>
\\
\_
\{
\|
\}
```

A

```
\above
\abovewithdelims
\acute
\aleph
\alpha
\amalg
\And
\angle
\approx
\approxeq                AMSsymbols
\arccos
```

\arcsin		
\arctan		
\arg		
\array		
\Arrowvert		
\arrowvert		
\ast		
\asymp		
\atop		
\atopwithdelims		

B

\backepsilon	AMSsymbols	
\backprime	AMSsymbols	
\backsim	AMSsymbols	
\backsimeq	AMSsymbols	
\backslash		
\backslash		
\bar		
\barwedge	AMSsymbols	
\Bbb		
\Bbbk	AMSsymbols	
\bbox	[bbox]	
\bcancel	cancel	
\because	AMSsymbols	
\begin		
\begingroup	begingroup	non-standard
\beta		
\beth	AMSsymbols	
\between	AMSsymbols	
\bf		
\Big		
\big		
\bigcap		
\bigcirc		
\bigcup		
\Bigg		
\bigg		
\Biggl		
\biggl		
\Biggm		
\biggm		
\Biggr		
\biggr		
\Bigl		
\bigl		
\Bigm		
\bigm		
\bigodot		
\bigoplus		
\bigotimes		
\Bigr		
\bigr		
\bigsqcup		
\bigstar	AMSsymbols	

\bigtriangledown	
\bigtriangleup	
\biguplus	
\bigvee	
\bigwedge	
\binom	AMSmath
\blacklozenge	AMSsymbols
\blacksquare	AMSsymbols
\blacktriangle	AMSsymbols
\blacktriangledown	AMSsymbols
\blacktriangleleft	AMSsymbols
\blacktriangleright	AMSsymbols
\bmod	
\boldsymbol	[boldsymbol]
\bot	
\bowtie	
\Box	AMSsymbols
\boxdot	AMSsymbols
\boxed	AMSmath
\boxminus	AMSsymbols
\boxplus	AMSsymbols
\boxtimes	AMSsymbols
\brace	
\bracevert	
\brack	
\breve	
\buildrel	
\bullet	
\Bumpeq	AMSsymbols
\bumpeq	AMSsymbols

C

\cal	
\cancel	cancel
\cancelto	cancel
\cap	
\Cap	AMSsymbols
\cases	
\cdot	
\cdotp	
\cdots	
\ce	mhchem
\cee	mhchem
\centerdot	AMSsymbols
\cf	mhchem
\cfrac	AMSmath
\check	
\checkmark	AMSsymbols
\chi	
\choose	
\circ	
\circeq	AMSsymbols
\circlearrowleft	AMSsymbols
\circlearrowright	AMSsymbols
\circledast	AMSsymbols

\circledcirc	AMSsymbols	
\circleddash	AMSsymbols	
\circledR	AMSsymbols	
\circledS	AMSsymbols	
\class	[HTML]	non-standard
\clubsuit		
\colon		
\color	color	
\colorbox	color	
\complement	AMSsymbols	
\cong		
\coprod		
\cos		
\cosh		
\cot		
\coth		
\cr		
\csc		
\cssId	[HTML]	non-standard
\cup		
\Cup	AMSsymbols	
\curlyeqprec	AMSsymbols	
\curlyeqsucc	AMSsymbols	
\curlyvee	AMSsymbols	
\curlywedge	AMSsymbols	
\curvearrowleft	AMSsymbols	
\curvearrowright	AMSsymbols	

D

\dagger		
\daleth	AMSsymbols	
\dashleftarrow	AMSsymbols	
\dashrightarrow	AMSsymbols	
\dashv		
\dbinom	AMSmath	
\ddagger		
\ddddot	AMSmath	
\dddot	AMSmath	
\ddot		
\ddots		
\DeclareMathOperator	AMSmath	
\definecolor	color	
\def	[newcommand]	
\deg		
\Delta		
\delta		
\det		
\dfrac	AMSmath	
\diagdown	AMSsymbols	
\diagup	AMSsymbols	
\diamond		
\Diamond	AMSsymbols	
\diamondsuit		
\digamma	AMSsymbols	
\dim		

\displaylines		
\displaystyle		
\div		
\divideontimes	AMSsymbols	
\dot		
\doteq		
\Doteq	AMSsymbols	
\doteqdot	AMSsymbols	
\dotplus	AMSsymbols	
\dots		
\dotsb		
\dotsc		
\dotsi		
\dotsm		
\dotso		
\doublebarwedge	AMSsymbols	
\doublecap	AMSsymbols	
\doublecup	AMSsymbols	
\Downarrow		
\downarrow		
\downdownarrows	AMSsymbols	
\downharpoonleft	AMSsymbols	
\downharpoonright	AMSsymbols	

E

\ell		
\emptyset		
\enclose	enclose	non-standard
\end		
\endgroup	begingroup	non-standard
\enspace		
\epsilon		
\eqalign		
\eqalignno		
\eqcirc	AMSsymbols	
\eqref	[AMSmath]	
\eqsim	AMSsymbols	
\eqslantgtr	AMSsymbols	
\eqslantless	AMSsymbols	
\equiv		
\eta		
\eth	AMSsymbols	
\exists		
\exp		

F

\fallingdotseq	AMSsymbols	
\fbox		
\fcolorbox	color	
\Finv	AMSsymbols	
\flat		
\forall		

\frac	
\frac	AMSmath
\frak	
\frown	

G

\Game	AMSsymbols
\Gamma	
\gamma	
\gcd	
\gdef	begingroup
\ge	
\genfrac	AMSmath
\geq	
\geqq	AMSsymbols
\geqslant	AMSsymbols
\gets	
\gg	
\ggg	AMSsymbols
\gggtr	AMSsymbols
\gimel	AMSsymbols
\global	begingroup
\gnapprox	AMSsymbols
\gneq	AMSsymbols
\gneqq	AMSsymbols
\gnsim	AMSsymbols
\grave	
\gt	
\gt	
\gtrapprox	AMSsymbols
\gtrdot	AMSsymbols
\gtreqless	AMSsymbols
\gtreqqless	AMSsymbols
\gtrless	AMSsymbols
\gtrsim	AMSsymbols
\gvertneqq	AMSsymbols

H

\hat	
\hbar	
\hbox	
\hdashline	
\heartsuit	
\hline	
\hom	
\hookleftarrow	
\hookrightarrow	
\hphantom	
\href	[HTML]
\hskip	
\hslash	AMSsymbols
\hspace	

\Huge	
\huge	
\idotsint	AMSmath

I

\iff	
\iiiint	AMSmath
\iiint	
\iint	
\Im	
\imath	
\impliedby	AMSsymbols
\implies	AMSsymbols
\in	
\inf	
\infty	
\injlim	AMSmath
\int	
\intercal	AMSsymbols
\intop	
\iota	
\it	

J

\jmath	
\Join	AMSsymbols

K

\kappa	
\ker	
\kern	

L

\label	[AMSmath]
\Lambda	
\lambda	
\land	
\langle	
\LARGE	
\Large	
\large	
\LaTeX	
\lbrace	
\lbrack	
\lceil	
\ldotp	

\ldots	
\le	
\leadsto	AMSsymbols
\left	
\Leftarrow	
\leftarrow	
\leftarrowtail	AMSsymbols
\leftharpoondown	
\leftharpoonup	
\leftleftarrows	AMSsymbols
\Leftrightarrow	
\leftrightarrow	
\leftrightarrows	AMSsymbols
\leftrightharpoons	AMSsymbols
\leftrightsquigarrow	AMSsymbols
\leftroot	
\leftthreetimes	AMSsymbols
\leq	
\leqalignno	
\leqq	AMSsymbols
\leqslant	AMSsymbols
\lessapprox	AMSsymbols
\lessdot	AMSsymbols
\lesseqgtr	AMSsymbols
\lesseqqgtr	AMSsymbols
\lessgtr	AMSsymbols
\lesssim	AMSsymbols
\let	[newcommand]
\lfloor	
\lg	
\lgroup	
\lhd	AMSsymbols
\lim	
\liminf	
\limits	
\limsup	
\ll	
\llap	
\llcorner	AMSsymbols
\Lleftarrow	AMSsymbols
\lll	AMSsymbols
\llless	AMSsymbols
\lmoustache	
\ln	
\lnapprox	AMSsymbols
\lneq	AMSsymbols
\lneqq	AMSsymbols
\lnot	
\lnsim	AMSsymbols
\log	
\Longleftarrow	
\longleftarrow	
\Longleftrightarrow	
\longleftrightarrow	
\longmapsto	
\Longrightarrow	
\longrightarrow	
\looparrowleft	AMSsymbols

\looparrowright	AMSsymbols	
\lor		
\lower		
\lozenge	AMSsymbols	
\lrcorner	AMSsymbols	
\Lsh	AMSsymbols	
\lt		
\lt		
\ltimes	AMSsymbols	
\lVert	AMSmath	
\lvert	AMSmath	
\lvertneqq	AMSsymbols	

M

\maltese	AMSsymbols	
\mapsto		
\mathbb		
\mathbf		
\mathbin		
\mathcal		
\mathchoice	[mathchoice]	
\mathclose		
\mathfrak		
\mathinner		
\mathit		
\mathop		
\mathopen		
\mathord		
\mathpunct		
\mathrel		
\mathring	AMSmath	
\mathrm		
\mathscr		
\mathsf		
\mathstrut		
\mathtip	action	non-standard
\mathtt		
\matrix		
\max		
\mbox		
\measuredangle	AMSsymbols	
\mho	AMSsymbols	
\mid		
\middle		
\min		
\mit		
\mkern		
\mmlToken		non-standard
\mod		
\models		
\moveleft		
\moveright		
\mp		
\mskip		
\mspace		

\mu	
\multimap	AMSsymbols

N

\nabla	
\natural	
\ncong	AMSsymbols
\ne	
\nearrow	
\neg	
\negmedspace	AMSmath
\negthickspace	AMSmath
\negthinspace	
\neq	
\newcommand	[newcommand]
\newenvironment	[newcommand]
\Newextarrow	extpfeil
\newline	
\nexists	AMSsymbols
\ngeq	AMSsymbols
\ngeqq	AMSsymbols
\ngeqslant	AMSsymbols
\ngtr	AMSsymbols
\ni	
\nLeftarrow	AMSsymbols
\nleftarrow	AMSsymbols
\nLeftrightarrow	AMSsymbols
\nleftrightarrow	AMSsymbols
\nleq	AMSsymbols
\nleqq	AMSsymbols
\nleqslant	AMSsymbols
\nless	AMSsymbols
\nmid	AMSsymbols
\nobreakspace	AMSmath
\nolimits	
\normalsize	
\not	
\notag	[AMSmath]
\notin	
\nparallel	AMSsymbols
\nprec	AMSsymbols
\npreceq	AMSsymbols
\nRightarrow	AMSsymbols
\nrightarrow	AMSsymbols
\nshortmid	AMSsymbols
\nshortparallel	AMSsymbols
\nsim	AMSsymbols
\nsubseteq	AMSsymbols
\nsubseteqq	AMSsymbols
\nsucc	AMSsymbols
\nsucceq	AMSsymbols
\nsupseteq	AMSsymbols
\nsupseteqq	AMSsymbols
\ntriangleleft	AMSsymbols
\ntrianglelefteq	AMSsymbols

\ntriangleright	AMSsymbols
\ntrianglerighteq	AMSsymbols
\nu	
\nVDash	AMSsymbols
\nVdash	AMSsymbols
\nvDash	AMSsymbols
\nvdash	AMSsymbols
\nwarrow	

O

\odot	
\oint	
\oldstyle	
\Omega	
\omega	
\omicron	
\ominus	
\operatorname	AMSmath
\oplus	
\oslash	
\otimes	
\over	
\overbrace	
\overleftarrow	
\overleftrightarrow	
\overline	
\overrightarrow	
\overset	
\overwithdelims	
\owns	

P

\parallel	
\partial	
\perp	
\phantom	
\Phi	
\phi	
\Pi	
\pi	
\pitchfork	AMSsymbols
\pm	
\pmatrix	
\pmb	
\pmod	
\pod	
\Pr	
\prec	
\precapprox	AMSsymbols
\preccurlyeq	AMSsymbols
\preceq	
\precnapprox	AMSsymbols

\precneqq	AMSsymbols
\precnsim	AMSsymbols
\precsim	AMSsymbols
\prime	
\prod	
\projlim	AMSmath
\propto	
\Psi	
\psi	

Q

\qquad	
\quad	

R

\raise		
\rangle		
\rbrace		
\rbrack		
\rceil		
\Re		
\ref	[AMSmath]	
\renewcommand	[newcommand]	
\renewenvironment	[newcommand]	
\require		non-standard
\restriction	AMSsymbols	
\rfloor		
\rgroup		
\rhd	AMSsymbols	
\rho		
\right		
\Rightarrow		
\rightarrow		
\rightarrowtail	AMSsymbols	
\rightharpoondown		
\rightharpoonup		
\rightleftarrows	AMSsymbols	
\rightleftharpoons		
\rightleftharpoons	AMSsymbols	
\rightrightarrows	AMSsymbols	
\rightsquigarrow	AMSsymbols	
\rightthreetimes	AMSsymbols	
\risingdotseq	AMSsymbols	
\rlap		
\rm		
\rmoustache		
\root		
\Rrightarrow	AMSsymbols	
\Rsh	AMSsymbols	
\rtimes	AMSsymbols	
\Rule		non-standard

\rVert	AMSmath
\rvert	AMSmath

S

\S		
\scr		
\scriptscriptstyle		
\scriptsize		
\scriptstyle		
\searrow		
\sec		
\setminus		
\sf		
\sharp		
\shortmid	AMSsymbols	
\shortparallel	AMSsymbols	
\shoveleft	AMSmath	
\shoveright	AMSmath	
\sideset	AMSmath	
\Sigma		
\sigma		
\sim		
\simeq		
\sin		
\sinh		
\skew		
\small		
\smallfrown	AMSsymbols	
\smallint		
\smallsetminus	AMSsymbols	
\smallsmile	AMSsymbols	
\smash		
\smile		
\Space		
\space		
\spadesuit		
\sphericalangle	AMSsymbols	
\sqcap		
\sqcup		
\sqrt		
\sqsubset	AMSsymbols	
\sqsubseteq		
\sqsupset	AMSsymbols	
\sqsupseteq		
\square	AMSsymbols	
\stackrel		
\star		
\strut		
\style	[HTML]	non-stanard
\subset		
\Subset	AMSsymbols	
\subseteq		
\subseteqq	AMSsymbols	
\subsetneq	AMSsymbols	
\subsetneqq	AMSsymbols	

\substack	AMSmath	
\succ		
\succapprox	AMSsymbols	
\succcurlyeq	AMSsymbols	
\succeq		
\succnapprox	AMSsymbols	
\succneqq	AMSsymbols	
\succnsim	AMSsymbols	
\succsim	AMSsymbols	
\sum		
\sup		
\supset		
\Supset	AMSsymbols	
\supseteq		
\supseteqq	AMSsymbols	
\supsetneq	AMSsymbols	
\supsetneqq	AMSsymbols	
\surd		
\swarrow		

T

\tag	[AMSmath]	
\tan		
\tanh		
\tau		
\tbinom	AMSmath	
\TeX		
\text		
\textbf		
\textit		
\textrm		
\textstyle		
\texttip	action	non-standard
\tfrac	AMSmath	
\therefore	AMSsymbols	
\Theta		
\theta		
\thickapprox	AMSsymbols	
\thicksim	AMSsymbols	
\thinspace		
\tilde		
\times		
\tiny		
\Tiny		non-standard
\to		
\toggle	action	non-standard
\top		
\triangle		
\triangledown	AMSsymbols	
\triangleleft		
\trianglelefteq	AMSsymbols	
\triangleq	AMSsymbols	
\triangleright		
\trianglerighteq	AMSsymbols	
\tt		

\twoheadleftarrow	AMSsymbols	
\twoheadrightarrow	AMSsymbols	

U

\ulcorner	AMSsymbols	
\underbrace		
\underleftarrow		
\underleftrightarrow		
\underline		
\underrightarrow		
\underset		
\unicode	[unicode]	non-standard
\unlhd	AMSsymbols	
\unrhd	AMSsymbols	
\Uparrow		
\uparrow		
\Updownarrow		
\updownarrow		
\upharpoonleft	AMSsymbols	
\upharpoonright	AMSsymbols	
\uplus		
\uproot		
\Upsilon		
\upsilon		
\upuparrows	AMSsymbols	
\urcorner	AMSsymbols	

V

\varDelta	AMSsymbols	
\varepsilon		
\varGamma	AMSsymbols	
\varinjlim	AMSmath	
\varkappa	AMSsymbols	
\varLambda	AMSsymbols	
\varliminf	AMSmath	
\varlimsup	AMSmath	
\varnothing	AMSsymbols	
\varOmega	AMSsymbols	
\varphi		
\varPhi	AMSsymbols	
\varpi		
\varPi	AMSsymbols	
\varprojlim	AMSmath	
\varpropto	AMSsymbols	
\varPsi	AMSsymbols	
\varrho		
\varsigma		
\varSigma	AMSsymbols	
\varsubsetneq	AMSsymbols	
\varsubsetneqq	AMSsymbols	
\varsupsetneq	AMSsymbols	
\varsupsetneqq	AMSsymbols	

\vartheta	
\varTheta	AMSsymbols
\vartriangle	AMSsymbols
\vartriangleleft	AMSsymbols
\vartriangleright	AMSsymbols
\varUpsilon	AMSsymbols
\varXi	AMSsymbols
\vcenter	
\vdash	
\Vdash	AMSsymbols
\vDash	AMSsymbols
\vdots	
\vec	
\vee	
\veebar	AMSsymbols
\verb	[verb]
\Vert	
\vert	
\vphantom	
\Vvdash	AMSsymbols

W

\wedge	
\widehat	
\widetilde	
\wp	
\wr	

X

\Xi	
\xi	
\xcancel	cancel
\xleftarrow	AMSmath
\xlongequal	extpfeil
\xmapsto	extpfeil
\xrightarrow	AMSmath
\xtofrom	extpfeil
\xtwoheadleftarrow	extpfeil
\xtwoheadrightarrow	extpfeil

Y

\yen	AMSsymbols

Z

\zeta	

Environments

LaTeX environments of the form `\begin{XXX} ... \end{XXX}` are provided where `XXX` is one of the following:

```
align                   [AMSmath]
align*                  [AMSmath]
alignat                 [AMSmath]
alignat*                [AMSmath]
aligned                 [AMSmath]
alignedat               [AMSmath]
array

Bmatrix
bmatrix

cases
CD                       AMSmath

eqnarray
eqnarray*
equation
equation*

gather                  [AMSmath]
gather*                 [AMSmath]
gathered                [AMSmath]

matrix
multline                [AMSmath]
multline*               [AMSmath]

pmatrix

smallmatrix              AMSmath
split                   [AMSmath]
subarray                 AMSmath

Vmatrix
vmatrix
```

MathJax MathML Support

The support for *MathML* in MathJax consists of three parts: the *mml2jax* preprocessor, the *MathML* input processor, and the *NativeMML* output processor. The first of these looks for `<math>` tags within your document and marks them for later processing by MathJax. The second converts the MathML to the internal format used by MathJax, and the third turns the internal format into MathML within the page so that it can be displayed by the browser's native MathML support.

Because of MathJax's modular design, you do not need to use all three of these components. For example, you could use the *tex2jax* preprocessor and the TeX input processor, but the NativeMML output processor, so that your mathematics is entered in TeX format, but displayed as MathML. Or you could use the *mml2jax* preprocessor and MathML input processor with the HTML-CSS output processor to make MathML available in browsers that don't have native MathML support. It is also possible to have MathJax select the output processor for you so that MathML is used in those browsers that support it well enough, while HTML-CSS is used for those that don't. See the *common*

configurations section for details and examples.

Of course it is also possible to use all three components together. It may seem strange to go through an internal format just to return to MathML in the end, but this is actually what makes it possible to view MathML within an HTML page (rather than an XHTML page), without the complications of handling special MIME-types for the document, or any of the other setup issues that make using native MathML difficult. MathJax handles the setup and properly marks the mathematics so that the browser will render it as MathML. In addition, MathJax provides its contextual menu for the MathML, which lets the user zoom the mathematics for easier reading, get and copy the source markup, and so on, so there is added value to using MathJax even with a pure MathML workflow.

MathML in HTML pages

For MathML that is handled via the preprocessor, you should not use named MathML entities, but rather use numeric entities like `&#x221A;` or unicode characters embedded in the page itself. The reason is that entities are replaced by the browser before MathJax runs, and some browsers report errors for unknown entities. For browsers that are not MathML-aware, that will cause errors to be displayed for the MathML entities. While that might not occur in the browser you are using to compose your pages, it can happen with other browsers, so you should avoid the named entities whenever possible. If you must use named entities, you may need to declare them in the *DOCTYPE* declaration by hand.

When you use MathML in an HTML document rather than an XHTML one (MathJax will work with both), you should not use the "self-closing" form for 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 (prior to HTML5) 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.

Content MathML

New in version 2.2 is experimental support for Content MathML. This uses an XSL style sheet developed by David Carlisle to convert Content MathML to Presentation MathML, which is then processed by MathJax.

To use Content MathML in your documents, simply include `"content-mathml.js"` in the `extensions` array of your MathML configuration block. For example

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  MathML: {
    extensions: ["content-mathml.js"]
  }
});
</script>
```

Note that this script tag must come *before* the script that loads `MathJax.js` itself.

Supported MathML commands

MathJax supports the MathML3.0 presentation 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 the elementary math tags: `mstack`, `mlongdiv`, `msgroup`, `msrow`, `mscarries`, and `mscarry`.
- No support for alignment groups in tables.
- No support for right-to-left rendering.
- Not all attributes are supported for tables. E.g., `columnspan` and `rowspan` are not implemented yet.

See the results of the MathML3.0 test suite for details.

Semantics and Annotations

Starting with MathJax version 2.3, some popular annotation formats like TeX, Maple, or Content MathML that are often included in the MathML source via the `semantics` element are accessible from the "`Show Math As`" menu. See the MathML Annotation Framework and the *The MathMenu extension* documentation for details.

MathJax AsciiMath Support

The support for *AsciiMath* in MathJax consists of two parts: the *asciimath2jax* preprocessor, and the *AsciiMath* input processor. The first of these looks for mathematics within your web page (indicated by delimiters like `` `...` ``) and marks the mathematics for later processing by MathJax. The AsciiMath input processor 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.

The AsciiMath input jax actually includes a copy of Peter Jipsen's `ASCIIMathML.js` file (see the AsciiMath home page for details), and is included by permission of the author. 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.

The *asciimath2jax* preprocessor can be configured to look for whatever markers you want to use for your math delimiters. See the *asciimath2jax configuration options* section for details on how to customize the action of *asciimath2jax*.

The AsciiMath input processor handles conversion of your mathematical notation into MathJax's internal format (which is essentially MathML). The AsciiMath input processor has few configuration options (see the *AsciiMath options* section for details).

The AsciiMath input jax handles only the original ASCIIMathML notation (from ASCIIMathML v1.4.7), not the extened 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 instead.

AsciiMath delimiters

By default, the *asciimath2jax* preprocessor defines the back-tick (`` ` ``) as the delimiters for mathematics in AsciiMath format. It does **not** define `$...$` as math delimiters. That is because dollar signs appear too often in non-mathematical settings, which could cause some text to be treated as mathematics unexpectedly. For example, with single-dollar

delimiters, "... the cost is $2.50 for the first one, and $2.00 for each additional one ..." would cause the phrase "2.50 for the first one, and" to be treated as mathematics since it falls between dollar signs. For this reason, if you want to use single-dollars for AsciiMath notation, you must enable that explicitly in your configuration:

```
MathJax.Hub.Config({
  asciimath2jax: {
    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 `config/default.js` file, or the *asiimath2jax configuration options* page, for additional configuration parameters that you can specify for the *asciimath2jax* preprocessor, which is the component of MathJax that identifies AsciiMath notation within the page.

AsciiMath in HTML documents

The AsciiMath syntax is descibed in the ASCIIMathML syntax page.

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>`) as AsciiMath-formatted math does not include HTML tags. Also, since the mathematics is initially given as text on 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 the browsers. For example,

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

will cause a problem, because the brower 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 "`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 to simply put spaces around these symbols to cause the browser to avoid them, so

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

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

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

Keep in mind that the browser interprets your text before MathJax does.

MathJax Output Formats

Currently, MathJax can render math in three ways:

- Using HTML-with-CSS to lay out the mathematics,
- Using *SVG* to lay out the mathematics, or

- Using a browser's native MathML support.

These are implemented by the *HTML-CSS*, *SVG* and *NativeMML* output processors.

If you are using one of the combined configuration files, then this will select one of these output processors for you. If the config file ends in `_HTML`, then it is the HTML-CSS output processor, and if it ends in `_SVG` then the SVG output processor will be used. If it ends in `_HTMLorMML`, then the NativeMML output processor will be chosen if the browser supports it well enough, otherwise HTML-CSS output will be used.

If you are performing your own in-line or file-based configuration, you select which one you want to use by including either `"output/HTML-CSS"`, `"output/SVG"`, or `"output/NativeMML"` in the *jax* array of your MathJax configuration. For example

```
jax: ["input/TeX","output/HTML-CSS"]
```

would specify TeX input and HTML-with-CSS output for the mathematics in your document.

The **HTML-CSS output processor** produces high-quality output in all major browsers, with results that are consistent across browsers and operating systems. This is MathJax's primary output mode. Its major advantage is its quality and consistency; its drawback is that it is slower than the NativeMML mode at rendering the mathematics. Historically, the performance in Internet Explorer (and IE8 in particular) was quite poor, with the page getting slower and slower as more math is processed. MathJax version 2.0 includes a number of optimizations to improve the display performance in IE, and it is now more comparable to other browsers. The HTML-CSS output uses web-based fonts so that users don't have to have math fonts installed on their computers, which introduces some printing issues in certain browsers.

The **SVG output processor** is new in MathJax version 2.0, and it uses *Scalable Vector Graphics* to render the mathematics on the page. SVG is supported in all the major browsers and most mobile devices; note, however, that Internet Explorer prior to IE9 does not support SVG, and IE9 only does in "IE9 standards mode", not its emulation modes for earlier versions. The SVG output mode is high quality and slightly faster than HTML-CSS, and it does not suffer from some of the font-related issues that HTML-CSS does, so prints well in all browsers. This format also works well in some ebook readers (e.g., iBooks). The disadvantages of this mode are the following: first, Internet Explorer only supports SVG in IE9 and later versions (and then only in IE9 standards mode or above), and some versions of the Android Internet browser don't have SVG enabled. Second, it does not take advantage of STIX fonts, and so only has access to the characters in the web-based fonts, and third, its variable-width tables become fixed size once they are typeset, and don't rescale if the window size changes (for example). Since equation numbers are handled through variable-width tables, that means equation numbers may not stay at the edge of the window if it is resized. For these reasons it is probably best not to force the use of SVG output unless you have some control over the browsers that are used to view your documents.

The **NativeMML output processor** uses the browser's internal MathML support (if any) to render the mathematics. Currently, Firefox has native support for MathML, and IE has the MathPlayer plugin for rendering MathML. Opera has some built-in support for MathML that works well with simple equations, but fails with more complex formulas, so we don't recommend using the NativeMML output processor with Opera. Safari has some support for MathML since version 5.1, but the quality is not as high as either Firefox's implementation or IE with MathPlayer. Chrome, Konqueror, and most other browsers don't support MathML natively, but this may change in the future, since MathML is part of the HTML5 specification.

The advantage of the NativeMML output Processor is its speed, since native MathML support is much faster than using complicated HTML and CSS to typeset mathematics, as the HTML-CSS output processor does. The disadvantage is that you are dependent on the browser's MathML implementation for your rendering, and these vary in quality of output and completeness of implementation. MathJax relies on features that are not available in some renderers (for example, Firefox's MathML support does not implement the features needed for labeled equations). The results using the NativeMML output processor may have spacing or other rendering problems that are outside of MathJax's control.

Automatic Selection of the Output Processor

Since not all browsers support MathML natively, it would be unwise to choose the NativeMML output processor unless you are sure of your audience's browser capabilities. MathJax can help with that, however, since a number of its combined configuration files will select NativeMML output when the browser supports it well enough, and HTML-CSS output otherwise. These are the configuration files that end in `_HTMLorMML`.

If you are doing your own configuration, there is a special configuration file that you can include that will choose between NativeMML and HTML-CSS depending on the browser in use. To invoke it, add `"MMLorHTML.js"` to your configuration's *config* array, and **do not** include an output processor in your *jax* array; MathJax will fill that in for you based on the abilities of your user's browser.

```
config: ["MMLorHTML.js"],
jax: ["input/TeX"]
```

By default, MathJax will choose HTML-CSS in all browsers except for one case: Internet Explorer when the MathPlayer plugin is present. In the past, MathJax selected NativeMML output for Firefox as well, but we have found that there are too many rendering issues with Firefox's native MathML implementation, and so MathJax now selects HTML-CSS output for Firefox by default as well. Users can still use the Mathjax contextual menu to select the NativeMML renderer if they wish to choose greater speed at the expense of some quality.

You can customize which choice MathJax makes on a browser-by-browser basis or a global basis. See the `config/default.js` file or the *Configuring MMLorHTML* section for further details. As an example, this configuration tells MathJax to use native MathML support rather than HTML-CSS output for Firefox:

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    MMLorHTML: { prefer: { Firefox: "MML" } }
  });
</script>
<script type="text/javascript"
  src="https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>
```

With this configuration, MathML output will be used for both Firefox and IE with the MathPlayer plugin. Note, however, that a user can employ the MathJax contextual menu to select the other renderer if he or she wishes.

MathJax produces MathML that models the underlying mathematics as best it can, rather than using complicated hacks to improve output for a particular MathML implementation. When you make the choice to use the NativeMML output processor, you are making a trade-off: gaining speed at the expense of quality and reliability, a decision that should not be taken lightly.

Automatic Line Breaking

The HTML-CSS and SVG output processors implement (most of) the MathML3 automatic line-breaking specification. (The NativeMML output processor relies on the browser's native MathML support to handle line breaking when it is used.) Since line-breaking takes extra processing and so can slow down the mathematical output, it is off by default, but you can enable it by adding

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  "HTML-CSS": { linebreaks: { automatic: true } },
         SVG: { linebreaks: { automatic: true } }
});
</script>
```

to your page just before the <script> tag that loads MathJax.js itself.

Note that line breaking only applies to displayed equations, not in-line equations (unless the in-line euqation is itself longer than a line), and that the line-breaks are only computed once when the equation is initially typeset, and do not change if the user changes the window size, or if the container changes size for some other reason.

You can control what width is used to determine where the line breaks shoud occur using the container parameter of the linebreaks block. By default it is the width of the containing element, but you can make it a fixed width, or make it a percentage of the container. See the *HTML-CSS configuration* or *SVG configuration* pages for more details.

The linbe-breaking 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. If you are using *MathML* as your input format, you can use the linebreak="goodbreak" and linebreak="badbreak" attributes on <mo> elements to help MathJax pick the best breakpoints for your mathematics.

HTML-CSS with IE

The performance of MathJax in Internet Explorer 8 and 9 has been substantially improved in version 2.0. The HTML-CSS output processing was redesigned to avoid the page reflows that were the main source of the speed problem in I8 and IE9. 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 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.

In the past, we recommended forcing IE8 and IE9 into IE7-emulation mode in order to get better performance. That is no longer necessary. Indeed, the fastest modes in IE8 and IE9 now are their IE8 standards and IE9 standards modes, so it is best to force the highest mode possible. That can be accomplished by adding

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

at the top of the <head> section of your HTML documents. Note that this line must come at the beginning of the <head>, before any stylesheets, scripts, or other content are loaded.

HTML-CSS Extensions

The HTML-CSS output jax uses elements with width set to 100% when it typesets displayed equations. If there are floating elements on the left or right, this can mean that displayed mathematics isn't properly centered, and can cause equation numbers to overlap the floating content. To avoid this, you can specify the *handle-floats* extension in the *extensions* array of your *HTML-CSS* configuration block.

```
"HTML-CSS": {
  extensions: ["handle-floats.js"]
}
```

This will use CSS that puts the displayed equations into elements that work like tabel cells, and won't overlap the floaring content. Because this is somewhat of a misuse of CSS, it is not used by default, but it has proved successful in most situations, so you may consider using it in pages that include material that floats to the left or right of text containing displayed mathematics, especially when equation numbers or tags are used.

See the *HTML-CSS configuration options* for other options of the HTML-CSS output jax.

MathJax Localization

As of version 2.2, MathJax's user interface (including its contextual menu, its help and about dialog boxes, and its warning messages) can all be localized to appear in languages other than English. For the available language options, see the TranslateWiki.net interface.

The language used by MathJax can be selected using the MathJax contextual menu. It includes a *Language* submenu that lists the available languages; selecting one will change the MathJax user interface to use that language.

Page authors can select a default language for MathJax so that, for example, a page that is written in French will have MathJax's user interface also in French. To do this, add `&locale=XX` after the configuration file in the `<script>` tag that loads the MathJax.js file, where `XX` is the two-letter code for the language. For example:

```
<script src="https://example.com/MathJax.js?config=TeX-AMS_HTML&locale=fr"></script>
```

will load MathJax using the French language. Users can still override this setting using the *Language* submenu of the MathJax contextual menu. This submenu can be disabled, however, using the *MathMenu configuration options*.

If you want to help in the translation process, please visit the TranslateWiki.net interface. The page localization.html is a convenient way to check the different translations.

MathJax Safe-mode

MathML includes the ability to include hyperlinks within your mathematics, and such links *could* be made to `javascript:` URL's. For example, the expression

```
<math>
  <mtext href="javascript:alert('Hello!')">Click Me</mtext>
</math>
```

would display the words "Click Me" that when clicked would generate an alert message in the browser. This is a powerful feature that provides authors the ability to tie actions to mathematical expressions.

Similarly, MathJax provides an HTML extension for the TeX language that allows you to include hyperlinks in your TeX formulas:

```
$E \href{javascript:alert("Einstein says so!")}{=} mc^2$
```

Here the equal sign will be a link that pops up the message about Einstein.

Both MathML and the HTML extension for TeX allow you to add CSS styles, classes, and id's to your math elements as well. These features can be used to produce interactive mathematical expressions to help your exposition, improve student learning, and so on.

If you are using MathJax in a community setting, however, like a question-and-answer forum, a wiki, a blog with user comments, or other situations where your readers can enter mathematics, then your readers would be able to use such powerful tools to corrupt the page, or fool other readers into giving away sensitive information, or interrupt their reading experience in other ways. In such environments, you may want to limit these abilities so that your readers are protected form these kinds of malicious actions.

(Authors who are writing pages that don't allow users to enter data on the site do not have to worry about such problems, as the only mathematical content will be their own. It is only when users can contribute to the page that you have to be careful.)

MathJax provides a *Safe* extension to help you limit your contributors' powers. There are two ways to load it. The easiest is to add `,Safe` after the configuration file when you are loading `MathJax.js`:

```
<script src="https://example.com/MathJax.js?config=TeX-AMS_HTML,Safe"></script>
```

This causes MathJax to load the `TeX-AMS_HTML` configuration file, and then the `Safe` configuration, which adds the Safe extension to your `extensions` array so that it will be loaded with the other extensions.

Alternatively, if you are using in-line configuration, you could just include `"Safe.js"` in your `extensions` array directly:

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  jax: ["input/TeX","output/HTML-CSS"],
  extensions: ["tex2jax.js","Safe.js"]
});
</script>
<script src="https://example.com/MathJax.js"></script>
```

The Safe extension has a number of configuration options that let you fine-tune what is allowed and what is not. See the *Safe extension options* for details.

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.

Forums

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

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

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

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.

Documentation

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

"Powered by MathJax"

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

CHAPTER 2

MathJax Configuration Options

Configuration Objects

The various components of MathJax, including its input and output processors, its preprocessors, its extensions, and the MathJax core, all can be configured through the `config/default.js` file, or via a `MathJax.Hub.Config()` call (indeed, if you look closely, you will see that `config/default.js` is itself one big call to `MathJax.Hub.Config()`). Anything that is in `config/default.js` can be included in-line to configure MathJax.

The structure that you pass to `MathJax.Hub.Config()` is a JavaScript object that includes *name:value* pairs giving the names of parameters and their values, with pairs separated by commas. Be careful not to include a comma after the last value, however, as some browsers (namely Internet Explorer) will fail to process the configuration if you do.

The MathJax components, like the TeX input processor, have their own sections in the configuration object labeled by the component name, and using an object as its value. That object is itself a configuration object made up of *name:value* pairs that give the configuration options for the component.

For example,

```
MathJax.Hub.Config({
  showProcessingMessages: false,
  jax: ["input/TeX", "output/HTML-CSS"],
  TeX: {
    TagSide: "left",
    Macros: {
      RR: '{\\bf R}',
      bold: ['{\\bf #1}',1]
    }
  }
});
```

is a configuration that includes two settings for the MathJax Hub (one for *showProcessingMessages* and one for the *jax* array), and a configuration object for the TeX input processor. The latter includes a setting for the TeX input processor's *TagSide* option (to set tags on the left rather than the right) and a setting for *Macros*, which defines new

TeX macros (in this case, two macros, one called `\RR` that produces a bold "R", and one called `\bold` that puts is argument in bold face).

The `config/default.js` file is another example that shows nearly all the configuration options for all of MathJax's components.

The Core Configuration Options

The options below control the MathJax Hub, and so determine the code behavior of MathJax. They are given with their default values.

`jax: ["input/TeX","output/HTML-CSS"]`
: A comma-separated list of input and output jax to initialize at startup. Their main code is loaded only when they are actually used, so it is not inefficient to include jax that may not actually be used on the page. These are found in the `MathJax/jax` directory.

`extensions: []`
: A comma-separated list of extensions to load at startup. The default directory is `MathJax/extensions`. The `tex2jax` and `mml2jax` preprocessors can be listed here, as well as a `FontWarnings` extension that you can use to inform your user that mathematics fonts are available that they can download to improve their experience of your site.

`config: []`
: A comma-separated list of configuration files to load when MathJax starts up, e.g., to define local macros, etc., and there is a sample config file named `config/local/local.js`. The default directory is the *MathJax/config* directory. The `MMLorHTML.js` configuration is one such configuration file, and there are a number of other pre-defined configurations (see *Using a configuration file* for more details).

`styleSheets: []`
: A comma-separated list of CSS stylesheet files to be loaded when MathJax starts up. The default directory is the *MathJax/config* directory.

`styles: {}`
: CSS styles to be defined dynamically at startup time. These are in the form *selector:rules* (see *CSS Style Objects* for complete details).

`preJax: null and postJax: null`
: Patterns to remove from before and after math script tags. If you are not using one of the preprocessors, you need to insert something extra into your HTML file in order to avoid a bug in Internet Explorer. IE removes spaces from the DOM that it thinks are redundant, and since a `<script>` tag usually doesn't add content to the page, if there is a space before and after a MathJax `<script>` tag, IE will remove the first space. When MathJax inserts the typeset mathematics, this means there will be no space before it and the preceding text. In order to avoid this, you should include some "guard characters" before or after the math SCRIPT tag; define the patterns you want to use below. Note that these are used as part of a regular expression, so you will need to quote special characters. Furthermore, since they are javascript strings, you must quote javascript special characters as well. So to obtain a backslash, you must use `\\` (doubled for javascript). For example, `"\\["` represents the pattern `\[` in the regular expression, or `[` in the text of the web page. That means that if you want an actual backslash in your guard characters, you need to use `"\\\\"` in order to get `\\` in the regular expression, and `\` in the actual text. If both `preJax` and `postJax` are defined, both must be present in order to be removed.

  See also the `preRemoveClass` comments below.

  Examples:

  `preJax: "\\\\\\\\\"` makes a double backslash the `preJax` text

  `preJax: "\\[\\[", postJax: "\\]\\]"` makes it so jax scripts must be enclosed in double brackets.

preRemoveClass: "MathJax_Preview"
This is the CSS class name for math previews that will be removed preceding a MathJax SCRIPT tag. If the tag just before the MathJax `<script>` tag is of this class, its contents are removed when MathJax processes the `<script>` tag. This allows you to include a math preview in a form that will be displayed prior to MathJax performing its typesetting. It also avoids the Internet Explorer space-removal bug, and can be used in place of `preJax` and `postJax` if that is more convenient.

For example

```
<span class="MathJax_Preview">[math]</span><script  type="math/tex">...</script>
```

would display "[math]" in place of the math until MathJax is able to typeset it.

See also the `preJax` and `postJax` comments above.

showProcessingMessages: true
This value controls whether the *Processing Math: nn%* messages are displayed in the lower left-hand corner. Set to `false` to prevent those messages (though file loading and other messages will still be shown).

messageStyle: "normal"
This value controls the verbosity of the messages in the lower left-hand corner. Set it to `"none"` to eliminate all messages, or set it to `"simple"` to show "Loading..." and "Processing..." rather than showing the full file name or the percentage of the mathematics processed.

displayAlign: "center" and displayIndent: "0em"
These two parameters control the alignment and shifting of displayed equations. The first can be `"left"`, `"center"`, or `"right"`, and determines the alignment of displayed equations. When the alignment is not `"center"`, the second determines an indentation from the left or right side for the displayed equations.

delayStartupUntil: "none"
Normally MathJax will perform its startup commands (loading of configuration, styles, jax, and so on) as soon as it can. If you expect to be doing additional configuration on the page, however, you may want to have it wait until the page's onload handler is called. If so, set this to `"onload"`. You can also set this to `"configured"`, in which case, MathJax will delay its startup until you explicitly call `MathJax.Hub.Configured()`. See *Configuring MathJax after it is loaded* for more details.

skipStartupTypeset: false
Normally MathJax will typeset the mathematics on the page as soon as the page is loaded. If you want to delay that process, in which case you will need to call `MathJax.Hub.Typeset()` yourself by hand, set this value to `true`.

elements: []
This is a list of DOM element ID's that are the ones to process for mathematics when any of the Hub typesetting calls (`Typeset()`, `Process()`, `Update()`, etc.) are called with no element specified, and during MathJax's initial typesetting run when it starts up. This lets you restrict the processing to particular containers rather than scanning the entire document for mathematics. If none are supplied, the complete document is processed.

positionToHash: true
Since typesetting usually changes the vertical dimensions of the page, if the URL contains an anchor position, then after the page is typeset, you may no longer be positioned at the correct position on the page. MathJax can reposition to that location after it completes its initial typesetting of the page. This value controls whether MathJax will reposition the browser to the `#hash` location from the page URL after typesetting for the page.

showMathMenu: true
showMathMenuMSIE: true
These control whether to attach the MathJax contextual menu to the expressions typeset by MathJax. Since the code for handling MathPlayer in Internet Explorer is somewhat delicate, it is controlled separately via `showMathMenuMSIE`, but the latter is now deprecated in favor of the MathJax contextual menu settings for MathPlayer (see below).

If `showMathMenu` is `true`, then right-clicking (on Windows or Linux) or control-clicking (on Mac OS X) will produce a MathJax menu that allows you to get the source of the mathematics in various formats, change the size of the mathematics relative to the surrounding text, get information about MathJax, and configure other MathJax settings.

Set this to `false` to disable the menu. When `true`, the `MathMenu` configuration block determines the operation of the menu. See *the MathMenu options* for more details.

These values used to be listed in the separate output jax, but have been moved to this more central location since they are shared by all output jax. MathJax will still honor their values from their original positions, if they are set there.

`menuSettings: { ... }`
This block contains settings for the mathematics contextual menu that act as the defaults for the user's settings in that menu. The possible values are:

`zoom: "None"`
This indicates when typeset mathematics should be zoomed. It can be set to `"None"`, `"Hover"`, `"Click"`, or `"Double-Click"` to set the zoom trigger.

`CTRL: false, ALT: false, CMD: false, Shift: false`
These values indicate which keys must be pressed in order for math zoom to be triggered. For example, if `CTRL` is set to `true` and `zoom` is `"Click"`, then math will be zoomed only when the user control-clicks on mathematics (i.e., clicks while holding down the *CTRL* key). If more than one is `true`, then all the indicated keys must be pressed for the zoom to occur.

`zscale: "200%"`
This is the zoom scaling factor, and it can be set to any of the values available in the *Zoom Factor* menu of the *Settings* submenu of the contextual menu.

`context: "MathJax"`
This controls what contextual menu will be presented when a right click (on a PC) or CTRL-click (on the Mac) occurs over a typeset equation. When set to `"MathJax"`, the MathJax contextual menu will appear; when set to `"Browser"`, the browser's contextual menu will be used. For example, in Internet Explorer with the MathPlayer plugin, if this is set to `"Browser"`, you will get the MathPlayer contextual menu rather than the MathJax menu.

`texHints: true`
This controls whether the "Show Source" menu item includes special class names that help MathJax to typeset the mathematics that was produced by the TeX input jax. If these are included, then you can take the output from "Show Source" and put it into a page that uses MathJax's MathML input jax and expect to get the same results as the original TeX. (Without this, there may be some spacing differences.)

There are also settings for `format`, `renderer`, `font`, `mpContext`, and `mpMouse`, but these are maintained by MathJax and should not be set by the page author.

`errorSettings: { ... }`
This block contains settings that control how MathJax responds to unexpected errors while processing mathematical equations. Rather than simply crash, MathJax can report an error and go on. The options you can set include:

`message: ["[Math Processing Error]"]`
This is an HTML snippet that will be inserted at the location of the mathematics for any formula that causes MathJax to produce an internal error (i.e., an error in the MathJax code itself). See the *description of HTML snippets* for details on how to represent HTML code in this way.

`style: {color:"#CC0000", "font-style":"italic"}`
This is the CSS style description to use for the error messages produced by internal MathJax errors. See the section on *CSS style objects* for details on how these are specified in JavaScript.

v1.0-compatible: true
: This controls whether MathJax issues the warning about not having an explicit configuration in the event that the *jax* array is empty after configuration is complete. If you really intend that array to be empty, set this flag to `false`. Note that setting this to false does **not** cause a default configuration file to be loaded.

The tex2jax Preprocessor

The options below control the operation of the *tex2jax* preprocessor that is run when you include `"tex2jax.js"` in the *extensions* array of your configuration. They are listed with their default values. To set any of these options, include a `tex2jax` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  tex2jax: {
    inlineMath: [ ['$','$'], ['\\(','\\)'] ]
  }
});
```

would set the `inlineMath` delimiters for the *tex2jax* preprocessor.

inlineMath: [['\(','\)']]
: Array of pairs of strings that are to be used as in-line math delimiters. The first in each pair is the initial delimiter and the second is the terminal delimiter. You can have as many pairs as you want. For example,

  ```
  inlineMath: [ ['$','$'], ['\\(','\\)'] ]
  ```

  would cause *tex2jax* to look for `$...$` and `\(...\)` as delimiters for inline mathematics. (Note that the single dollar signs are not enabled by default because they are used too frequently in normal text, so if you want to use them for math delimiters, you must specify them explicitly.)

  Note that the delimiters can't look like HTML tags (i.e., can't include the less-than sign), as these would be turned into tags by the browser before MathJax has the chance to run. You can only include text, not tags, as your math delimiters.

displayMath: [['$$','$$'], ['\[','\]']]
: Array of pairs of strings that are to be used as delimiters for displayed equations. The first in each pair is the initial delimiter and the second is the terminal delimiter. You can have as many pairs as you want.

  Note that the delimiters can't look like HTML tags (i.e., can't include the less-than sign), as these would be turned into tags by the browser before MathJax has the chance to run. You can only include text, not tags, as your math delimiters.

balanceBraces: true,
: This value determines whether *tex2jax* requires braces to be balanced within math delimiters (which allows for nested dollar signs). Set to `false` to get pre-v2.0 compatibility. When `true`,

  ```
  $y = x^2 \hbox{ when $x > 2$}$.
  ```

  will be properly handled as a single expression. When `false`, it would be interpreted as two searpate expressions, each with improperly balanced braces.

processEscapes: false
: When set to `true`, you may use `\$` to represent a literal dollar sign, rather than using it as a math delimiter. When `false`, `\$` will not be altered, and the 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 *tex2jax* will convert it to a regular dollar sign in the rendered document.

processEnvironments: true
When `true`, *tex2jax* looks not only for the in-line and display math delimiters, but also for LaTeX environments (`\begin{something}...\end{something}`) and marks them for processing by MathJax. When `false`, LaTeX environments will not be processed outside of math mode.

preview: "TeX"
This controls whether *tex2jax* inserts `MathJax_Preview` spans to make a preview available, and what preview to use, when it locates in-line or display mathematics in the page. The default is `"TeX"`, which means use the TeX code as the preview (which will be visible until it is processed by MathJax). Set to `"none"` to prevent previews from being inserted (the math will simply disappear until it is typeset). Set to an array containing the description of an HTML snippet in order to use the same preview for all equations on the page.

Examples:

```
preview: ["[math]"],     //  insert the text "[math]" as the preview
```

```
preview: [["img",{src: "/images/mypic.jpg"}]],  // insert an image as the preview
```

See the *description of HTML snippets* for details on how to represent HTML code in this way.

skipTags: ["script","noscript","style","textarea","pre","code"]
This array lists the names of the tags whose contents should not be processed by *tex2jax* (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.

ignoreClass: "tex2jax_ignore"
This is the class name used to mark elements whose contents should not be processed by tex2jax (other than to look for the `processClass` 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 `ignoreClass: "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 `ignoreClass: "class1|class2"` any element assigned a class of either `class1` or `class2` will be skipped.

processClass: "tex2jax_process"
This is the class name used to mark elements whose contents *should* be processed by *tex2jax*. This is used to restart processing within tags that have been marked as ignored via the `ignoreClass` or to cause a tag that appears in the `skipTags` 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 `processClass: "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 `processClass: "class1|class2"` any element assigned a class of either `class1` or `class2` will have its contents processed.

The mml2jax Preprocessor

The options below control the operation of the *mml2jax* preprocessor that is run when you include `"mml2jax.js"` in the *extensions* array of your configuration. They are listed with their default values. To set any of these options, include a `mml2jax` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  mml2jax: {
    preview: "mathml"
```

```
  }
});
```

would set the `preview` parameter to `"mathml"`.

`preview: "mathml"`
: This controls whether *mml2jax* inserts `MathJax_Preview` spans to make a preview available, and what preview to use, when it locates mathematics on the page. Possible values are: `"mathml"`, `"alttext"`, , `"altimg"`, `"none"`, or an HTML snippet.

  The default is `"mathml"`, in which case MathJax keeps the content of the `<math>` tag as the preview (until it is processed by MathJax). Set to `"alttext"`, to use the `<math>` tag's `alttext` attribute as the preview, if the tag has one. Set to `"altimg"` to use an image described by the `altimg*` attributes of the `<math>` element. Set to `"none"` to prevent the previews from being inserted (the math will simply disappear until it is typeset). Set to an array containing the description of an HTML snippet in order to use the same preview for all equations on the page (e.g., you could have it say `"[math]"` or load an image).

  Examples:

```
preview: ["[math]"],     //  insert the text "[math]" as the preview
```

```
preview: [["img",{src: "/images/mypic.jpg"}]],  // insert an image as the preview
```

  See the *description of HTML snippets* for details on how to represent HTML code in this way.

The asciimath2jax Preprocessor

The options below control the operation of the *asciimath2jax* preprocessor that is run when you include `"asciimath2jax.js"` in the *extensions* array of your configuration. They are listed with their default values. To set any of these options, include a `asciimath2jax` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  asciimath2jax: {
    delimiters: [['`','`'], ['$','$']]
  }
});
```

would set the ASCIIMath delimiters for the *asciimath2jax* preprocessor to include dollar signs as well as back-ticks.

`delimiters: [['`','`']]`
: Array of pairs of strings that are to be used as 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,

```
delimiters: [ ['$','$'], ['`','`'] ]
```

  would cause *asciimath2jax* to look for `$...$` and `` `...` `` as delimiters for inline mathematics. (Note that the single dollar signs are not enabled by default because they are used too frequently in normal text, so if you want to use them for math delimiters, you must specify them explicitly.)

  Note that the delimiters can't look like HTML tags (i.e., can't include the less-than sign), as these would be turned into tags by the browser before MathJax has the chance to run. You can only include text, not tags, as your math delimiters.

`preview: "AsciiMath"`
: This controls whether *asciimath2jax* inserts `MathJax_Preview` spans to make a preview available, and what

preview to use, when it locates in-line or display mathematics in the page. The default is `"AsciiMath"`, which means use the ASCIIMath code as the preview (which will be visible until it is processed by MathJax). Set to `"none"` to prevent previews from being inserted (the math will simply disappear until it is typeset). Set to an array containing the description of an HTML snippet in order to use the same preview for all equations on the page.

Examples:

```
preview: ["[math]"],     //  insert the text "[math]" as the preview
```

```
preview: [["img",{src: "/images/mypic.jpg"}]],  // insert an image as the preview
```

See the *description of HTML snippets* for details on how to represent HTML code in this way.

skipTags: ["script","noscript","style","textarea","pre","code"]
This array lists the names of the tags whose contents should not be processed by *asciimath2jax* (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.

ignoreClass: "asciimath2jax_ignore"
This is the class name used to mark elements whose contents should not be processed by asciimath2jax (other than to look for the `processClass` 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 `ignoreClass: "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 `ignoreClass: "class1|class2"` any element assigned a class of either `class1` or `class2` will be skipped.

processClass: "asciimath2jax_process"
This is the class name used to mark elements whose contents *should* be processed by *asciimath2jax*. This is used to restart processing within tags that have been marked as ignored via the `ignoreClass` or to cause a tag that appears in the `skipTags` 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 `processClass: "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 `processClass: "class1|class2"` any element assigned a class of either `class1` or `class2` will have its contents processed.

The jsMath2jax Preprocessor

The options below control the operation of the *jsMath2jax* preprocessor that is run when you include `"jsMath2jax.js"` in the *extensions* array of your configuration. They are listed with their default values. To set any of these options, include a `jsMath2jax` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  jsMath2jax: {
    preview: "none"
  }
});
```

would set the `preview` parameter to `"none"`.

preview: "TeX"
This controls whether *jsMath2jax* inserts `MathJax_Preview` spans to make a preview available, and what

preview to use, when it locates in-line or display mathematics in the page. The default is `"TeX"`, which means use the TeX code as the preview (which will be visible until it is processed by MathJax). Set to `"none"` to prevent previews from being inserted (the math will simply disappear until it is typeset). Set to an array containing the description of an HTML snippet in order to use the same preview for all equations on the page.

Examples:

```
preview: ["[math]"],     //  insert the text "[math]" as the preview
```

```
preview: [["img",{src: "/images/mypic.jpg"}]],  // insert an image as the preview
```

See the *description of HTML snippets* for details on how to represent HTML code in this way.

The TeX input processor

The options below control the operation of the TeX input processor that is run when you include `"input/TeX"` in the *jax* array of your configuration or load a combined configuration file 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.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  TeX: {
    Macros: {
      RR: '{\\bf R}',
      bold: ['{\\bf #1}', 1]
    }
  }
});
```

would set the `Macros` configuration option to cause two new macros to be defined within the TeX input processor.

`TagSide: "right"`
: This specifies the side on which `\tag{}` macros will place the tags. Set it to `"left"` to place the tags on the left-hand side.

`TagIndent: ".8em"`
: This is the amount of indentation (from the right or left) for the tags produced by the `\tag{}` macro.

`MultLineWidth: "85%"`
: The width to use for the *multline* environment that is part of the `AMSmath` extension. This width gives room for tags at either side of the equation, but if you are displaying mathematics in a small area or a thin column of text, you might need to change the value to leave sufficient margin for tags.

`equationNumbers: {}`
: This object controls the automatic equation numbering and the equation referencing. It contains the following values:

  `autoNumber: "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.

  `formatNumber: function (n) {return n}`
  : 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.

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

formatID: function {return 'mjx-eqn-'+String(n).replace(/[:'"<>&]/g,"")}
A function that rells MathJax what ID to use as an anchor for the equation (so that it can be used in URL references).

formatURL: function (id) {return '#'+escape(id)}
A function that takes an equation ID and returns the URL to link to it.

useLabelIds: true
This controls whether element ID's use the `\label` name or the equation number. When `true`, use the label, when `false`, use the equation number.

See the MathJax examples page for some examples of equation numbering.

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 an array of the form *[value,n]*, where *value* is the replacement text and *n* is the number of parameters for the macro. Note that since the *value* is a javascript string, backslashes in the replacement text must be doubled to prevent them from acting as javascript escape characters.

For example,

```
Macros: {
  RR: '{\\bf R}',
  bold: ['{\\bf #1}', 1]
}
```

would ask the TeX processor to define two new macros: `\RR`, which produces a bold-face "R", and `\bold{...}`, which takes one parameter and sets it in the bold-face font.

MAXMACROS: 10000
Because a definition of the form `\def\x{\x} \x` would cause MathJax to loop infinitely, the *MAXMACROS* constant will limit the number of macro substitutions allowed in any expression processed by MathJax.

MAXBUFFER: 5*1024
Because a definition of the form `\def\x{\x aaa} \x` would loop infinitely, and at the same time stack up lots of a's in MathJax's equation buffer, the *MAXBUFFER* constant is used to limit the size of the string being processed by MathJax. It is set to 5KB, which should be sufficient for any reasonable equation.

The MathML input processor

The options below control the operation of the MathML input processor that is run when you include `"input/MathML"` in the *jax* array of your configuration or load a combined configuration file that includes the MathML input jax. They are listed with their default values. To set any of these options, include a `MathML` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  MathML: {
    useMathMLspacing: true
  }
});
```

would set the `useMathMLspacing` option so that the MathML rules for spacing would be used (rather than TeX spacing rules).

useMathMLspacing: false
Specifies whether to use TeX spacing or MathML spacing when the *HTML-CSS* output jax is used.

The AsciiMath input processor

The options below control the operation of the AsciiMath input processor that is run when you include "input/AsciiMath" in the *jax* array of your configuration or load a combined configuration file that includes the AsciiMath input jax. They are listed with their default values. To set any of these options, include a AsciiMath section in your MathJax.Hub.Config() call. For example

```
MathJax.Hub.Config({
  AsciiMath: {
    displaystyle: false
  }
});
```

would set the displaystyle configuration option so that the limits for operators like summation symbols will appear next to them rather than above and below.

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 appear 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.

decimal: "."
This is the character to be used for decimal points in numbers. if you change this to ",", then you need to be careful about entering points or intervals. E.g., use (1, 2) rather than (1,2) in that case.

The HTML-CSS output processor

The options below control the operation of the HTML-CSS output processor that is run when you include "output/HTML-CSS" in the *jax* array of your configuration or load a combined configuration file that includes the HTML-CSS output jax. They are listed with their default values. To set any of these options, include a "HTML-CSS" section in your MathJax.Hub.Config() call. Note that, because of the dash, you need to enclose the name in quotes. For example

```
MathJax.Hub.Config({
  "HTML-CSS": {
    preferredFont: "STIX"
  }
});
```

would set the preferredFont option to the *STIX* fonts.

scale: 100
The scaling factor (as a percentage) of math with respect to the surrounding text. The *HTML-CSS* output processor tries to match the ex-size of the mathematics with that of the text where it is placed, but you may want to adjust the results using this scaling factor. The user can also adjust this value using the contextual menu item associated with the typeset mathematics.

minScaleAdjust: 50
This gives a minimum scale (as a percent) for the scaling used by MathJax to match the equation to the surrounding text. This will prevent MathJax from making the mathematics too small.

availableFonts: ["STIX","TeX"]
: This is a list of the fonts to look for on a user's computer in preference to using MathJax's web-based fonts. These must correspond to directories available in the `jax/output/HTML-CSS/fonts` directory, where MathJax stores data about the characters available in the fonts. Set this to `["TeX"]`, for example, to prevent the use of the *STIX* fonts, or set it to an empty list, *[]*, if you want to force MathJax to use web-based or image fonts.

preferredFont: "TeX"
: Which font to prefer out of the `availableFonts` list, when more than one is available on the user's computer. Set it to `null` if you want MathJax to use web-based or image fonts.

webFont: "TeX"
: This is the web-based font to use when none of the fonts listed above are available on the user's computer. The possible values are `TeX`, `STIX-Web`, `Asana-Math`, `Neo-Euler`, `Gyre-Pagella`, `Gyre-Termes` and `Latin-Modern`. Note that not all mathematical characters are available in all fonts (e.g., Neo-Euler does not include italic characters), so some mathematics may work better in some fonts than in others. The `STIX-Web` font is the most complete.

  These fonts are stored in the `fonts/HTML-CSS` folder in the MathJax directory. Set this to `null` to disable web fonts.

imageFont: "TeX"
: This is the font to use for image fallback mode (when none of the fonts listed above are available and the browser doesn't support web-fonts via the `@font-face` CSS directive). Note that currently only the TeX font is available as an image font (they are stored in the `fonts/HTML-CSS` directory).

  Set this to `null` if you want to prevent the use of image fonts (e.g., you have deleted or not installed the image fonts on your server). In this case, only browsers that support web-based fonts will be able to view your pages without having the fonts installed on the client computer. The browsers that support web-based fonts include: IE6 and later, Chrome, Safari3.1 and above, Firefox3.5 and later, and Opera10 and later. Note that Firefox3.0 is **not** on this list.

undefinedFamily: "STIXGeneral, 'Arial Unicode MS', serif"
: This is the font-family CSS value used for characters that are not in the selected font (e.g., for web-based fonts, this is where to look for characters not included in the MathJax web fonts). IE will stop looking after the first font that exists on the system (even if it doesn't contain the needed character), so order these carefully.

mtextFontInherit: false
: This setting controls whether `<mtext>` elements will be typeset using the math fonts or the font of the surrounding text. When `false`, the font for `mathvariant="normal"` will be used; when `true`, the font will be inherited from the surrounding paragraph.

EqnChunk: 50
EqnChunkFactor: 1.5
EqnChunkDelay: 100
: These values control how "chunky" the display of mathematical expressions will be; that is, how often the equations will be updated as they are processed.

  `EqnChunk` is the number of equations that will be typeset before they appear on screen. Larger values make for less visual flicker as the equations are drawn, but also mean longer delays before the reader sees anything.

  `EqChunkFactor` is the factor by which the `EqnChunk` will grow after each chunk is displayed.

  `EqChunkDelay` is the time (in milliseconds) to delay between chunks (to allow the browser to respond to other user interaction).

  Set `EqnChunk` to 1, `EqnChunkFactor` to 1, and `EqnChunkDelay` to 10 to get the behavior from MathJax v1.1 and below.

matchFontHeight: true
This option indicates whether MathJax should try to adjust the x-height of equations to match the x-height of the surrounding text. See the *MatchWebFonts options* for finer control, especially if you are using Web fonts.

linebreaks: {}
This is an object that configures automatic linebreaking in the HTML-CSS output. In order to be backward compatible with earlier versions of MathJax, only explicit line breaks are performed by default, so you must enable line breaks if you want automatic ones. The object contains the following values:

automatic: false
This controls the automatic breaking of expressions: when `false`, only `linebreak="newline"` is processed; when `true`, line breaks are inserted automatically in long expressions.

width: "container"
This controls how wide the lines of mathematics can be.

Use an explicit width like `"30em"` for a fixed width. Use `"container"` to compute the size from the containing element. Use `"nn% container"` for a portion of the container. Use `"nn%"` for a portion of the window size.

The container-based widths may be slower, and may not produce the expected results if the layout width changes due to the removal of previews or inclusion of mathematics during typesetting.

styles: {}
This is a list of CSS declarations for styling the HTML-CSS output. See the definitions in `jax/output/HTML-CSS/config.js` for some examples of what are defined by default. See *CSS Style Objects* for details on how to specify CSS style in a JavaScript object.

showMathMenu: true
This value has been moved to the core configuration block, since it applies to all output jax, but it will still be honored (for now) if it is set here. See the *Core configuration options* for more details.

tooltip: { ... }
This sets the configuration options for `<maction>` elements with `actiontype="tooltip"`. (See also the `#MathJax_Tooltip` style setting in `jax/output/HTML-CSS/config.js`, which can be overridden using the `styles` option above.)

The `tooltip` section can contain the following options:

delayPost: 600
The delay (in milliseconds) before the tooltip is posted after the mouse is moved over the `maction` element.

delayClear: 600
The delay (in milliseconds) before the tooltop is cleared after the mouse moves out of the `maction` element.

offsetX: 10
offsetY: 5
These are the offset from the mouse position (in pixels) where the tooltip will be placed.

The NativeMML output processor

The options below control the operation of the NativeMML output processor that is run when you include `"output/NativeMML"` in the *jax* array of your configuration or load a combined configuration file taht includes the NativeMML output jax. They are listed with their default values. To set any of these options, include a `NativeMML` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  NativeMML: {
    scale: 105
  }
});
```

would set the `scale` option to 105 percent.

scale: 100
: The scaling factor (as a percentage) of math with respect to the surrounding text. The *NativeMML* output processor tries to match the ex-size of the mathematics with that of the text where it is placed, but you may want to adjust the results using this scaling factor. The user can also adjust this value using the contextual menu item associated with the typeset mathematics.

minScaleAdjust: 50
: This gives a minimum scale (as a percent) for the scaling used by MathJax to match the equation to the surrounding text. This will prevent MathJax from making the mathematics too small.

matchFontHeight: true
: This option indicates whether MathJax should try to adjust the x-height of equations to match the x-height of the surrounding text. See the *MatchWebFonts options* for finer control, especially if you are using Web fonts.

showMathMath: true
showMathMenuMSIE: true
: These values have been moved to the core configuration block, since it applies to all output jax, but they will still be honored (for now) if it is set here. See the *Core configuration options* for more details.

styles: {}
: This is a list of CSS declarations for styling the NativeMML output. See the definitions in `jax/output/NativeMML/config.js` for some examples of what are defined by default. See *CSS Style Objects* for details on how to specify CSS style in a JavaScript object.

The SVG output processor

The options below control the operation of the SVG output processor that is run when you include `"output/SVG"` in the *jax* array of your configuration or load a combined configuration file that includes the SVG output jax. They are listed with their default values. To set any of these options, include an `SVG` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  SVG: {
    scale: 120
  }
});
```

would set the `scale` option to 120%.

scale: 100
: The scaling factor (as a percentage) of math with respect to the surrounding text. The *SVG* output processor tries to match the ex-size of the mathematics with that of the text where it is placed, but you may want to adjust the results using this scaling factor. The user can also adjust this value using the contextual menu item associated with the typeset mathematics.

minScaleAdjust: 50
: This gives a minimum scale (as a percent) for the scaling used by MathJax to match the equation to the surrounding text. This will prevent MathJax from making the mathematics too small.

font: "TeX"
This is the font to use for rendering the mathematics. The possible values are `TeX`, `STIX-Web`, `Asana-Math`, `Neo-Euler`, `Gyre-Pagella`, `Gyre-Termes` and `Latin-Modern`. Note that not all mathematical characters are available in all fonts (e.g., Neo-Euler does not include italic characters), so some mathematics may work better in some fonts than in others. The `STIX-Web` font is the most complete.

blacker: 10
This is the stroke width to use for all character paths (1em = 1000 units). This is a cheap way of getting slightly lighter or darker characters, but remember that not all displays will act the same, so a value that is good for you may not be good for everyone.

undefinedFamily: "STIXGeneral, 'Arial Unicode MS', serif"
This is the font-family CSS value used for characters that are not in the selected font (e.g., this is where to look for characters not included in the MathJax TeX fonts). IE will stop looking after the first font that exists on the system (even if it doesn't contain the needed character), so order these carefully.

mtextFontInherit: false
This setting controls whether `<mtext>` elements will be typeset using the math fonts or the font of the surrounding text. When `false`, the font for `mathvariant="normal"` will be used; when `true`, the font will be inherited from the surrounding paragraph.

addMMLclasses: false
This controls whether the MathML structure is retained and CSS classes are added to mark the original MathML elements (as in the output from the *HTML-CSS* output jax). By default, the SVG output jax removes unneeded nesting in order to produce a more efficient markup, but if you want to use CSS to style the elements as if they were MathML, you might need to set this to true.

EqnChunk: 50
EqnChunkFactor: 1.5
EqnChunkDelay: 100
These values control how "chunky" the display of mathematical expressions will be; that is, how often the equations will be updated as they are processed.

`EqnChunk` is the number of equations that will be typeset before they appear on screen. Larger values make for less visual flicker as the equations are drawn, but also mean longer delays before the reader sees anything.

`EqChunkFactor` is the factor by which the `EqnChunk` will grow after each chunk is displayed.

`EqChunkDelay` is the time (in milliseconds) to delay between chunks (to allow the browser to respond to other user interaction).

Set `EqnChunk` to 1, `EqnChunkFactor` to 1, and `EqnChunkDelay` to 10 to get the behavior from MathJax v1.1 and below.

matchFontHeight: true
This option indicates whether MathJax should try to adjust the x-height of equations to match the x-height of the surrounding text. See the *MatchWebFonts options* for finer control, especially if you are using Web fonts.

linebreaks: {}
This is an object that configures automatic linebreaking in the SVG output. In order to be backward compatible with earlier versions of MathJax, only explicit line breaks are performed by default, so you must enable line breaks if you want automatic ones. The object contains the following values:

automatic: false
This controls the automatic breaking of expressions: when `false`, only `linebreak="newline"` is processed; when `true`, line breaks are inserted automatically in long expressions.

width: "container"
This controls how wide the lines of mathematics can be.

Use an explicit width like "30em" for a fixed width. Use "container" to compute the size from the containing element. Use "nn% container" for a portion of the container. Use "nn%" for a portion of the window size.

The container-based widths may be slower, and may not produce the expected results if the layout width changes due to the removal of previews or inclusion of mathematics during typesetting.

styles: {}
This is a list of CSS declarations for styling the SVG output. See the definitions in jax/output/SVG/config.js for some examples of what are defined by default. See *CSS Style Objects* for details on how to specify CSS style in a JavaScript object.

tooltip: { ... }
This sets the configuration options for <maction> elements with actiontype="tooltip". (See also the #MathJax_Tooltip style setting in jax/output/SVG/config.js, which can be overridden using the styles option above.)

The tooltip section can contain the following options:

delayPost: 600
The delay (in milliseconds) before the tooltip is posted after the mouse is moved over the maction element.

delayClear: 600
The delay (in milliseconds) before the tooltop is cleared after the mouse moves out of the maction element.

offsetX: 10
offsetY: 5
These are the offset from the mouse position (in pixels) where the tooltip will be placed.

The MMLorHTML configuration options

The options below control the operation of the MMLorHTML configuration file that is run when you include "MMLorHTML.js" in the *config* array of your configuration, or when you use one of the combined configuration files that ends with _HTMLorMML. They are listed with their default values. To set any of these options, include a MMLorHTML section in your MathJax.Hub.Config() call. For example

```
MathJax.Hub.Config({
  MMLorHTML: {
    prefer: {
      Opera: "MML"
    }
  }
});
```

would set the prefer option so that the Opera browser would prefer MathML to HTML-CSS output (while leaving the settings for other browsers unchanged).

Note that if you use the MMLorHTML.js configuration file, you should **not** specify an output processor in the *jax* array of your configuration; *MMLorHTML* will fill that in for you.

prefer: {
MSIE: "MML",
Firefox: "HTML",
Safari: "HTML",
Chrome: "HTML",
Opera: "HTML",

other: "HTML"
}

This lets you set the preferred renderer on a browser-by-browser basis. You set the browser to either `"MML"` or `"HTML"` depending on whether you want to use the *NativeMML* or *HTML-CSS* output processor. Note that although Opera and Safari do process some MathML natively, their support is not sufficient to handle the more complicated output generated by MathJax, so their settings are `"HTML"` by default. Although Firefox does support a large subset of MathJax, it does not implement all the features needed by MathJax, and so it is also set to `"HTML"` by default (this is new in v2.0).

Note that users can still use the MathJax contextual menu to select a different renderer after the default one has been chosen by `MMLorHTML.js`.

The MathMenu extension

The options below control the operation of the contextual menu that is available on mathematics that is typeset by MathJax. They are listed with their default values. To set any of these options, include a `MathMenu` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  MathMenu: {
    delay: 600
  }
});
```

would set the `delay` option to 600 milliseconds.

delay: 150

This is the hover delay for the display (in milliseconds) for submenus in the contextual menu: when the mouse is over a submenu label for this long, the menu will appear. (The submenu also will appear if you click on its label.)

helpURL: "http://www.mathjax.org/help/user/"

This is the URL for the MathJax Help menu item. When the user selects that item, the browser opens a new window with this URL.

showRenderer: true

This controls whether the "Math Renderer" item will be displayed in the "Math Settings" submenu of the MathJax contextual menu. It allows the user to change between the *HTML-CSS*, *NativeMML*, and *SVG* output processors for the mathematics on the page. Set to `false` to prevent this menu item from showing.

showFontMenu: false

This controls whether the "Font Preference" item will be displayed in the "Math Settings" submenu of the MathJax contextual menu. This submenu lets the user select what font to use in the mathematics produced by the *HTML-CSS* output processor. Note that changing the selection in the font menu will cause the page to reload. Set to `false` to prevent this menu item from showing.

showLocale: true

This controls whether the "Language" item will be displayed in the MathJax contextual menu. This submenu allows the user to select the language to use for the MathJax user interface, including the contextual menu, the about and help dialogs, the message box at the lower left, and any warning messages produced by MathJax. Set this to `false` to prevent this menu item from showing. This will force the user to use the language you have set for MathJax.

showMathPlayer: true

This controls whether the "MathPlayer" item will be displayed in the "Math Settings" submenu of the MathJax contextual menu. This submenu lets the user select what events should be passed on to the MathPlayer plugin,

when it is present. Mouse events can be passed on (so that clicks will be processed by MathPlayer rather than MathJax), and menu events can be passed on (to allow the user access to the MathPlayer menu). Set to `false` to prevent this menu item from showing.

`showContext: false`
: This controls whether the "Contextual Menu" item will be displayed in the "Math Settings" submenu of the MathJax contextual menu. It allows the user to decide whether the MathJax menu or the browser's default contextual menu will be shown when the context menu click occurs over mathematics typeset by MathJax. Set to `false` to prevent this menu item from showing.

`semanticsAnnotations: { ... }`
: 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 following annotation formats, the source will be available via the "Show Math As" menu. Each format has a list of possible encodings. For example, `"TeX": ["TeX", "LaTeX", "application/x-tex"]` will map an annotation with an encoding of `"TeX"`, `"LaTeX"`, or `"application/x-tex"` to the `"TeX"` menu.

`windowSettings: { ... }`
: These are the settings for the `window.open()` call that creates the *Show Source* window. The initial width and height will be reset after the source is shown in an attempt to make the window fit the output better.

`styles: {}`
: This is a list of CSS declarations for styling the menu components. See the definitions in `extensions/MathMenu.js` for details of what are defined by default. See *CSS Style Objects* for details on how to specify CSS style in a JavaScript object.

The MathZoom extension

The options below control the operation of the Math-Zoom feature that allows users to see an enlarged version of the mathematics when they click or hover over typeset mathematics. They are listed with their default values. To set any of these options, include a `MathZoom` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  MathZoom: {
    styles: {
      "#MathJax_Zoom": {
        "background-color": "#0000F0"
      }
    }
  }
});
```

would set the background color of the Zoom box to a very light blue.

Mathematics is zoomed when the user "triggers" the zoom by an action, either clicking on the mathematics, double-clicking on it, or holding the mouse still over it (i.e., "hovering"). Which trigger is used is set by the user via the math contextual menu (or by the author using the `menuSettings` configuration section of the *core configuration options* <*configure-hub*>).

`delay: 500`
: This value is now stored as the `hover` parameter in the *MathEvents* configuration options, and will have no effect if given here.

`styles: {}`
: This is a list of CSS declarations for styling the zoomed mathematics. See the definitions in `extensions/MathZoom.js` for details of what are defined by default. See *CSS Style Objects* for details on how to specify CSS style in a JavaScript object.

The MathEvents extension

The options below control the operation of the MathEvents component that allows handles mouse and menu events attached to mathematics that is typeset by MathJax. They are listed with their default values. To set any of these options, include a `MathEvents` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  MathEvents: {
    hover: 400
  }
});
```

would set the required delay for hovering over a math element to 400 milliseconds.

`hover: 500`
: This value is the time (in milliseconds) that a user must hold the mouse still over a math element before it is considered to be hovering over the math.

`styles: {}`
: This is a list of CSS declarations for styling the zoomed mathematics. See the definitions in `extensions/MathEvents.js` for details of what are defined by default. See *CSS Style Objects* for details on how to specify CSS style in a JavaScript object.

The FontWarnings extension

The options below control the operation of the *FontWarnings* extension that is run when you include `"FontWarnings.js"` in the *extensions* array of your configuration. They are listed with their default values. To set any of these options, include a `FontWarnings` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  FontWarnings: {
    fadeoutTime: 2*1000
  }
});
```

would set the `fadeoutTime` option to 2000 milliseconds (2 seconds).

`messageStyle: { ... }`
: This sets the CSS styles to be used for the font warning message window. See the `extensions/FontWarnings.js` file for details of what are set by default. See the *CSS style objects* for details about how to specify CSS styles via javascript objects.

`Message: { ... }`
: This block contains HTML snippets to be used for the various messages that the FontWarning extension can produce. There are three messages that you can redefine to suit your needs:

  `webFont: [ ... ]`
  : The message used for when MathJax uses web-based fonts (rather than local fonts installed on the user's system).

  `imageFonts: [ ... ]`
  : The message used for when MathJax must use image fonts rather than local or web-based fonts (for those browsers that don't handle the `@font-face` CSS directive).

noFonts: [...]
The message used when MathJax is unable to find any font to use (i.e., neither local nor web-based nor image-based fonts are available).

Any message that is set to `null` rather than an HTML snippet array will not be presented to the user, so you can set, for example, the `webFont` message to `null` in order to have the `imageFonts` and `noFonts` messages, but no message if MathJax uses web-based fonts.

See the description of *HTML snippets* for details about how to describe the messages using HTML snippets. Note that in addition to the usual rules for defining such snippets, the FontWarnings snippets can include references to pre-defined snippets (that represent elements common to all three messages). These are defined below in the `HTML` block, and are referenced using `["name"]` within the snippet, where *name* is the name of one of the snippets defined in the `HTML` configuration block. For example

```
Message: {
  noFonts: [
    ["closeBox"],
    "MathJax is unable to locate a font to use to display ",
    "its mathematics, and image fonts are not available, so it ",
    "is falling back on generic unicode characters in hopes that ",
    "your browser will be able to display them.  Some characters ",
    "may not show up properly, or possibly not at all.",
    ["fonts"],
    ["webfonts"]
  ]
}
```

refers to the `closeBox`, `fonts` and `webfonts` snippets declared in the `HTML` section.

HTML: { ... }
This object defines HTML snippets that are common to more than one message in the `Message` section above. They can be included in other HTML snippets by by using `["name"]` in an HTML snippet, where *name* refers to the name of the snippet in the `HTML` block. The pre-defined snippets are:

closeBox
The HTML for the close box in the FontWarning message.

webfonts
The HTML for a paragraph suggesting an upgrade to a more modern browser that supports web fonts.

fonts
HTML that includes links to the MathJax and STIX font download pages.

STIXfonts
HTML that gives the download link for the STIX fonts only. (Used in place of *fonts* when the *HTML-CSS* option for *availableFonts* only includes the *STIX* fonts.)

TeXfonts
HTML that gives the download link for the MathJax TeX fonts only. (Used in place of *fonts* when the *HTML-CSS* option for *availableFonts* only includes the *TeX* fonts.)

You can add your own pre-defined HTML snippets to this object, or override the ones that are there with your own text.

removeAfter: 12*1000
This is the amount of time to show the FontWarning message, in milliseconds. The default is 12 seconds. Setting this value to zero means that the message will not fade out (the user must close it manually).

fadeoutSteps: 10
This is the number of steps to take while fading out the FontWarning message. More steps make for a smoother fade-out. Set to zero to cause the message to be removed without fading.

fadeoutTime: 1.5*1000
This is the time used to perform the fade-out, in milliseconds. The default is 1.5 seconds.

The Safe extension

The options below control the operation of the *Safe* extension that is run when you include "Safe.js" in the *extensions* array of your configuration, or include Safe in the config= options when you load MathJax.js. They are listed with their default values. To set any of these options, include a Safe section in your MathJax.Hub.Config() call. For example

```
MathJax.Hub.Config({
  Safe: {
    allow: {
      URLs: "safe",
      classes: "safe",
      cssIDs: "safe",
      styles: "safe",
      fontsize: "all",
      require: "safe"
    }
  }
});
```

would set the fontsize option to "all", and the others to "safe" (these are described below).

The Safe extension affects both the TeX input and MathML input jax.

allow: { ... }
This block contains the flags that control what the Safe extension will allow, and what it will block. The flags can be set to "all", "none", or "safe". When set to "all", no filtering is done for these values (this gives MathJax's default behavior). When set to "none", these values are always filtered out. When set to "safe", then only some values are allowed, as described below.

URLs: "safe"
When set to "safe" only URL's with protocols that are listed in the safeProtocols property (see below) are allowed as targets of href attributes or the \href macro. By default, these are http://, https://, and file:// URL's.

classes: "safe"
When set to "safe", only class names that begin with MJX- and contain only letters, numbers, or the characters -, _, or . are allowed.

cssIDs: "safe"
When set to "safe", only ID's that begin with MJX- and contain only letters, numbers, or the characters -, _, or . are allowed.

styles: "safe"
When set to "safe", only styles taken from a predefined set of styles are allowed to be given. These are listed in the safeStyles property (see below).

require: "safe"
When set to "safe", only the extensions listed in the safeRequire property (see below) are allowed to be loaded by the \require{} macro.

fontsize: "all"
When set to "safe", MathJax will try to limit the font size to sizes between those given by the sizeMin and sizeMax properties. These are .7 and 1.44 by default, which means sizes between \scriptsize

and \large are allowed. This also filters MathML fontsize, mathsize, and scriptminsize attributes, but here, "safe" acts as "none", since they are given in sizes with units, and the actual size of the units is not determined at input time (it is part of the output processing). In addition, the scriptlevel attribute is restricted to non-negative values (so scripts can't be made larger), and the scriptsizemultiplier is restricted to being no larger than 1, and no less than .6.

sizeMin: .7
This is the minimum font size (in em's) that the TeX input jax will allow when fontsize is set to "safe" above. The default is the size of \scriptsize. Values less than this are set to this value.

sizeMax: 1.44
This is the maximum font size (in em's) that the TeX input jax will allow when fontsize is set to "safe" above. The default is the size of \large. Values larger than this are set to this value.

safeProtocols: {...}
This is an object that lists the protocols that can be used in href attributes and the \href macro when URLs is set to "safe" above. The default is

```
safeProtocols: {
  http: true,
  https: true,
  file: true,
  javascript: false
}
```

Note that if a protocol doesn't appear in the list, it is assumed to be false, so technically, javascript need not have been listed, but it is given to make it explicit that it should not be allowed.

safeStyles: {...}
This is an object that lists the style properties that can be used in MathML style attributes and the \style and \bbox macros when styles is set to "safe" in the allowed property above. The default is

```
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
}
```

Any style property that doesn't appear on this list is not allowed to be entered and will be removed (silently) from the style definition.

safeRequire: {...}
This is an object that lists the TeX extensions that can be loaded via the \require{} macro when require is set to "safe" in the allowed property above. The default is

```
safeRequire: {
  action: true,
  amscd: true,
  amsmath: true,
```

```
  amssymbols: true,
  autobold: false,
  "autoload-all": false,
  bbox: true,
  begingroup: true,
  boldsymbol: true,
  cancel: true,
  color: true,
  enclose: true,
  extpfeil: true,
  HTML: true,
  mathchoice: true,
  mhchem: true,
  newcommand: true,
  noErrors: false,
  noUndefined: false,
  unicode: true,
  verb: true
}
```

These configuration options give you a lot of control over what actions MathJax is allowed to take. It is also possible override the individual filtering functions in order to customize the filtering even further, should that be needed. See the code for the details of the function names and their definitions.

The Match Web Fonts extension

The options below control the operation of the *MatchWebFonts* extension that is run when you include `"MatchWebFonts.js"` in the *extensions* array of your configuration. They are listed with their default values. To set any of these options, include a `MatchWebFonts` section in your `MathJax.Hub.Config()` call. For example

```
MathJax.Hub.Config({
  MatchWebFonts: {
    matchFor: {
      "HTML-CSS": true,
      NativeMML: false,
      SVG: false
    },
    fontCheckDelay: 2000,
    fontCheckTimeout: 30 * 1000
  }
});
```

would ask to apply font size matching for the *HTML-CSS* output mode but not for the *NativeMML* or *SVG* modes. It would also tell the extension to wait 2 seconds before starting to look for web font arrivals, and to continue checking for 30 seconds.

This extension is designed for pages that have mathematics within text that is displayed using webfonts, and works around a basic problem of webfonts – a missing API. Webfonts often don't appear until after a delay, and the browser will substitute another font until then; unfortunately there is no signal for when the font becomes available. Since the arrival of the webfonts can significantly change ex and em sizes (and MathJax checks these to match them with its own font size), this extension will check for changes of em and ex sizes (indicating the arrival of webfonts) and rerender equations if necessary.

`matchFor: { ... }`

This block controls whether to apply font size matching for each output mode.

"HTML-CSS": "true"
Whether to match the font size for the *HTML-CSS* output.

NativeMML: "true"
Whether to match the font size for the *NativeMML* output.

SVG: "true"
Whether to match the font size for the *SVG* output.

fontCheckDelay: 500
Initial delay before the first check for web fonts (in milliseconds).

fontCheckTimeout: 15 * 1000
How long to keep looking for fonts (in milliseconds).

CHAPTER 3

Upgrading MathJax

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)

What's New in MathJax v2.2

MathJax v2.2 includes a number of new features, as well a more than 40 important bug fixes.

Features:

- Localization of MathJax user interface. (German and French translations currently available in addition to English.)
- Commutative diagrams via the `AMScd` extension.
- New Safe-mode extension that allows you to restrict potentially dangerous features of MathJax when it is used in a shared environment (e.g., href to javascript, styles and classes, etc.)
- Improve MathML rendering for `mfenced` and `mlabeldtr` elements in browsers that don't support them well.
- Experimental Content MathML support.

TeX input:

- Avoid potential infinite loops in `\mathchoice` constructs. (Issue #373)
- Add error message when an evironment closes with unbalanced braces. (Issue #454)
- Allow spaces in the RGB, rgb, and greyscale color specifications. (Issue #446)
- Process `\$` in `\text` arguements. (ssue #349)
- Preserve spaces within `\verb` arguments. (Issue #381)
- Make `\smallfrown` and `\smallsmile` come from the variant font so they have the correct size. (Issue #436)
- Make the input TeX jax generate mrow plus mo elements rather than mfenced elements (for better compatibility with native MathML implementations).
- Make `\big` and its relatives use script or scriptscript fonts (although size is still absolute, as it is in TeX) so that it balances the text weight in scripts. (Issue #350)
- Convert true and false attributes to booleans in `\mmlToken`. (Issue #451)

AsciiMath:

- Rename AsciiMath config option from `decimal` to `decimalsign`. (Issue #384)

Fonts:

- Add Greek Delta to SVG fonts. (Issue #347)
- Fix monospace space character to be the same width as the other monospace characters. (Issue #380)
- Better handling of unknown or invalid values for mathvariant or values not supported by generic fonts.

MathML:

- Handle empty child nodes better.
- Improved MathML rendering for `mfenced` and `mlabeldtr` elements.
- Ignore `linebreak` attribute on `mspace` when dimensional attributes are set. (Issue #388)
- Implement `rowspacing`/`columnspacing` for `mtable` in native MathML output in Firefox using cell padding.

HTML-CSS/SVG output

- Allow `\color` to override link color in SVG output. (Issue #427)
- Add min-width to displayed equations with labels so that they cause their containers to have non-zero width (like when they are in a table cell or an absolutlye positioned element). (Issue #428)
- Fix a processing error with elements that contain hyperlinks. (Issue #364)
- Try to isolate MathJax from CSS transitions. (Issue #449)
- Go back to using em's (rounded to nearest pixel) for Chrome. Rounding makes the placement work more reliably, while still being in relative units. (Issue #443)

- Prevent error when math contains characters outside of the MathJax fonts. (Issue #441)
- Make final math size be in relative units so that it prints even if print media has a different font size. (Issue #386)
- Don't scale line thickness for `menclose` elements (so lines won't disapear in scripts). (Issue #414)
- Fix `fontdata.js` to allow it to be included in combined configuration files. (Issue #413)
- Makes math-based tooltips be spaced properly when rendered. (Issue #412)
- Fix Math Processing Error when `&ApplyFunction;` is used without preceeding content. (Issue #410)
- Fix a problem using an empty table as a super- or subscript. (Issue #392)
- Handle the case where selection in maction is invalid or out of range. (Issue #365)
- Add a pixel extra around the SVG output to accommodate antialiasing pixels. (Issue #383)
- Fix Math Processing Error for `msubsup`/`msub`/`msup` elements.
- Limit the number of repetition to build stretchy chars in HTML-CSS. (Issue #366)
- Fix Math Processing Error in `mmultiscripts`/`menclose`. (Issue 362)

Interface:

- Make zoom work properly with expressions that have full width (e.g., tagged equations).
- Handle zooming when it is inside a scrollable element when it is not the main body element. (Issue #435)
- Update math processing errors to include original format and actual error message in the "Show Math As" menu. (Issue #450)
- Add a Help dialog box (rather than link to mathjax.org).
- Remove the v1.0 configuration warning. (Issue #445)
- Trap errors while saving cookies (and go on silently). (Issue #374)
- Fix typo in IE warning message. (Issue #397)
- Use UA string sniffing for identifying Firefox and handle detecting mobile versions better.
- Make MathML source show non-BMP characters properly. (Issue #361)
- Make tool tips appear above zoom boxes. (Issue #351)

Miscellaneous:

- Allow preview for preprocessors to be just a plain string (rather than requiring `[string]`).
- Remap back-tick to back-quote. (Issue #402)
- Handle script tags in `HTML.Element()` so they work in IE. (Issue #342)
- Add the `MathJax_Preview` class to the `ignoreClass` list so that `tex2jax` and `asciimath2jax` won't process previews accidentally. (Issue #378)
- Fix processing errors with various table and menclose attributes. (Issue #367)
- Use `hasOwnProperty()` when checking file specification objects (prevents problems when `Object.prototype` has been modified). (Issue #352)

What's New in MathJax v2.1

MathJax v2.1 is primarily a bug-fix release. Numerous display bugs, line-breaking problems, and interface issues have been resolved. The following lists indicate the majority of the bugs that have been fixed for this release.

Interface

- Make NativeMML output properly handle iOS double-tap-and-hold, and issue warning message when switching to NativeMML output.
- Use `scrollIntoView` to handle `positionToHash` rather than setting the document location to prevent pages from refreshing after MathJax finishes processing the math.
- Handle positioning to a hash URL when the link is to an element within SVG output.
- Make `href`'s work in SVG mode in all browsers.
- Fix problem with opening the "Show Math As" window in WebKit (affected Chrome 18, and Safari 5.1.7).
- Use MathJax message area rather than window status line for `maction` with `actiontype='statusline'` to avoid security restrictions in some browsers.
- Fix issue where zoom box for math that has been wrapped to the beginning of a line would be positioned at the end of the previous line.
- Fix a problem where IE would try to typset the page before it was completely available, causing it to not typeset all the math on the page (or in some cases *any* of the math).
- Allow decimal scale values in the dialog for setting the scale.
- Fix SVG output so that setting the scale will rescale the existing mathematics.
- Add close button to About box and don't make clicking box close it (only clicking button).
- Make About box show 'woff or otf' when otf fonts are used (since both are requested).
- Have output jax properly skip math when the input jax has had an internal failure and so didn't produce any element jax.
- Produce `MathJax.Hub` signal when `[Math Processing Error]` is generated.

Line-breaking

- Fix problem with SVG output disappearing during line breaks when equation numbers are also present.
- Fix problem with potential infinite loop when an `<mspace>` is an embellished operator that causes a linebreak to occur.
- Allow line breaks within the base of `<msubsup>` to work so that the super and subscripts stay with the last line of the base.
- Fix `<mfenced>` so that when it contains a line break the delimiters and separators are not lost.
- Allow line breaks at delimiters and separators in <mfenced> elements.
- Fix issue with line breaking where some lines were going over the maximum width.
- Fix problem with line breaking inside `<semantics>` elements.
- Fix problem with line breaking where the incorrect width was being used to determine breakpoint penalties, so some long lines were not being broken.

HTML-CSS/SVG display

- Fix several Chrome alignment and sizing issues, including problems with horizontal lines at the tops of roots, fraction bars being too long, etc.
- Resolve a problem with how much space is reserved for math equations when a minimum font size is set in the browser.
- Force final math span to be remeasured so that we are sure the container is the right size.
- Fix alignment problem in `<msubsup>`.
- Fix processing error when rowalign has a bad value.
- Fix a vertical placement problem with stretched elements in mtables in HTML-CSS, and improve performace for placeing the extension characters.
- Handle spacing for U+2061 (function apply) better.
- Better handling of primes and other pseudo scripts in HTML-CSS and SVG output.
- Fixed a problem with `<mmultiscripts>` in SVG mode that caused processing error messages.
- Fix misplaced `\vec` arrows in Opera and IE.
- Make `<mi>` with more than one letter have `texClass` OP rather than ORD in certain cases so it will space as a function.
- Make HTML snippet handler accept a string as contents, even if not enclosed in braces.
- Fix spacing for functions that have powers (e.g., `\sin^2 x`).
- Fix problem with SVG handling of `\liminf` and `\limsup` where the second half of the function name was dropped.
- Fixed a problem where HTML-CSS and SVG output could leave partial equations in the DOM when the equation processing was interrupted to load a file.
- Fix problems with `<mtable>`, `<ms>`, and `<mmultiscripts>` which weren't handling styles.
- Make column widths and row heights take minsize into account in `<mtable>`.
- Fix typo in `handle-floats.js` that caused it to not compile.
- Fix problem in HTML-CSS output with `<msubsup>` when super- or subscript has explicit style.

TeX emulation

- Allow negative dimensions for `\\[]` but clip to 0 since this isn't really allowed in MathML.
- Fixed problem where \ with whitespace followed by [would incorrectly be interpretted as \[dimen].
- Make `jsMath2jax` run before other preprocessors so that `tex2jax` won't grab environments from inside the jsMath spans and divs before jsMath2jax sees them.
- Fix issue with `\vec` not producing the correct character for `\vec{\mathbf{B}}` and similar constructs.
- Combine multiple primes into single unicode characters.
- Updated the unicode characters used for some accents and a few other characters to more appropriate choices. See issues #116, #119, and #216 in the MathJax issue tracker on GitHub.
- Remove unwanted 'em' from `eqnarray columnwidth` values.
- Make eqnarray do equation numbering when numbering is enabled.

- Make vertical stretchy characters stand on the baseline, and improve spacing of some stretchy chars.
- Make `mtextFontInherit` use the style and weight indicated in the math, so that `\textbf` and `\textit` will work properly.
- Add `\textcolor` macro to the color extension.
- Added RGB color model to the color extension.
- Automatically load the AMSmath extension when needed by the `mhchem` extension.
- Add <<=> arrow to `mhchecm` extension
- Fix alignment of prescripts in `mhchem` to properly right-justify the scripts.
- Expose the CE object in the `mhchem` extension.
- Make `autoload-all` skip extensions that are already loaded, and not redefine user-defined macros.
- Fix most extensions to not overwrite user defined macros when the extension is loaded.
- Ignore `\label{}` with no label.
- Make `\injlim` and friends produce single `<mi>` elements for thier names rather than one for each letter.
- Handle primes followed by superscript as real TeX does in TeX input jax.
- Handle a few more negations (e.g., of arrows) to produce the proper Unicode points for these.
- Don't produce a processing error when `\limits` is used without a preceding operator.

MathML Handling

- Prevent align attribute on `<mtable>` from applying to `<mover>/<munder>/<munderover>` elements.
- Ignore `_moz-math-*` attributes in MathML input so they don't appear in MathML output.
- Prevent duplicate `xmlns` attributes in "Show Math As -> MathML".
- Fixed a problem in MathML output where dimensions given to `<mpadded>` with leading +'s could lose the plus and become absolute rather than relative.
- Fix `setTeXclass` for `TeXatom` so that it handles the spacing for relations correctly.
- Add more CSS to isolate `NativeMML` output from page.
- Handle setup of MathPlayer better for IE10, and avoid some IE10 bugs in setting the document namespace for MathML.

Fonts

- Fix a problem where bold-script didn't work properly in STIX fonts.
- Work around Chrome bug with MathJax web fonts that affects some combining characters.
- Remove dependencies of TeX->MathML conversion on the choice of fonts (TeX versus STIX).
- For stretchy characters that don't have a single-character version in the MathJax fonts, make sure they are properly sized when not stretched or stretched to a small size.
- Fix an error with `U+u005E` (^) which caused it to show as a plus when used as a stretchy accent.
- Fix a problem with greek letters in STIX font producing the wrong letter (an offset was off by one).
- Handle more characters in sans-serif-italic and bold-italic STIX fonts.

What's New in MathJax v2.0

MathJax version 2.0 includes many new and improved features, including much better speeds in Internet Explorer, a new AsciiMath input processor, a new *SVG* output processor, support for additional LaTeX commands, and many bug fixes, to name just a few of the changes.

Major speed improvement for HTML-CSS output, particularly in IE

The HTML-CSS output processing was redesigned to avoid the page reflows that were the main source of the speed problem in Internet Explorer 8 and 9. For test pages having between 20 and 50 typeset expressions, we see an 80% reduction in output processing time for IE8, a 50% reduction for IE9, and between 15% and 25% reduction for most other browsers over the corresponding v1.1a times. Since the processing time in v1.1a grows non-linearly in IE, you should see even larger savings for pages with more equations when using v2.0. Forcing IE7 emulation mode is no longer necessary (and indeed is no longer recommended).

Reduced flickering during typsetting

In the past, each expression was displayed as soon as it was typeset, which caused a lot of visual flickering as MathJax processed the page. In v2.0, the output is processed in blocks so that typeset expressions are revealed in groups. This reduces the visual distraction, and also speeds up the processing. The number of equations in a block can be controlled through the `EqnChunk` parameter in the HTML-CSS or SVG block of your configuration. See the *configuration options for HTML-CSS* and *configuration options for SVG* pages for details.

If the page URL includes a hash reference (a link to a particular location within the page), MathJax v2.0 will jump to that location after the page has finished typsetting. (Since the size of the page may have changed due to the mathematical typsetting, that location may no longer be visible on screen, so MathJax moves there when it is done with the initial typesetting.) You can control this behavior with the `positionToHash` parameter in the main section of your configuration. See the *core configuration options* page for details.

Automatic equation numbering of TeX formulas

The TeX input jax now can be configured to add equation numbers (though the default is not to number equations so that existing pages will not change their appearance). This is controlled through the `equationNumbers` section of the `TeX` block of your configuration (see the *equation numbering* section for details). You can request that the numbering follow the AMS-style numbering of environments, or you can request that every displayed equation be numbered. There are now `\label`, `\ref`, and `\eqref` commands to make it easier to link to particular equations within the document.

Automatic line breaking of long displayed equations

MathJax now implements the MathML3 specification for automatic line breaking of displayed equations in its HTML-CSS output. This is disabled by default, but can be enabled via the `linebreaks` section of the `HTML-CSS` or `SVG` block of your configuration (see the *automatic line breaking* section for details). Note that automatic line breaking only applies to displayed equations, not in-line equations, unless they are themselves longer than a line. The algorithm uses the nesting depth, the type of operator, the size of spaces, and other factors to decide on the breakpoints, but it does not know the meaning of the mathematics, and may not choose the optimal breakpoints. We will continue to work on the algorithm as we gain information from its actual use in the field.

New AsciiMath input jax and SVG output jax

MathJax currently processes math in either *TeX* and *LaTeX* format, or *MathML* notation; version 2.0 augments that to include *AsciiMath* notation (see the ASCIIMathML home page for details on this format). This is a notation that is easier for students to use than TeX, and has been requested by the user community. See the *AsciiMath support* page for details.

In addition to the HTML-CSS and Native MathML output available in v1.1, MathJax v2.0 includes an *SVG*-based output jax. This should prove to be more reliable than the HTML-CSS output, as it avoids some CSS, web-font, and printing issues that the HTML-CSS output suffers from, and it currently has no browser-dependent code. The SVG mode even works in some ebook readers (like Apple iBooks and Calibre). See the *output formats* documentation for details.

New combined configuration files

Pre-defined configuration files that include the AsciiMath and SVG processors are now available with MathJax v2.0. These include `AM_HTMLorMML`, `TeX-AMS-MML_SVG`, and `TeX-MML-AM_HTMLorMML`. See the *common configurations* section for details.

MathJax contextual menu now available on mobile devices

MathJax v2.0 provides access to its contextual menu in mobile devices that are based on the WebKit (Safari) and Gecko (Firefox) engines. For Mobile Firefox, the menu is accessed by a tap-and-hold on any expression rendered by MathJax (this is Mobile Firefox's standard method of triggering a contextual menu). In Mobile Safari, use a double-tap-and-hold (you may need to zoom in a bit to be able to accomplish this). This is the first step toward providing a better interface for mobile devices.

Improved support for screen readers

Some issues surrounding the use of screen readers and their interaction with MathPlayer have been resolved in MathJax v2.0. In particular, there are additional menu items that allow the user finer control over some aspects of MathJax's interface that were interfering with some screen readers' ability to properly identify the mathematics. Several stability issues with MathPlayer have also been addressed. In Internet Explorer when MathPlayer is installed, there is now a new contextual menu item to allow you to specify what events are handled by MathJax and what should be handled by MathPlayer. This gives you finer control over MathPlayer's interaction with some screen readers.

Many new TeX additions and enhancements

- New *mhchem* chemistry extension (adds `\ce`, `\cf`, and `\cee` macros)
- New *cancel* extension (adds `\cancel`, `\bcancel`, `\xcancel`, and `\cancelto` macros)
- New *extpfeil* extension (adds more stretchy arrows)
- New *color* extension (makes `\color` work as a switch, as in LaTeX). Adds `\definecolor`, other color models, LaTeX named colors, `\colorbox`, `\fcolorbox`, etc.
- New *begingroup* extension to allow macro definitions to be localized. Adds `\begingroup` and `\endgroup` for isolating macro declarations, and defines `\let`, `\renewenvironment`, `\global`, and `\gdef`.
- New *enclose* extension to give TeX access to `<menclose>` elements. Adds `\enclose{type}[attributes]{math}` macro.

- New *action* extension to give TeX access to `<maction>` elements. Adds `\mathtip{math}{tip}`, `\texttip{math}{tip}`, and `\toggle{math1}{math2}...\endtoggle` macros.
- New `\mmToken{type}[attributes]{text}` macro for producing `<mo>`, `<mi>`, `<mtext>`, and other token MathML elements directly.
- New `\bbox[color;attributes]{math}` macro to add background color, padding, borders, etc.
- New `\middle` macro for stretchy delimiters between `\left` and `\right`.
- New `\label`, `\ref`, and `\eqref` macros for numbered equations.
- Better implementation of `\not` so it produces proper MathML when possible.
- Better implementation of `\dots` that selects `\ldots` or `\cdots` depending on the context.
- Better implementation of `\cases` that automatically uses `\text` on the second entry in each row.
- Safer implementation of `\require` that only allows loading from extensions directory.
- Allow `\newcommand` to provide a default parameter.
- Allow `\\` to take an optional argument that specifies additional space between lines.
- Allow `\\` to be used anywhere (to force a line break), not just in arrays.
- Allow optional alignment parameter for array, aligned, and gathered environments.

See the *TeX support* page for details on these extensions and macros.

Font enhancements

- Work around for the OS X Lion STIX font problem.
- Support for STIX-1.1 fonts (detection of which version you have, and use data appropriate for that).
- New WOFF versions of the web fonts (smaller, so faster to download).
- Data for more stretchy characters in HTML-CSS output.
- Add support for Unicode planes 1 through 10 (not just the Math Alphabet block) in HTML-CSS output.
- Increased timeout for web fonts (since it was switching to image fonts too often, especially for mobile devices).
- Only switch to image fonts if the first web font fails to load (if we can access one, assume we can access them all).
- Allow `<mtext>` elements to use the page font rather than MathJax fonts (optionally). This is controlled by the `mtextFontInerhit` configuration parameter for HTML-CSS and SVG output jax.
- Provide better control over the font used for characters that are not in the MathJax fonts.
- Allow Firefox to use web-based fonts when a local URL uses MathJax from a CDN (in the past it would force image fonts when that was not necessary).

Interface improvements

- The MathJax contextual menu has been reorganized to make it easier to get the source view, and to control the parameters for MathPlayer in IE.
- The MathJax contextual menu is available in mobile devices (see description above).
- Warning messages are issued if you switch renderers to one that is inappropriate for your browser.

- MathJax now starts processing the page on the `DOMContentLoaded` event rather than the page `onload` event (this allows the mathematics to appear sooner).
- Native MathML output is now scaled to better match the surrounding font (like it is for HTML-CSS output).
- Better CSS styling for NativeMML output in Firefox in order to handle `\cal` and other fonts.
- MathML output now (optionally) includes class names to help mark special situations generated by the TeX input jax. (This lets the MathML from the Show Source menu item better reproduce the original TeX output.)
- MathJax now loads the menu and zoom code (if they haven't been loaded already) after the initial typesetting has occured so that they will be available immediately when a user needs those features, but do not delay the initial typesetting of the mathematics.
- For the *tex2jax* preprocessor, the `processClass` can now be used to override the `skipTags` to force a tag that is usually skipped to have its contents be processed.
- The *noErrors* and *noUndefined* extensions can now be disabled via a configuration option (since they are included in many of the combined configuration files). See the *noErrors* and *noUndefined* sections of the *TeX support* page for more information.
- There is a new `MathJax.Hub.setRenderer()` function that can be used to switch the current renderer. See the *MathJax Hub API* documentation for details.
- A user-defined macros is no longer overridden if an extension is loaded that redefines that macro.
- Improved web-font detection reliability.

Important changes from previous versions

- The default renderer for Firefox has been changed from *NativeMML* to *HTML-CSS* (in those configurations that choose between the two). The only browser that defaults to *NativeMML* is now IE with MathPlayer installed. You can configure this to your liking using the *MMLorHTML configuration options*.
- *NativeMML* output will now be selected in IE9 when MathPlayer is present (since IE9 was released the same day as MathJax v1.1a, and there had been problems with IE9 beta releases, we weren't sure if MathPlayer would work with the official release, and so did not select NativeMML by default.)
- The performance improvements in IE8 and IE9 now make it unnecessary to use a `<meta>` tag to force IE7 emulation mode. In fact IE9 in IE9 standards mode now runs faster than IE9 in IE7 standards mode, and IE8 in IE8 standards mode is comparable to IE8 in IE7 standards mode. We now recommend that you use

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

  to obtain the highest emulation mode available in IE, which will be the fastest one for MathJax 2.0.
- The *tex2jax* preprocessor now balances braces when looking for the closing math delimiter. That allows expressions like

```
$y = x^2 \hbox{ when $x > 2$}$
```

  to be properly parsed as a single math expression rather than two separate ones with unbalanced braces. The old behavior can be obtained by setting `balanceBraces` to false in the `tex2jax` block of your configuration. (See the *tex2jax configuration options* for details.)
- If you are hosting your own copy of MathJax on your server, and that copy is being used from pages in a different domain, you will have set up the access control paramters for the font directory to allow Firefox to access the font files properly. Since MathJax 2.0 includes fonts in WOFF format, you will need to include `woff` in you access control declaration for the fonts. E.g., use

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

in the `.htaccess` file for the ``Mathjax/fonts` directory if you are using the Apache web server. See *Notes about shared installations* for details.

- The `\cases` macro now properly places the second column in text mode not math mode. In the past, one needed to use `\text` in the second column to achieve the proper results; pages that did this will still work properly in v2.0. Pages that took advantage of the math mode in the second column will need to be adjusted.
- The `\dots` macro now produces `\ldots` or `\cdots` depending on the context (in the past, `\dots` always produced `\ldots`).
- A one pixel padding has been added above and below HTML-CSS and SVG output so that math on successive lines of a paragraph won't bump into each other.
- There is a new *MathPlayer* submenu of the *Math Settings* menu in the MathJax contextual menu that allows the user to control what events are passed on to MathPlayer. This allows better control for those using assistive devices like screen readers. When menu events are being passed on to MathPlayer, the MathJax menu can be obtained by ALT-clicking on a typeset expression (so the user can still access MathJax's other features).
- In order to improve stability with IE when MathPlayer is installed, MathJax now adds the namespace and object bindings that are needed for MathPlayer at the time that Mathjax is first loaded, rather than waiting for the *NativeMML* output jax to be loaded. Since this is before the configuration information has been obtained, this will happen regardless of whether the *NativeMML* output jax is requested. This means that IE may ask the user to allow MathPlayer to be used, and may show the MathPlayer splash dialog even when MathPlayer is not in the end used by MathJax. Note that this setup can only be performed if MathJax is loaded explicitly as part of the initial web page; if it is injected into the page later by adding a `<script>` tag to the page dynamically, then MathPlayer will be set up when the *NativeMML* jax is loaded as in the past, and some stability issues may occur if events are passed to MathPlayer.
- The MathJax typesetting is now started on `DOMContentLoaded` rather than at the page `onload` event, when possible, so that means MathJax may start typesetting the page earlier than in the past. This should speed up typesetting one pages with lots of images or side-bar content, for example.
- MathJax now attempts to determine whether the page's `onload` event had already occurred, and if it has, it does not try to wait for the `DOMContentLoaded` or `onload` event before doing its initial typeset pass. This means that it is no longer necessary to call `MathJax.Hub.Startup.onload()` by hand if you insert MathJax into the page dynamically (e.g., from a GreaseMonkey script).
- If the page URL includes a hash reference (a link to a particular location within the page), MathJax v2.0 will jump to that location after the page has finished typsetting. Since the size of the page may have changed due to the mathematical typsetting, that location may no longer be visible on screen, so MathJax moves there when it is done with the initial typesetting. You can control this behavior with the `positionToHash` parameter in the main section of your configuration (see *core configuration options*).
- In the event that MathJax is not able to load the configuration file you have specified in the script tag that loads `MathJax.js` via `config=filename`, it will no longer issue the warning message about a missing configuration. The configuration process changed in v1.1, and that message was to help page maintainers update their configurations, but it turns out that for users with slow network connections, MathJax could time out waiting for the configuration file and would issue the warning message in that case, even though the page included the proper configuration. That should no longer occur in MathJax v2.0.

Other enhancements

- Use prioritized lists of callbacks for StartupHooks, MessageHooks, LoadHooks, PreProcessors, and pre- and post-filters on the input jax.
- Updated operator dictionary to correspond to current W3C version.
- Improved browser detection for Gecko and WebKit browsers.
- Make prefilters and postfilters for all input jax, and make them into hook lists rather than a single hook.
- Use `<mi>` rather than `<mo>` for `\sin`, `\cos`, and other such functions, for `\mathop{\rm...}` and `\operatorname`.
- Add `&ApplyFunction;` after `\mathop{}` and other macros that are functions (e.g., `\sin`).
- The `MathJax_Preview` style has been moved from `HTML-CSS/jax.js` to `MathJax.js`, since it is common to all output.
- The *autobold* extension now uses `\boldsymbol` rather than `\bf` so that it will affect more characters.
- Make units of `mu`'s be relative to the scriptlevel (as they are supposed to be).
- Reorganized the event-handling code to make it more modular and reduce redundancy in the different output jax.
- Modified CSS in *NativeMML* output for Firefox to use local copies of the web fonts, if they are available.
- Error messages now have the MathJax contextual menu.
- Better handling of some characters not in the web fonts (remap to locations where they exist, when possible).
- Better choice of accent characters in some cases.
- Better handling of pseudo-scripts (like primes).
- Better sizing of characters introduced by `\unicode{}`, or otherwise outside of the fonts known to MathJax.
- Provide a new extension to handle tagged equations better in *HTML-CSS* output when there are floating elements that might reduce the area available to displayed equations. (See the HTML-CSS extensions section of the *output formats* documentation for detais.)
- Use a text font for `\it` rather than the math italics, so spacing is better.
- Handle italic correction better in *HTML-CSS* output
- Handle `href` attributes better, especially when on `<math>` elements.
- Allow `\sqrt\frac{}{}` without producing an error.

Other bug fixes

- MathPlayer setup changed to prevent crashes.
- Moved remapping of `<mo>` contents to the output jax so that the original contents aren't changed.
- Don't combine `mathvariant` with `fontstyle` or `fontweight` (as per the MathML specification).
- Isolate non-standard attributes on MathML elements so that they don't interfere with the inner workings of MathJax.
- Properly handle width of border and padding in merrors in *HTML-CSS* output.
- Properly handle lower-case Greek better.
- Process weight and style of unknown characters properly.

- Fixed spacing problems with `\cong` in MathJax web fonts .
- Choose better sizes for `\widehat` and `\widetilde`
- Fixed problem with detecting em/ex sizes when uses in mobile devices with small screen widths.
- Fixed MathML output when dimensions of `mu`'s are used in TeX input.
- Better handling of table borders from TeX.
- Fixed some problems with table widths and heights, and spacing.
- Better handling of colored backgrounds in *HTML-CSS* output.
- Handle border and padding CSS styles better in *HTML-CSS* output.
- Fixed multline environment to put tags on bottom row when `TagSide` is set to `right`.
- Force reflow after equations are typeset so that some rendering problems in tables are corrected in Firefox and WebKit browsers.
- Fixed a number of bugs with the size of zoom boxes and the size of their content.
- Have equations with tags zoom into a full-width zoom box to accommodate the tag.
- Fixed positioning problem with zoom boxes in NativeMML mode.
- Don't allow mouse events on zoomed math.
- Fixed `MathJax.Hub.getJaxFor()` and `MathJax.Hub.isJax()` to properly handle elements that are part of an output jax's output (in particular, you can find the element jax from any DOM element in the output).
- Fixed a number of font anomalies (problems in the data files).
- Fixed problem where `<mspace>` with a background color would not always overlay previous items.
- Fixed a problem with colored `<mspace>` elements being too tall in IE/quirks mode.
- Fixed problem where `<mtable>` with `equalrows="true"` would not produce equal height rows.
- Allow `<mpadded>` background color to be specified exactly (i.e., without the 1px padding) when one of its dimensions is given explicitly (or there is no content).
- Avoiding flicker problem with hover zoom trigger in Firefox.
- Fix `\unicode` bug with font names that include spaces.
- Remove internal multiple spaces in token elements as per the MathML specification.
- Work around HTML5 removing namespaces, so that `xmlns:xlink` becomes `xlink` with no namespace, which confuses the XML parsers.
- Fix `MathJax.Message.Set()` and `MathJax.Message.Clear()` so that a delay of 0 is properly handled.
- Produce better MathML for `\bmod`, `\mod`, and `\pmod`.
- Don't allow Safari/Windows to use STIX fonts since it can't access characters in Plane1 (the mathematical alphabets).
- Fix `\thickapprox` to use the correct glyph in *HTML-CSS* output with MathJax web fonts.
- Make style attributes work on `<mstyle>` elements.
- Better handling of border and padding on MathML elements in *HTML-CSS* output.
- Fixed error with size of `\:` space.
- Allow delimiter of `.` on `\genfrac` (it was accidentally rejected).

- Handle AMSmath control sequences with stars better (`\cs{*}` no longer counts as `\cs*`).
- Fixed wrong character number in stretchy data for *U+221A*.
- Fixed `<annotation-xml>` to use the proper scaling in *HTML-CSS* output.
- Fixed a problem with combining characters when they are used as accents.
- Fixed a problem in Firefox with `\mathchoice` when the contents have negative width.
- TeX input jax no longer incorrectly combines `<mo>` elements that have different variants, styles, classes, or id's.
- Fixed the `scriptlevel` when `<munderover>` has base with `movablelimits="true"` in non-display mode.
- Fixed typo in implementation of `SimpleSUPER`.
- Fixed typo in self-closing flag for `<mprescript>` tag.
- Prevent infinite loop if one of the jax fails to load (due to failure to compile or timeout waiting for it to load).
- Fixed a whitespace issue in token elements with IE/quirks mode in the *MathML* input jax.
- Make sure height is above depth when making spaces and rules in *HTML-CSS* and *SVG* output.
- Fixed *HTML-CSS* tooltip to be work properly when a restart occurs within the tooltip.
- Fixed problem with size of colored backgrounds on `<mo>` in some circumstances in *HTML-CSS* output.
- Make `\ulcorner`, etc. use more approprate unicode positions, and remap those positions to the locations in the MathJax web fonts.

Some technical changes

- Break the processing phase into two separate phases to do input processing separately from output processing (they used to be interleaved). This makes it easier to implement forward references for the `\ref` macro.
- Make `Font Preference` menu honor the `imageFont` setting.
- Changed the name of the preview filter commands to `previewFilter` in all preprocessors.
- Make ^ and _ be stretchy even though that isn't in the W3C dictionary.
- Fixed *HTML-CSS* output problem when a multi-character token element has characters taken from multiple fonts.
- Force message text to be black in FontWarnings and configuration warnings.
- Added `Find()` and `IndexOf()` commands to menus to locate menu items.
- Added menu signals for post/unpost and activation of menu items.
- Added signals for typesetting of unknown characters.
- Added signals for zoom/unzoom.
- Added More signals for error conditions.
- Allow preferences to select MathML output for Safari with late enough version.
- Improved *About MathJax* box.
- Have *tex2jax* handle empty delimiter arrays and don't scan page if there is nothing to look for.
- Make delay following a *processing* message configurable and lengthen it to make browser more responsive during typesetting.
- Make thin rules be in pixels to try to improve results in IE (disappearing division lines).

- Mark all output elements as `isMathJax`, so it can be used to identify what elements are part of mathematical output.
- Force MathZoom and MathMenu to wait for the `Begin Styles` message before inserting their styles so when they are included in the combined files, the author can still configure them.
- Add default id's to the jax base object classes.
- Mark top-level math element as having a `texError` when it is one (to make it easier to recognize).
- Have `Update()` method ask ElementJax to determine if it needs updating (which in turn asks the associated input jax).
- Make `Remove()` work for just clearing output (without detaching) if desired.
- Have ElementJax store input and output jax ID's rather than pointers (to help avoid circular references for cleanup purposes).
- Move input/output jax and preprocessor registries from `Hub.config` to Hub itself (they are not user configurable through `Hub.Config`, and so even though they are configurations, they don't belong there).
- Make sure embelished large ops are type `OP` not `ORD` to get spacing right.
- Added `MathJax.HTML.getScript()` to get the contents of a script (needed since it works differently in different browsers).
- Move code that prevents numbers from being treated as a unit for super- and subscripts to the super- and subscript routine in the *TeX* input jax (prevents making changes to `\text{}`, `\hbox{}`, `\href{}`, etc.).
- Make *mml2jax* work better with IE namespaces (IE9 no longer seems to list the `xmlns` entries on the `<html>` element).

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 a 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. a 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.

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/HTML-CSS"],
    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_HTML"></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_HTML"></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.

To use a CDN service, simply load MathJax as follows:

```
<script type="text/javascript"
   src="https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</scrip>
```

Of course, you can load any configuration file that you wish, or use a `text/x-mathajx-config` block to configure MathJax in-line. *More details* are available, if you need them.

The use of `cdn.mathjax.org` is governed by its terms of service, so be sure to read that before linking to a CDN server.

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_HTML"></script>
```

will load the pre-defined `TeX-AMS_HTML` 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 a 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.

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_HTML">
</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.

CHAPTER 4

Advanced Topics

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 plugins, but rendering will be enhanced if high-quality math fonts (e.g., *STIX*) are available to the browser.

MathJax is broken into several different kinds of components: page preprocessors, input processors, output processors, and the MathJax Hub that organizes and connects 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 tags that hold mathematics; for each such tag, it locates an appropriate input jax which it uses to convert the mathematics into an internal form (called an element jax), 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.

Often, and especially with pages that are authored by hand, the mathematics is not stored (initially) within the special tags needed by MathJax, as that would require more notation than the average page author is willing to type. Instead, it is entered in a form that is more natural to the page author, for example, using the standard TeX math delimiters `$...$` and `$$...$$` to indicate what part of the document is to be typeset as mathematics. In this case, MathJax can run a preprocessor to locate the math delimiters and replace them by the special tags that it uses to mark the formulas. There are preprocessors for *TeX notation*, *MathML notation*, *AsciiMath notation* and the *jsMath notation* that uses *span* and *div* tags.

For pages that are constructed programmatically, such as HTML pages that result from running a processor on text in some other format (e.g., pages produced from Markdown documents, or via programs like *tex4ht*), it would be best to use MathJax's special tags directly, as described below, rather than having MathJax run another preprocessor. This will speed up the final display of the mathematics, since the extra preprocessing step would not be needed. It also avoids the conflict between the use of the less-than sign, <, in mathematics and as an HTML special character (that starts an HTML tag), and several other issues involved in having the mathematics directly in the text of the page (see the documentation on the various input jax for more details on this).

How mathematics is stored in the page

In order to identify mathematics in the page, MathJax uses special `<script>` tags to enclose the mathematics. This is done because such tags can be located easily, and because their content is not further processed by the browser; for example, less-than signs can be used as they are in mathematics, without worrying about them being mistaken for the beginnings of HTML tags. One may also consider the math notation as a form of "script" for the mathematics, so a `<script>` tag makes at least some sense for storing the math.

Each `<script>` tag has a `type` attribute that identifies the kind of script that the tag contains. The usual (and default) value is `type="text/javascript"`, and when a script has this type, the browser executes the script as a javascript program. MathJax, however, uses the type *math/tex* to identify mathematics in the TeX and LaTeX notation, *math/mml* for mathematics in MathML notation, and *math/asciimath* for mathematics in AsciiMath notation. When the *tex2jax*, *mml2jax*, or *asciimath2jax* preprocessors run, they create `<script>` tags with these types so that MathJax can process them when it runs its main typesetting pass.

For example,

```
<script type="math/tex">x+\sqrt{1-x^2}</script>
```

represents an in-line equation in TeX notation, and

```
<script type="math/tex; mode=display">
  \sum_{n=1}^\infty {1\over n^2} = {\pi^2\over 6}
</script>
```

is a displayed TeX equation.

Alternatively, using MathML notation, you could use

```
<script type="math/mml">
  <math>
    <mi>x</mi>
    <mo>+</mo>
    <msqrt>
      <mn>1</mn>
      <mo>&#x2212;<!--  --></mo>
      <msup>
        <mi>x</mi>
        <mn>2</mn>
      </msup>
    </msqrt>
  </math>
</script>
```

for in-line math, or

```
<script type="math/mml">
  <math display="block">
    <mrow>
      <munderover>
        <mo>&#x2211;<!--  --></mo>
        <mrow>
          <mi>n</mi>
          <mo>=</mo>
          <mn>1</mn>
        </mrow>
        <mi mathvariant="normal">&#x221E;<!-- ∞ --></mi>
      </munderover>
    </mrow>
```

```
    <mrow>
      <mfrac>
        <mn>1</mn>
        <msup>
          <mi>n</mi>
          <mn>2</mn>
        </msup>
      </mfrac>
    </mrow>
    <mo>=</mo>
    <mrow>
      <mfrac>
        <msup>
          <mi>&#x03C0;<!-- π --></mi>
          <mn>2</mn>
        </msup>
        <mn>6</mn>
      </mfrac>
    </mrow>
  </math>
</script>
```

for displayed equations in MathML notation. As other input jax are created, they will use other types to identify the mathematics they can process.

Page authors can use one of MathJax's preprocessors to convert from math delimiters that are more natural for the author to type (e.g., TeX math delimiters like `$$...$$`) to MathJax's `<script>` format. Blog and wiki software could extend from their own markup languages to include math delimiters, which they could convert to MathJax's `<script>` format automatically.

Note, however, that Internet Explorer has a bug that causes it to remove the space before a `<script>` tag if there is also a space after it, which can cause serious spacing problems with in-line math in Internet Explorer. There are three possible solutions to this in MathJax. The recommended way is to use a math preview (an element with class `MathJax_Preview`) that is non-empty and comes right before the `<script>` tag. Its contents can be just the word `[math]`, so it does not have to be specific to the mathematics script that follows; it just has to be non-empty (though it could have its style set to `display:none`). See also the `preJax` and `postJax` options in the *Core Configuration Options* document for another approach.

The components of MathJax

The main components of MathJax are its preprocessors, its input and output jax, and the MathJax Hub, which coordinates the actions of the other components.

Input jax are associated with the different script types (like *`math/tex`* or *`math/mml`*) and the mapping of a particular type to a particular jax is made when the various jax register their abilities with the MathJax Hub at configuration time. For example, the MathML input jax registers the *`math/mml`* type, so MathJax will know to call the MathML input jax when it sees math elements of that type. The role of the input jax is to convert the math notation entered by the author into the internal format used by MathJax (called an *element jax*). This internal format is essentially MathML (represented as JavaScript objects), so an input jax acts as a translator into MathML.

Output jax convert that internal element jax format into a specific output format. For example, the NativeMML output jax inserts MathML tags into the page to represent the mathematics, while the HTML-CSS output jax uses HTML with CSS styling to lay out the mathematics so that it can be displayed even in browsers that don't understand MathML. MathJax also has an *SVG* output jax that will render the mathematics using scalable vector grtaphics. Output jax could be produced that render the mathematics using HTML5 canvas elements, for example, or that speak an equation for blind users. The MathJax contextual menu can be used to switch between the output jax that are available.

Each input and output jax has a small configuration file that is loaded when that input jax is included in the *jax* array in the MathJax configuration, and a larger file that implements the core functionality of that particular jax. The latter file is loaded the first time the jax is needed by MathJax to process some mathematics. Most of the combined configuration files include only the small configuration portion for the input and output jax, making the configuraiton file smaller and faster to load for those pages that don't actually incldue mathematics; the combined configurations that end in `-full` include both parts of the jax, so there is no delay when the math is to be rendered, but at the expense of a larger initial download.

The **MathJax Hub** keeps track of the internal representations of the various mathematical equations on the page, and can be queried to obtain information about those equations. For example, one can obtain a list of all the math elements on the page, or look up a particular one, or find all the elements with a given input format, and so on. In a dynamically generated web page, an equation where the source mathematics has changed can be asked to re-render itself, or if a new paragraph is generated that might include mathematics, MathJax can be asked to process the equations it contains.

The Hub also manages issues concerning mouse events and other user interaction with the equation itself. Parts of equations can be made active so that mouse clicks cause event handlers to run, or activate hyperlinks to other pages, and so on, making the mathematics as dynamic as the rest of the page.

The MathJax Startup Sequence

When you load `MathJax.js` into a web page, it configures itself and immediately begins loading the components it needs. As MathJax starts up, it uses its *signaling mechanism* to indicate the actions that it is taking so that MathJax extensions can tie into the initialization process, and so other applications within the page can synchronize their actions with MathJax.

The startup process performs the following actions:

- It creates the `MathJax` variable, and defines the following subsystems:
  - `MathJax.Object` (object-oriented programming model)
  - `MathJax.Callback` (callbacks, signals, and queues)
  - `MathJax.Ajax` (file-loading and style-creation code)
  - `MathJax.HTML` (support code for creating HTML elements)
  - `MathJax.Localization` (alternative language support)
  - `MathJax.Message` (manages the menu line in the lower left)
  - `MathJax.Hub` (the core MathJax functions)
- It then creates the base `MathJax.InputJax`, `MathJax.OutputJax`, and `MathJax.ElementJax` objects.
- MathJax sets up the default configuration, and creates the signal objects used for the startup and hub actions.
- MathJax locates the `<script>` tag that loaded the `MathJax.js` file, and sets the `MathJax.Hub.config.root` value to reflect the location of the MathJax root directory.
- MathJax determines the browser being used and its version. It sets up the `MathJax.Hub.Browser` object, which includes the browser name and version, plus `isMac`, `isPC`, `isMSIE`, and so on.
- MathJax executes the `AuthorInit()` function specified from in-line `MathJax = {...}` configuration.
- MathJax sets up the `MathJax.Hub.queue` command queue, and populates it with the commands MathJax runs at startup. This includes creating the `MathJax.Hub.Startup.onload` onload handler that is used to synchronize MathJax's action with the loading of the page.

Once the `MathJax.Hub.queue` is created, the following actions are pushed into the queue:

1. Post the `Begin` startup signal
2. Perform the configuration actions:
    - Post the `Begin Config` startup signal
    - Load any configuration files specified via `config=` as a script parameter
    - Perform author configuration from in-line `MathJax = {...}`
    - Execute the content of the `<script>` that loaded MathJax, if it is not empty
    - Wait for the `delayStartupUntil` condition to be met, if one was specified
    - Execute any `text/x-mathjax-config` script blocks
    - load the files listed in the `MathJax.Hub.config.config` array
    - Post the `End Config` startup signal
3. Load the cookie values:
    - Post the `Begin Cookie` startup signal
    - Load the menu cookie values
    - Use the cookie to set the renderer, if it is set
    - Post the `End Cookie` startup signal
4. Define the MathJax styles:
    - Post the `Begin Styles` startup signal
    - Load the stylesheet files from the `MathJax.Hub.config.stylesheets` array
    - Define the stylesheet described in `MathJax.Hub.config.styles`
    - Post the `End Styles` startup signal
5. Initialize the Message system (the grey information box in the lower left)
6. Load the jax configuration files:
    - Post the `Begin Jax` startup signal
    - Load the jax config files from the `MathJax.Hub.config.jax` array
        - The jax will register themselves when they are loaded
    - Post the `End Jax` startup signal
7. Load the extension files:
    - Post the `Begin Extensions` startup signal
    - Load the files from the `MathJax.Hub.config.extensions` array
        - Most extensions will post a `[name] Ready` or `Extension [name] Ready` startup message when they are loaded (where `[name]` is the name of the extension)
    - Post the `End Extensions` startup signal
8. Set the MathJax menu's renderer value based on the jax that have been loaded
9. Wait for the onload handler to fire (in MathJax v2.0 this can occur on the `DOMContentLoaded` event rather than the page's `onload` event, so processing of mathematics can start earlier)
10. Set `MathJax.isReady` to `true`
11. Perform the typesetting pass (preprocessors and processors)

- Post the `Begin Typeset` startup signal
- Post the `Begin PreProcess` hub signal
- Run the registered preprocessors
- Post the `End PreProcess` hub signal
- Clear the hub signal history
- Post the `Begin Process` hub signal
- Process the math script elements on the page
  - There are a number of Hub signals generated during math processing, including a signal that a `Math` action is starting (with a parameter indicating what action that is), `Begin` and `End Math Input` messages, and `Begin` and `End Math Output` signals.
  - Each new math element generates a `New Math` hub signal with the math element's ID
- Post the `End Process` hub signal
- Post the `End Typeset` startup signal

12. Jump to the location specified in the URL's hash reference, if any.
13. Initiate timers to load the zoom and menu code, if it hasn't already been loading in the configuration (so it will be ready when the user needs it).
14. Post the `End` startup signal

The loading of the jax and extensions in steps 6 and 7 are now done in parallel, rather than sequentially. That is, all the jax and extensions are requested simultaneously, so they load concurrently. That means they can load in any order, and that the begin and end signals for the jax and extensions can be intermixed. (In general, you will get *Begin Jax* followed by *Begin Extensions*, but the order of *End Jax* and *End Extensions* will depend on the files being loaded.) Both 6 and 7 must complete, however, before 8 will be performed.

See the test/sample-signals.html file to see the signals in action.

Synchronizing your code with MathJax

MathJax performs much of its activity asynchronously, meaning that the calls that you make to initiate these actions will return before the actions are completed, and your code will continue to run even though the actions have not been finished (and may not even be started yet). Actions such as loading files, loading web-based fonts, and creating stylesheets all happen asynchronously within the browser, and since JavaScript has no method of halting a program while waiting for an action to complete, synchronizing your code with these types of actions is made much more difficult. MathJax uses three mechanisms to overcome this language shortcoming: callbacks, queues, and signals.

Callbacks are functions that are called when an action is completed, so that your code can continue where it left off when the action was initiated. Rather than have a single routine that initiates an action, waits for it to complete, and then goes on, you break the function into two parts: a first part that sets up and initiates the action, and a second that runs after the action is finished. Callbacks are similar to event handlers that you attach to DOM elements, and are called when a certain action occurs. See the *Callback Object* reference page for details of how to specify a callback.

Queues are MathJax's means of synchronizing actions that must be performed sequentially, even when they involve asynchronous events like loading files or dynamically creating stylesheets. The actions that you put in the queue are *Callback* objects that will be performed in sequence, with MathJax handling the linking of one action to the next. MathJax maintains a master queue that you can use to synchronize with MathJax, but you can also create your own private queues for actions that need to be synchronized with each other, but not to MathJax as a whole. See the *Queue Object* reference page for more details.

Signals are another means of synchronizing your own code with MathJax. Many of the important actions that MathJax takes (like typesetting new math on the page, or loading an external component) are "announced" by posting a message to a special object called a *Signal*. Your code can register an interest in receiving one or more of these signals by providing a callback to be called when the signal is posted. When the signal arrives, MathJax will call your code. This works somewhat like an event handler, except that many different types of events can go through the same signal, and the signals have a "memory", meaning that if you register an interest in a particular type of signal and that signal has already occurred, you will be told about the past occurrences as well as any future ones. See the *Signal Object* reference page for more details. See also the test/sample-signals.html file in the MathJax `test` directory for a working example of using signals.

Each of these is explained in more detail in the links below:

Using Callbacks

A "callback" is a function that MathJax calls when it completes an action that may occur asynchronously (like loading a file). Many of MathJax's functions operate asynchronously, and MathJax uses callbacks to allow you to synchronize your code with the action of those functions. The *MathJax.Callback* structure manages these callbacks. Callbacks can include not only a function to call, but also data to be passed to the function, and an object to act as the JavaScript *this* value in the resulting call (i.e., the object on which the callback is to execute).

Callbacks can be collected into *Queues* where the callbacks will be processed in order, with later callbacks waiting until previous ones have completed before they are called. They are also used with *Signals* as the means of receiving information about the signals as they occur.

A number of methods in *MathJax.Hub* and *MathJax.Ajax* accept callback specifications as arguments and return callback structures. These routines always will return a callback even when none was specified in the arguments, and in that case, the callback is a "do nothing" callback. The reason for this is so that the resulting callback can be used in a *MathJax.Callback.Queue* for synchronization purposes, so that the actions following it in the queue will not be performed until after the callback has been fired.

For example, the `MathJax.Ajax.Require()` method can be used to load external files, and it returns a callback that is called when the file has been loaded and executed. If you want to load several files and wait for them all to be loaded before performing some action, you can create a *Queue* into which you push the results of the `MathJax.Ajax.Require()` calls, and then push a callback for the action. The final action will not be performed until all the file-load callbacks (which precede it in the queue) have been called; i.e., the action will not occur until all the files are loaded.

Specifying a Callback

Callbacks can be specified in a number of different ways, depending on the functionality that is required of the callback. The easiest case is to simply provide a function to be called, but it is also possible to include data to pass to the function when it is called, and to specify the object that will be used as *this* when the function is called.

For example, the `MathJax.Ajax.Require()` method can accept a callback as its second argument (it will be called when the file given as the first argument is loaded and executed). So you can call

```
MathJax.Ajax.Require("[MathJax]/config/myConfig.js",function () {
  alert("My configuration file is loaded");
});
```

and an alert will appear when the file is loaded. An example of passing arguments to the callback function includes the following:

```
function loadHook (x) {alert("loadHook: "+x)}
MathJax.Ajax.Require("[MathJax]/config/myConfig.js",[loadHook,"myConfig"]);
```

Here, the `loadHook()` function accepts one argument and generates an alert that includes the value passed to it. The callback in the `MathJax.Ajax.Require()` call is `[loadHook,"myConfig"]`, which means that (the equivalent of) `loadHook("myConfig")` will be performed when the file is loaded. The result should be an alert with the text *loadHook: myConfig*.

The callback for the `MathJax.Ajax.Require()` method actually gets called with a status value, in addition to any parameters already included in the callback specification, that indicates whether the file loaded successfully, or failed for some reason (perhaps the file couldn't be found, or it failed to compile and run). So you could use

```
MathJax.Ajax.Require("[MathJax]/config/myConfig.js",function (status) {
  if (status === MathJax.Ajax.STATUS.OK) {
    alert("My configuration file is loaded");
  } else {
    alert("My configuration file failed to load!");
  }
});
```

to check if the file loaded properly. With additional parameters, the example might be

```
function loadHook (x,status) {alert("loadHook: "+x+" has status "+status)}
MathJax.Ajax.Require("[MathJax]/config/myConfig.js",[loadHook,"myConfig"]);
```

Note that the parameters given in the callback specification are used first, and then additional parameters from the call to the callback come afterward.

Callbacks to Object Methods

When you use a method of a JavaScript object, a special variable called *this* is defined that refers to the object whose method is being called. It allows you to access other methods or properties of the object without knowing explicitly where the object is stored.

For example,

```
var aPerson = {
  firstname: "John",
  lastname: "Smith",
  showName: function () {alert(this.firstname+" "+this.lastname)}
};
```

creates an object that contains three items, a *firstname*, and *lastname*, and a method that shows the person's full name in an alert. So `aPerson.showName()` would cause an alert with the text `John Smith` to appear. Note, however that this only works if the method is called as `aPerson.showName()`; if instead you did

```
var f = aPerson.showName;  // assign f the function from aPerson
f();                       // and call the function
```

the association of the function with the data in `aPerson` is lost, and the alert will probably show `undefined undefined`. (In this case, `f` will be called with `this` set to the `window` variable, and so `this.firstname` and `this.lastname` will refer to undefined values.)

Because of this, it is difficult to use an object's method as a callback if you refer to it as a function directly. For example,

```
var aFile = {
  name: "[MathJax]/config/myConfig.js",
  onload: function (status) {
    alert(this.name+" is loaded with status "+status);
```

```
  }
};

MathJax.Ajax.Require(aFile.name,aFile.onload);
```

would produce an alert indicating that "undefined" was loaded with a particular status. That is because `aFile.onload` is a reference to the *onload* method, which is just a function, and the association with the *aFile* object is lost. One could do

```
MathJax.Ajax.Require(aFile.name,function (status) {aFile.onload(status)});
```

but that seems needlessly verbose, and it produces a closure when one is not really needed. Instead, MathJax provides an alternative specification for a callback that allows you to specify both the method and the object it comes from:

```
MathJax.Ajax.Require(aFile.name,["onload",aFile]);
```

This requests that the callback should call `aFile.onload` as the function, which will maintain the connection between `aFile` and its method, thus preserving the correct value for *this* within the method.

As in the previous cases, you can pass parameters to the method as well by including them in the array that specifies the callback:

```
MathJax.Ajax.Require("filename",["method",object,arg1,arg2,...]);
```

This approach is useful when you are pushing a callback for one of MathJax's Hub routines into the MathJax processing queue. For example,

```
MathJax.Hub.Queue(["Typeset",MathJax.Hub,"MathDiv"]);
```

pushes the equivalent of `MathJax.Hub.Typeset("MathDiv")` into the processing queue.

See the *Callback Object* reference pages for more information about the valid methods of specifying a callback.

Creating a Callback Explicitly

When you call a method that accepts a callback, you usually pass it a callback specification (like in the examples above), which *describes* a callback (the method will create the actual *Callback* object, and return that to you as its return value). You don't usually create *Callback* objects directly yourself.

There are times, however, when you may wish to create a callback object for use with functions that don't create callbacks for you. For example, the `setTimeout()` function can take a function as its argument, and you may want that function to be a method of an object, and would run into the problem described in the previous section if you simply passed the object's method to `setTimeout()`. Or you might want to pass an argument to the function called by `setTimeout()`. (Altough the `setTimeout()` function can accept additional arguements that are supposed to be passed on to the code when it is called, some versions of Internet Explorer do not implement that feature, so you can't rely on it.) You can use a *Callback* object to do this, and the `MathJax.Callback()` method will create one for you. For example,

```
function f(x) {alert("x = "+x)}
setTimeout(MathJax.Callback([f,"Hello World!"]),500);
```

would create a callback that calls `f("Hello World!")`, and schedules it to be called in half a second.

Using Queues

The *callback queue* is one of MathJax's main tools for synchronizing its actions, both internally, and with external programs, like javascript code that you may write as part of dynamic web pages. Because many actions in MathJax (like loading files) operate asynchronously, MathJax needs a way to coordinate those actions so that they occur in the right order. The *MathJax.Callback.Queue* object provides that mechanism.

A *callback queue* is a list of commands that will be performed one at a time, in order. If the return value of one of the commands is a *Callback* object, processing is suspended until that callback is called, and then processing of the commands is resumed. In this way, if a command starts an asynchronous operation like loading a file, it can return the callback for that file-load operation and the queue will wait until the file has loaded before continuing. Thus a queue can be used to guarantee that commands don't get performed until other ones are known to be finished, even if those commands usually operate asynchronously.

Constructing Queues

A queue is created via the `MathJax.Callback.Queue()` command, which returns a *MathJax.Callback.Queue* object. The queue itself consists of a series of commands given as callback specifications (see *Using Callbacks* for details on callbacks), which allow you to provide functions (together with their arguments) to be executed. You can provide the collection of callback specifications when the queue is created by passing them as arguments to `MathJax.Callback.Queue()`, or you can create an empty queue to which commands are added later. Once a *MathJax.Callback.Queue* object is created, you can push additional callbacks on the end of the queue; if the queue is empty, the command will be performed immediately, while if the queue is waiting for another command to complete, the new command will be queued for later processing.

For example,

```
function f(x) {alert(x)}
var queue = MathJax.Callback.Queue([f, 15], [f, 10], [f, 5]);
queue.Push([f, 0]);
```

would create a queue containing three commands, each calling the function `f` with a different input, that are performed in order. A fourth command is then added to the queue, to be performed after the other three. In this case, the result will be four alerts, the first with the number 15, the second with 10, the third with 5 and the fourth with 0. Of course `f` is not a function that operates asynchronously, so it would have been easier to just call `f` four times directly. The power of the queue comes from calling commands that could operate asynchronously. For example:

```
function f(x) {alert(x)}
MathJax.Callback.Queue(
  [f, 1],
  ["Require", MathJax.Ajax, "[MathJax]/extensions/AMSmath.js"],
  [f, 2]
);
```

Here, the command `MathJax.Ajax.Require("[MathJax]/extensions/AMSmath.js")` is queued between two calls to `f`. The first call to `f(1)` will be made immediately, then the `MathJax.Ajax.Require()` statement will be performed. Since the `Require` method loads a file, it operates asynchronously, and its return value is a *MathJax.Callback* object that will be called when the file is loaded. The call to `f(2)` will not be made until that callback is performed, effectively synchronizing the second call to `f` with the completion of the file loading. This is equivalent to

```
f(1);
MathJax.Ajax.Require("[MathJax]/extensions/AMSmath.js", [f, 2]);
```

since the `Require()` command allows you to specify a (single) callback to be performed on the completion of the file load. Note, however, that the queue could be used to synchronize several file loads along with multiple function

calls, so is more flexible.

For example,

```
MathJax.Callback.Queue(
  ["Require", MathJax.Ajax, "[MathJax]/extensions/AMSmath.js"],
  [f, 1],
  ["Require", MathJax.Ajax, "[MathJax]/config/local/AMSmathAdditions.js"],
  [f, 2]
);
```

would load the AMSmath extension, then call `f(1)` then load the local AMSmath modifications, and then call `f(2)`, with each action waiting for the previous one to complete before being performed itself.

Callbacks versus Callback Specifications

If one of the callback specifications is an actual callback object itself, then the queue will wait for that action to be performed before proceeding. For example,

```
 MathJax.Callback.Queue(
     [f, 1],
     MathJax.Ajax.Require("[MathJax]/extensions/AMSmath.js"),
     [f, 2],
);
```

starts the loading of the AMSmath extension before the queue is created, and then creates the queue containing the call to `f`, the callback for the file load, and the second call to `f`. The queue performs `f(1)`, waits for the file load callback to be called, and then calls `f(2)`. The difference between this and the second example above is that, in this example the file load is started before the queue is even created, so the file is potentially loaded and executed before the call to `f(1)`, while in the example above, the file load is guaranteed not to begin until after `f(1)` is executed.

As a further example, consider

```
MathJax.Callback.Queue(
  MathJax.Ajax.Require("[MathJax]/extensions/AMSmath.js"),
  [f, 1],
  MathJax.Ajax.Require("[MathJax]/config/local/AMSmathAdditions.js"),
  [f, 2]
);
```

in comparison to the example above that uses `["Require", MathJax.Ajax, "[MathJax]/extensions/AMSmath.js"]` and `["Require", MathJax.Ajax, "[MathJax]/config/local/AMSmathAdditions.js"]` instead. In that example, `AMSmath.js` is loaded, then `f(1)` is called, then the local additions are loaded, then `f(2)` is called.

Here, however, both file loads are started before the queue is created, and are operating in parallel (rather than sequentially as in the earlier example). It is possible for the loading of the local additions to complete before the AMSmath extension is loaded in this case, which was guaranteed **not** to happen in the other example. Note, however, that `f(1)` is guaranteed not to be performed until after the AMSmath extensions load, and `f(2)` will not occur until after both files are loaded.

In this way, it is possible to start asynchronous loading of several files simultaneously, and wait until all of them are loaded (in whatever order) to perform some command. For instance,

```
MathJax.Callback.Queue(
    MathJax.Ajax.Require("file1.js"),
    MathJax.Ajax.Require("file2.js"),
    MathJax.Ajax.Require("file3.js"),
```

```
    MathJax.Ajax.Require("file4.js"),
    [f, "all done"]
);
```

starts four files loading all at once, and waits for all four to complete before calling `f("all done")`. The order in which they complete is immaterial, and they all are being requested simultaneously.

The MathJax Processing Queue

MathJax uses a queue stored as `MathJax.Hub.queue` to regulate its own actions so that they operate in the right order even when some of them include asynchronous operations. You can take advantage of that queue when you make calls to MathJax methods that need to be synchronized with the other actions taken by MathJax. It may not always be apparent, however, which methods fall into that category.

The main source of asynchronous actions in MathJax is the loading of external files, so any action that may cause a file to be loaded may act asynchronously. Many important actions do so, including some that you might not expect; e.g., typesetting mathematics can cause files to be loaded. This is because some TeX commands, for example, are rare enough that they are not included in the core TeX input processor, but instead are defined in extensions that are loaded automatically when needed. The typesetting of an expression containing one of these TeX commands can cause the typesetting process to be suspended while the file is loaded, and then restarted when the extension has become available.

As a result, any call to `MathJax.Hub.Typeset()` (or `MathJax.Hub.Process()`, or `MathJax.Hub.Update()`, etc.) could return long before the mathematics is actually typeset, and the rest of your code may run before the mathematics is available. If you have code that relies on the mathematics being visible on screen, you will need to break that out into a separate operation that is synchronized with the typesetting via the MathJax queue.

Furthermore, your own typesetting calls may need to wait for file loading to occur that is already underway, so even if you don't need to access the mathematics after it is typeset, you may still need to queue the typeset command in order to make sure it is properly synchronized with *previous* typeset calls. For instance, if an earlier call started loading an extension and you start another typeset call before that extension is fully loaded, MathJax's internal state may be in flux, and it may not be prepared to handle another typeset operation yet. This is even more important if you are using other libraries that may call MathJax, in which case your code may not be aware of the state that MathJax is in.

For these reasons, it is always best to perform typesetting operations through the MathJax queue, and the same goes for any other action that could cause files to load. A good rule of thumb is that, if a MathJax function includes a callback argument, that function may operate asynchronously; you should use the MathJax queue to perform it and any actions that rely on its results.

To place an action in the MathJax queue, use the `MathJax.Hub.Queue()` command. For example

```
MathJax.Hub.Queue(["Typeset",MathJax.Hub,"MathDiv"]);
```

would queue the command `MathJax.Hub.Typeset("MathDiv")`, causing the contents of the DOM element with *id* equal to `MathDiv` to be typeset.

One of the uses of the MathJax queue is to allow you to synchronize an action with the startup process for MathJax. If you want to have a function performed after MathJax has become completely set up (and performed its initial typesetting of the page), you can push it onto the `MathJax.Hub.queue` so that it won't be performed until MathJax finishes everything it has queued when it was loaded. For example,

```
<script type="text/javascript" src="/MathJax/MathJax.js"></script>
<script>
  MathJax.Hub.Queue(function () {
    // ... your startup commands here ...
  });
</script>
```

Using Signals

Because much of MathJax operates asynchronously, it is important for MathJax to be able to indicate to other components operating on the page that certain actions have been taken. For example, as MathJax is starting up, it loads external files such as its configuration files and the various input and output *jax* that are used on the page. This means that MathJax may not be ready to run until well after the `<script>` tag that loads `MathJax.js` has executed. If another component on the page needs to call MathJax to process some mathematics, it will need to know when MathJax is ready to do that. Thus MathJax needs a way to signal other components that it is initialized and ready to process mathematics. Other events that might need to be signaled include the appearance of newly processed mathematics on the web page, the loading of a new extension, and so on.

The mechanism provided by MathJax for handling this type of communication is the *Callback Signal*. The *Callback Signal* object provides a standardized mechanism for sending and receiving messages between MathJax and other code on the page. A signal acts like a mailbox where MathJax places messages for others to read. Those interested in seeing the messages can register an interest in receiving a given signal, and when MathJax posts a message on that signal, all the interested parties will be notified. No new posts to the signal will be allowed until everyone who is listening to the signal has had a chance to receive the first one. If a signal causes a listener to begin an asynchronous operation (such as loading a file), the listener can indicate that its reply to the signal is going to be delayed, and MathJax will wait until the asynchronous action is complete before allowing additional messages to be posted to this signal. In this way, posting a signal may itself be an asynchronous action.

The posts to a signal are cached so that if a new listener expresses an interest in the signal, it will receive all the past posts as well as any future ones. For example, if a component on the page needs to know when MathJax is set up, it can express an interest in the startup signal's `End` message. If MathJax is not yet set up, the component will be signaled when MathJax is ready to begin, but if MathJax is already set up, the component will receive the `End` message immediately, since that message was cached and is available to any new listeners. In this way, signals can be used to pass messages without worrying about the timing of when the signaler and listener are ready to send or receive signals: a listener will receive messages even if it starts listening after they were sent.

One way that MathJax makes use of this feature is in configuring its various extensions. The extension may not be loaded when the user's configuration code runs, so the configuration code can't modify the extension because it isn't there yet. Fortunately, most extensions signal when they are loaded and initialized via an `Extension [name] Ready` message, or just `[name] Ready`, so the configuration code can implement a listener for that message, and have the listener perform the configuration when the message arrives. But even if the extension *has* already been loaded, this will still work, because the listener will receive the ready signal even if it has already been posted. In this way, listening for signals is a robust method of synchronizing code components no matter when they are loaded and run.

In some cases, it may be inappropriate for a new listener to receive past messages that were sent to a signal object. There are two ways to handle this: first, a new listener can indicate that it doesn't want to hear old messages when it attaches itself to a signal object. The sender can also indicate that past messages are not appropriate for new listeners. It does this by clearing the message history so that new listeners have no old posts to hear.

The actual message passed along by the signal can be anything, but is frequently a string constant indicating the message value. It could also be a JavaScript array containing data, or an object containing *key:value* pairs. All the listeners receive the data as part of the message, and can act on it in whatever ways they see fit.

Creating a Listener

MathJax maintains two separate pre-defined signal channels: the *startup signal* and the *processing signal* (or the *hub signal*). The startup signal is where the messages about different components starting up and becoming ready appear. The processing signal is where the messages are sent about processing mathematics, like the `New Math` messages

for when newly typeset mathematics appears on the page. The latter is cleared when a new processing pass is started (so messages from past processing runs are not kept).

The easiest way to create a listener is to use either `MathJax.Hub.Register.StartupHook()` or `MathJax.Hub.Register.MessageHook()`. The first sets a listener on the startup signal, and the latter on the hub processing signal. You specify the message you want to listen for, and a callback to be called when it arrives. For example

```
MathJax.Hub.Register.StartupHook("TeX Jax Ready ",function () {
  alert("The TeX input jax is loaded and ready!");
});
```

See the *MathJax Startup Sequence* page for details of the messages sent during startup. See also the test/sample-signals.html file (and its source) for examples of using signals. This example lists all the signals that occur while MathJax is processing that page, so it gives useful information about the details of the signals produced by various components.

In this example, the listener starts loading an extra configuration file (from the same directory as the web page). Since it returns the callback from that request, the signal processing will wait until that file is completely loaded before it continues; that is, the configuration process is suspended until the extra configuration file has loaded.

```
MathJax.Hub.Register.StartupHook("Begin Config",
  function () {return MathJax.Ajax.Require("myConfig.js")}
);
```

Here is an example that produces an alert each time new mathematics is typeset on the page. The message includes the DOM *id* of the element on the page that contains the newly typeset mathematics as its second element, so this listener locates the `<script>` tag for the math, and displays the original source mathematics for it.

```
MathJax.Hub.Register.MessageHook("New Math", function (message) {
  var script = MathJax.Hub.getJaxFor(message[1]).SourceElement();
  alert(message.join(" ")+": '"+script.text+"'");
})
```

Listening for All Messages

If you want to process *every* message that passes through a signal channel, you can do that by registering an interest in the signal rather than registering a message hook. You do this by calling the signal's *`Interest()`* method, as in the following example.

```
MathJax.Hub.Startup.signal.Interest(
  function (message) {alert("Startup: "+message)}
);
MathJax.Hub.signal.Interest(
  function (message) {alert("Hub: "+message)}
);
```

This will cause an alert for every signal that MathJax produces. You probably don't want to try this out, since it will produce a *lot* of them; instead, use the test/sample-signals.html file, which displays them in the web page.

See the *Signal Object* reference page for details on the structure and methods of the signal object.

Loading MathJax Dynamically

MathJax is designed to be included via a `<script>` tag in the `<head>` section of your HTML document, and it does rely on being part of the original document in that it uses an `onload` or `DOMContentLoaded` event handler

to synchronize its actions with the loading of the page. If you wish to insert MathJax into a document after it has been loaded, that will normally occur *after* the page's `onload` handler has fired, and prior to version 2.0, MathJax had to be told not to wait for the page `onload` event by calling `MathJax.Hub.Startup.onload()` by hand. That is no longer necessary, as MathJax v2.0 detects whether the page is already available and when it is, it processes it immediately rather than waiting for an event that has already happened.

Here is an example of how to load and configure MathJax dynamically:

```
(function () {
  var script = document.createElement("script");
  script.type = "text/javascript";
  script.src  = "https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML";
  document.getElementsByTagName("head")[0].appendChild(script);
})();
```

If you need to provide in-line configuration, you can do that using a MathJax's configuration script:

```
(function () {
  var head = document.getElementsByTagName("head")[0], script;
  script = document.createElement("script");
  script.type = "text/x-mathjax-config";
  script[(window.opera ? "innerHTML" : "text")] =
    "MathJax.Hub.Config({\n" +
    "  tex2jax: { inlineMath: [['$','$'], ['\\\\(','\\\\)']] }\n" +
    "});"
  head.appendChild(script);
  script = document.createElement("script");
  script.type = "text/javascript";
  script.src  = "https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML";
  head.appendChild(script);
})();
```

You can adjust the configuration to your needs, but be careful to get the commas right, as Internet Explorer 6 and 7 will not tolerate an extra comma before a closing brace. The `window.opera` test is because some versions of Opera don't handle setting `script.text` properly, while some versions of Internet Explorer don't handle setting `script.innerHTML`.

Note that the **only** reliable way to configure MathJax is to use an in-line configuration block of the type discussed above. You should **not** call `MathJax.Hub.Config()` directly in your code, as it will not run at the correct time — it will either run too soon, in which case `MathJax` may not be defined and the function will throw an error, or it will run too late, after MathJax has already finished its configuration process, so your changes will not have the desired effect.

MathJax and GreaseMonkey

You can use techniques like the ones discussed above to good effect in GreaseMonkey scripts. There are GreaseMonkey work-alikes for all the major browsers:

- Firefox: GreaseMonkey
- Safari: GreaseKit (also requires SIMBL)
- Opera: Built-in (instructions)
- Internet Explorer: IEPro7
- Chrome: Built-in for recent releases

Note, however, that most browsers don't allow you to insert a script that loads a `file://` URL into a page that comes from the web (for security reasons). That means that you can't have your GreaseMonkey script load a local copy of MathJax, so you have to refer to a server-based copy. a CDN works nicely for this.

Here is a script that runs MathJax in any document that contains MathML (whether it includes MathJax or not). That allows browsers that don't have native MathML support to view any web pages with MathML, even if they say it only works in Firefox and IE+MathPlayer.

```
// ==UserScript==
// @name           MathJax MathML
// @namespace      http://www.mathjax.org/
// @description    Insert MathJax into pages containing MathML
// @include        *
// ==/UserScript==

if ((window.unsafeWindow == null ? window : unsafeWindow).MathJax == null) {
  if ((document.getElementsByTagName("math").length > 0) ||
      (document.getElementsByTagNameNS == null ? false :
      (document.getElementsByTagNameNS("http://www.w3.org/1998/Math/MathML","math").
↪length > 0))) {
    var script = document.createElement("script");
    script.type = "text/javascript";
    script.src = "https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML-full";
    document.getElementsByTagName("head")[0].appendChild(script);
  }
}
```

Source: mathjax_mathml.user.js

Here is a script that runs MathJax in Wikipedia pages after first converting the math images to their original TeX code.

```
// ==UserScript==
// @name           MathJax in Wikipedia
// @namespace      http://www.mathjax.org/
// @description    Insert MathJax into Wikipedia pages
// @include        http://en.wikipedia.org/wiki/*
// ==/UserScript==

if ((window.unsafeWindow == null ? window : unsafeWindow).MathJax == null) {
  //
  //  Replace the images with MathJax scripts of type math/tex
  //
  var images = document.getElementsByTagName('img'), count = 0;
  for (var i = images.length - 1; i >= 0; i--) {
    var img = images[i];
    if (img.className === "tex") {
      var script = document.createElement("script"); script.type = "math/tex";
      if (window.opera) {script.innerHTML = img.alt} else {script.text = img.alt}
      img.parentNode.replaceChild(script,img); count++;
    }
  }
  if (count) {
    //
    //  Load MathJax and have it process the page
    //
```

```
    var script = document.createElement("script");
    script.type = "text/javascript";
    script.src = "https://example.com/MathJax.js?config=TeX-AMS-MML_HTMLorMML-full";
    document.getElementsByTagName("head")[0].appendChild(script);
  }
}
```

Source: mathjax_wikipedia.user.js

Modifying Math on the Page

If you are writing a dynamic web page where content containing mathematics may appear after MathJax has already typeset the rest of the page, then you will need to tell MathJax to look for mathematics in the page again when that new content is produced. To do that, you need to use the `MathJax.Hub.Typeset()` method. This will cause the preprocessors (if any were loaded) to run over the page again, and then MathJax will look for unprocessed mathematics on the page and typeset it, leaving unchanged any math that has already been typeset.

You should not simply call this method directly, however. Because MathJax operates asynchonously (see *Synchronizing with MathJax* for details), you need to be sure that your call to `MathJax.Hub.Typeset()` is synchronized with the other actions that MathJax is taking. For example, it may already be typesetting portions of the page, or it may be waiting for an output jax to load, etc., and so you need to queue the typeset action to be performed after MathJax has finished whatever else it may be doing. That may be immediately, but it may not, and there is no way to tell.

To queue the typeset action, use the command

```
MathJax.Hub.Queue(["Typeset",MathJax.Hub]);
```

This will cause MathJax to typeset the page when it is next able to do so. It guarantees that the typesetting will synchronize properly with the loading of jax, extensions, fonts, stylesheets, and other asynchronous activity, and is the only truly safe way to ask MathJax to process additional material.

The `MathJax.Hub.Typeset()` command also accepts a parameter that is a DOM element whose content is to be typeset. That could be a paragraph, or a `<div>` element, or even a MathJax math `<script>` tag. It could also be the DOM *id* of such an object, in which case, MathJax will look up the DOM element for you. So

```
MathJax.Hub.Queue(["Typeset",MathJax.Hub,"MathExample"]);
```

would typeset the mathematics contained in the element whose *id* is `MathExample`. This is equivalent to

```
var math = document.getElementById("MathExample");
MathJax.Hub.Queue(["Typeset",MathJax.Hub,math]);
```

If no element or element *id* is provided, the whole document is typeset.

Note that the `MathJax.Hub.Queue()` method will return immediately, regardless of whether the typesetting has taken place or not, so you can not assume that the mathematics is visible after you make this call. That means that things like the size of the container for the mathematics may not yet reflect the size of the typeset mathematics. If you need to perform actions that depend on the mathematics being typeset, you should push *those* actions onto the `MathJax.Hub.queue` as well.

This can be quite subtle, so you have to think carefully about the structure of your code that works with the typeset mathematics. Also, the things you push onto the queue should be *Callback* objects that perform the actions you want when they are called, not the *results* of calling the functions that do what you want.

Manipulating Individual Math Elements

If you are not changing a complete DOM structure, but simply want to update the contents of a single mathematical equation, you do not need to use `innerHTML` and `MathJax.Hub.Typeset()` to preprocess and process an element's new content. Instead, you can ask MathJax to find the *element jax* for the math element on the page, and use its methods to modify and update the mathematics that it displays.

For example, suppose you have the following HTML in your document

```
<div id="MathDiv">
  The answer you provided is: \({}\).
</div>
```

and MathJax has already preprocessed and typeset the mathematics within the div. A student has typed something elsewhere on the page, and you want to typeset their answer in the location of the mathematics that is already there. You could replace the entire contents of the *MathDiv* element and call `MathJax.Hub.Typeset()` as described above, but there is a more efficient approach, which is to ask MathJax for the element jax for the mathematics, and call its method for replacing the formula shown by that element. For example:

```
var math = MathJax.Hub.getAllJax("MathDiv")[0];
MathJax.Hub.Queue(["Text",math,"x+1"]);
```

This looks up the list of math elements in the *MathDiv* element (there is only one) and takes the first one (element 0) and stores it in `math`. This is an *element jax* object (see the *Element Jax* specification for details), which has a *`Text()`* method that can be used to set the input text of the math element, and retypeset it.

Again, since the typesetting should be synchronized with other actions of MathJax, the call should be pushed onto the MathJax processing queue using `MathJax.Hub.Queue()`, as shown above, rather than called directly. The example above performs the equivalent of `math.Text("x+1")` as soon as MathJax is able to do so. Any additional actions that rely on the expression `x+1` actually showing on screen should also be pushed onto the queue so that they will not occur before the math is typeset.

The actions you can perform on an element jax include:

`Text(newmath)`
: to set the math text of the element to *newmath* and typeset.

`Rerender()`
: to remove the output and reproduce it again (for example, if CSS has changed that would alter the spacing of the mathematics). Note that the internal representation isn't regenerated; only the output is.

`Reprocess()`
: to remove the output and then retranslate the input into the internal MathML and rerender the output.

`Remove()`
: to remove the output for this math element (but not the original `<script>` tag).

`needsUpdate()`
: to find out if the mathematics has changed so that its output needs to be updated.

`SourceElement()`
: to obtain a reference to the original `<script>` object that is associated with this element jax.

Note that once you have located an element jax, you can keep using it and don't have to look it up again. So for the example above, if the student is going to be able to type several different answers that you will want to typeset, you can look up the element jax once at the beginning after MathJax has processed the page the first time, and then use that result each time you adjust the mathematics to be displayed.

To get the element jax the first time, you need to be sure that you ask MathJax for it **after** MathJax has processed the page the first time. This is another situation where you want to use the MathJax queue. If your startup code performs the commands

```
var studentDisplay = null;
MathJax.Hub.Queue(function () {
  studentDisplay = MathJax.Hub.getAllJax("MathDiv")[0];
});
```

then you can use

```
MathJax.Hub.Queue(["Text",studentDisplay,studentAnswer])
```

to change the student's answer to be the typeset version of whatever is in the `studentAnswer` variable.

Here is a complete example that illustrates this approach. Note, however, that Internet Explorer does not fire the `onchange` event when you press RETURN, so this example does not work as expected in IE. A more full-featured version that addresses this problem is available in test/sample-dynamic.html.

```
<html>
<head>
<title>MathJax Dynamic Math Test Page</title>

<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    tex2jax: {
      inlineMath: [["$","$"],["\\(","\\)"]]
    }
  });
</script>
<script type="text/javascript"
  src="https://example.com/MathJax.js?config=TeX-AMS_HTML-full">
</script>

</head>
<body>

<script>
  //
  //  Use a closure to hide the local variables from the
  //  global namespace
  //
  (function () {
    var QUEUE = MathJax.Hub.queue;  // shorthand for the queue
    var math = null;                // the element jax for the math output.

    //
    //  Get the element jax when MathJax has produced it.
    //
    QUEUE.Push(function () {
      math = MathJax.Hub.getAllJax("MathOutput")[0];
    });

    //
    //  The onchange event handler that typesets the
    //  math entered by the user
    //
    window.UpdateMath = function (TeX) {
      QUEUE.Push(["Text",math,"\\displaystyle{"+TeX+"}"]);
```

```
    }
  })();
</script>

Type some TeX code:
<input id="MathInput" size="50" onchange="UpdateMath(this.value)" />
<p>

<div id="MathOutput">
You typed: ${}$
</div>

</body>
</html>
```

There are a number of additional example pages at test/examples.html that illustrate how to call MathJax dynamically or perform other actions with MathJax.

The MathJax API

The following links document the various components that make up MathJax. These are implemented as JavaScript objects contained within the single global variable, `MathJax`. Although JavaScript includes an object system with some inheritance capabilities, they do not constitute a full object-oriented programming model, so MathJax implements its own object library. This means there is an ambiguity when we speak of an "object", as it could be either a native JavaScript object, or a MathJax object. When the distinction is important, we will use *Object* (capitalized) or *MathJax.Object* for the latter; the javascript object will always be listed in lower case.

You may also want to view the *advanced topics* on the main MathJax documentation page.

The MathJax variable

MathJax has a single global variable, `MathJax`, in which all its data, and the data for loaded components, are stored. The MathJax variable is a nested structure, with its top-level properties being objects themselves.

Main MathJax Components

`MathJax.Hub`
: Contains the MathJax hub code and variables, including the startup code, the onload handler, the browser data, and so forth.

`MathJax.Ajax`
: Contains the code for loading external modules and creating stylesheets. Most of the code that causes MathJax to operate asynchronously is handled here.

`MathJax.Message`
: Contains the code to handle the intermittent message window that periodically appears in the lower left-hand corner of the window.

`MathJax.HTML`
: Contains support code for creating HTML elements dynamically from descriptions stored in JavaScript objects.

`MathJax.CallBack`
: Contains the code for managing MathJax callbacks, queues and signals.

MathJax.Extension
Initially empty, this is where extensions can load their code. For example, the *tex2jax* preprocessor creates `MathJax.Extension.tex2jax` for its code and variables.

MathJax.Menu
Initially null, this is where the MathJax contextual menu is stored, when `extensions/MathMenu.js` is loaded.

MathJax.Object
Contains the code for the MathJax object-oriented programming model.

MathJax.InputJax
The base class for all input *jax* objects. Subclasses for specific input jax are created as sub-objects of `MathJax.InputJax`. For example, the TeX input jax loads itself as `MathJax.InputJax.TeX`.

MathJax.OutputJax
The base class for all output *jax* objects. Subclasses for specific output jax are created as sub-objects of `MathJax.OutputJax`. For example, the HTML-CSS output jax loads itself as `MathJax.OutputJax["HTML-CSS"]`.

MathJax.ElementJax
The base class for all element *jax* objects. Subclasses for specific element jax are created as sub-objects of `MathJax.ElementJax`. For example, the mml element jax loads itself as `MathJax.ElementJax.mml`.

Properties

MathJax.version
The version number of the MathJax library as a whole.

MathJax.fileversion
The version number of the `MathJax.js` file specifically.

MathJax.isReady
This is set to `true` when MathJax is set up and ready to perform typesetting actions (and is `null` otherwise).

The MathJax.Hub Object

The MathJax Hub, *MathJax.Hub*, is the main control structure for MathJax. It is where input and output *jax* are tied together, and it is what handles processing of the MathJax `<script>` tags. Processing of the mathematics on the page may require external files to be loaded (when the mathematics includes less common functionality, for example, that is defined in an extension file), and since file loading is asynchronous, a number of the methods below may return before their actions are completed. For this reason, they include callback functions that are called when the action completes. These can be used to synchronize actions that require the mathematics to be completed before those actions occur. See the *Using Callbacks* documentation for more details.

Properties

config: { ... }
This holds the configuration parameters for MathJax. Set these values using `MathJax.Hub.Config()` described below. The options and their default values are given in the *Core Options* reference page.

processUpdateTime: 250
The minimum time (in milliseconds) between updates of the "Processing Math" message. After this amount of time has passed, and after the next equation has finished being processed, MathJax will stop processing momentarily so that the update message can be displayed, and so that the browser can handle user interaction.

processUpdateDelay: 10
The amount of time (in milliseconds) that MathJax pauses after issuing its processing message before starting the processing again (to give browsers time to handle user interaction).

signal
The hub processing signal (tied to the `MathJax.Hub.Register.MessageHook()` method).

queue
MathJax's main processing queue. Use `MathJax.Hub.Queue()` to push callbacks onto this queue.

Browser
The name of the browser as determined by MathJax. It will be one of `Firefox`, `Safari`, `Chrome`, `Opera`, `MSIE`, `Konqueror`, or `unkown`. This is actually an object with additional properties and methods concerning the browser:

version
The browser version number, e.g., `"4.0"`

isMac and isPC
These are boolean values that indicate whether the browser is running on a Macintosh computer or a Windows computer. They will both be `false` for a Linux computer.

isMobile
This is `true` when MathJax is running a mobile version of a WebKit or Gecko-based browser.

isFirefox, isSafari, isChrome, isOpera, isMSIE, isKonqueror
These are `true` when the browser is the indicated one, and `false` otherwise.

versionAtLeast(version)
This tests whether the browser version is at least that given in the *version* string. Note that you can not simply do a numeric comparison, as version 4.10 should be considered later than 4.9, for example. Similarly, 4.10 is different from 4.1, for instance.

Select(choices)
This lets you perform browser-specific functions. Here, *choices* is an object whose properties are the names of the browsers and whose values are the functions to be performed. Each function is passed one parameter, which is the `MathJax.Hub.Browser` object. You do not need to include every browser as one of your choices — only those for which you need to do special processing. For example:

```
MathJax.Hub.Browser.Select({
  MSIE: function (browser) {
    if (browser.versionAtLeast("8.0")) {... do version 8 stuff ... }
    ... do general MSIE stuff ...
  },

  Firefox: function (browser) {
    if (browser.isMac) {... do Mac stuff ... }
    ... do general Firefox stuff
  }
});
```

inputJax
An object storing the MIME types associated with the various registered input jax (these are the types of the `<script>` tags that store the math to be processed by each input jax).

outputJax
An object storing the output jax associate with the various element jax MIME types for the registered output jax.

Methods

Config(*options*)
Sets the configuration options (stored in `MathJax.Hub.config`) to the values stored in the *options* object. See *Configuring MathJax* for details on how this is used and the options that you can set.

Parameters

- **options** — object containing options to be set

Returns `null`

Configured()
When `delayStartupUntil` is specified in the configuration file or in the script that loads `MathJax.js`, MathJax's startup sequence is delayed until this routine is called. See *Configuring MathJax* for details on how this is used.

Returns `null`

Register.PreProcessor(callback)
Used by preprocessors to register themselves with MathJax so that they will be called during the `MathJax.Hub.PreProcess()` action.

Parameters

- **callback** — the callback specification for the preprocessor

Returns `null`

Register.MessageHook(type,callback)
Registers a listener for a particular message being sent to the hub processing signal (where *PreProcessing*, *Processing*, and *New Math* messages are sent). When the message equals the *type*, the *callback* will be called with the message as its parameter.

Parameters

- **type** — a string indicating the message to look for
- **callback** — a callback specification

Returns `null`

Register.StartupHook(type,callback)
Registers a listener for a particular message being sent to the startup signal (where initialization and component startup messages are sent). When the message equals the *type*, the *callback* will be called with the message as its parameter. See the *Using Signals* documentation for more details.

Parameters

- **type** — a string indicating the message to look for
- **callback** — a callback specification

Returns `null`

Register.LoadHook(file,callback)
Registers a callback to be called when a particular file is completely loaded and processed. (The callback is called when the file makes its `MathJax.Ajax.loadComplete()` call.) The *file* should be the complete file name, e.g., `"[MathJax]/config/default.js"`.

Parameters

- **file** — the name of the file to wait for
- **callback** — a callback specification

Returns the callback object

Queue (*callback*, ...)
Pushes the given callbacks onto the main MathJax command queue. This synchronizes the commands with MathJax so that they will be performed in the proper order even when some run asynchronously. See *Using Queues* for more details about how to use queues, and the MathJax queue in particular. You may supply as many *callback* specifications in one call to the `Queue()` method as you wish.

Parameters

- **callback** — a callback specification

Returns the callback object for the last callback added to the queue

Typeset ([*element*[, *callback*]])
Calls the preprocessors on the given element (or elements if it is an array of elements), and then typesets any math elements within the element. If no *element* is provided, the whole document is processed. The *element* is either the DOM *id* of the element, a reference to the DOM element itself, or an array of id's or refereneces. The *callback* is called when the process is complete. See the *Modifying Math* section for details of how to use this method properly.

Parameters

- **element** — the element(s) whose math is to be typeset
- **callback** — the callback specification

Returns the callback object

PreProcess ([*element*[, *callback*]])
Calls the loaded preprocessors on the entire document, or on the given DOM element (or elements, if it is an array of elements). The *element* is either the DOM *id* of the element, a reference to the DOM element itself, or an array of id's or references. The *callback* is called when the processing is complete.

Parameters

- **element** — the element to be preprocessed
- **callback** — the callback specification

Returns the callback object

Process ([*element*[, *callback*]])
Scans either the entire document or a given DOM *element* (or array of elements) for MathJax `<script>` tags and processes the math those tags contain. The *element* is either the DOM *id* of the element to scan, a reference to the DOM element itself, or an array of id's or references. The *callback* is called when the processing is complete.

Parameters

- **element** — the element(s) to be processed
- **callback** — the callback specification

Returns the callback object

Update ([*element*[, *callback*]])
Scans either the entire document or a given DOM element (or elements if it is an array of elements) for mathematics that has changed since the last time it was processed, or is new, and typesets the mathematics they contain. The *element* is either the DOM *id* of the element to scan, a reference to the DOM element itself, or an array of id's or references. The *callback* is called when the processing is complete.

Parameters

- **element** — the element(s) to be updated

- **callback** — the callback specification

Returns the callback object

Reprocess ([*element* [, *callback*]])

Removes any typeset mathematics from the document or DOM element (or elements if it is an array of elements), and then processes the mathematics again, re-typesetting everything. This may be necessary, for example, if the CSS styles have changed and those changes would affect the mathematics. Reprocess calls both the input and output jax to completely rebuild the data for mathematics. The *element* is either the DOM *id* of the element to scan, a reference to the DOM element itself, or an array of id's or references. The *callback* is called when the processing is complete.

Parameters

- **element** — the element(s) to be reprocessed
- **callback** — the callback specification

Returns the callback object

Rerender ([*element* [, *callback*]])

Removes any typeset mathematics from the document or DOM element (or elements if it is an array of elements), and then renders the mathematics again, re-typesetting everything from the current internal version (without calling the input jax again). The *element* is either the DOM *id* of the element to scan, a reference to the DOM element itself, or an array of id's or references. The *callback* is called when the processing is complete.

Parameters

- **element** — the element(s) to be reprocessed
- **callback** — the callback specification

Returns the callback object

getAllJax ([*element*])

Returns a list of all the element jax in the document or a specific DOM element. The *element* is either the DOM *id* of the element, or a reference to the DOM element itself.

Parameters

- **element** — the element to be searched

Returns array of *element jax* objects

getJaxByType (*type* [, *element*])

Returns a list of all the element jax of a given MIME-type in the document or a specific DOM element. The *element* is either the DOM *id* of the element to search, or a reference to the DOM element itself.

Parameters

- **type** — MIME-type of *element jax* to find
- **element** — the element to be searched

Returns array of *element jax* objects

getJaxByInputType (*type* [, *element*])

Returns a list of all the element jax associated with input `<script>` tags with the given MIME-type within the given DOM element or the whole document. The *element* is either the DOM *id* of the element to search, or a reference to the DOM element itself.

Parameters

- **type** — MIME-type of input (e.g., `"math/tex"`)
- **element** — the element to be searched

Returns array of *element jax* objects

getJaxFor(*element*)

Returns the element jax associated with a given DOM element. If the element does not have an associated element jax, `null` is returned. The *element* is either the DOM *id* of the element, or a reference to the DOM element itself.

Parameters

- **element** — the element whose element jax is required

Returns *element jax* object or `null`

isJax(*element*)

Returns `0` if the element is not a `<script>` that can be processed by MathJax or the result of an output jax, returns `-1` if the element is an unprocessed `<script>` tag that could be handled by MathJax, and returns `1` if the element is a processed `<script>` tag or an element that is the result of an output jax.

Parameters

- **element** — the element to inspect

Returns integer (-1, 0, 1)

setRenderer(*renderer*[, *type*])

Sets the output jax for the given element jax `type` (or `jax/mml` if none is specified) to be the one given by `renderer`, which must be the name of a renderer, such as `NativeMML` or `HTML-CSS`. Note that this does not cause the math on the page to be rerendered; it just sets the renderer for output in the future (call :meth:`Rerender()` above to replace the current renderings by new ones).

Parameters

- **renderer** — the name of the output jax to use for rendering
- **type** — the element jax MIME type whose renderer to set

Returns `null`

Insert(*dst*, *src*)

Inserts data from the *src* object into the *dst* object. The *key:value* pairs in *src* are (recursively) copied into *dst*, so that if *value* is itself an object, its content is copied into the corresponding object in *dst*. That is, objects within *src* are merged into the corresponding objects in *dst* (they don't replace them).

Parameters

- **dst** — the destination object
- **src** — the source object

Returns the modified destination object

formatError(*script*, *error*)

This is called when an internal error occurs during the processing of a math element (i.e., an error in the MathJax code itself). The *script* is a reference to the `<script>` tag where the error occurred, and *error* is the `Error` object for the error. The default action is to insert an HTML snippet at the location of the script, but this routine can be overriden during MathJax configuration in order to perform some other action. `MathJax.Hub.lastError` holds the `error` value of the last error on the page.

Parameters

- **script** — the `<script>` tag causing the error
- **error** — the `Error` object for the error

Returns `null`

The MathJax.Ajax Object

The *MathJax.Ajax* structure holds the data and functions for handling loading of external modules. Modules are loaded only once, even if called for in several places. The loading of files is asynchronous, and so the code that requests an external module will continue to run even when that module has not completed loading, so it is important to be aware of the timing issues this may cause. Similarly, creating or loading stylesheets is an asynchronous action. In particular, all actions that rely on the file or stylesheet having been loaded must be delayed until after the file has been downloaded completely. This is the reason for the large number of routines that take callback functions.

Any operation that could cause the loading of a file or stylesheet must be synchronized with the rest of the code via such callbacks. Since processing any mathematics might cause files to be loaded (e.g., little-used markup might be implemented in an extension that is loaded only when that markup is used), any code that dynamically typesets mathematics will need to be structured to use callbacks to guarantee that the mathematics has been completely processed before the code tries to use it. See the *Synchronizing with MathJax* documentation for details on how to do this properly.

Properties

`timeout`
Number of milliseconds to wait for a file to load before it is considered to have failed to load.

Default: 15 seconds

`STATUS.OK`
The value used to indicate that a file load has occurred successfully.

`STATUS.ERROR`
The value used to indicate that a file load has caused an error or a timeout to occur.

`loaded`
An object containing the names of the files that have been loaded (or requested) so far. `MathJax.Ajax.loaded["file"]` will be non-`null` when the file has been loaded, with the value being the `MathJax.Ajax.STATUS` value of the load attempt.

`loading`
An object containing the files that are currently loading, the callbacks that are to be run when they load or timeout, and additional internal data.

`loadHooks`
An object containing the load hooks for the various files, set up by the *`LoadHook()`* method, or by the `MathJax.Hub.Register.LoadHook()` method.

Methods

`Require` (*file*[, *callback*])
Loads the given file if it hasn't been already. The file must be a JavaScript file or a CSS stylesheet; i.e., it must end in `.js` or `.css`. Alternatively, it can be an object with a single *key:value* pair where the *key* is one of `js` or `css` and the *value* is the file of that type to be loaded (this makes it possible to have the file be created by a CGI script, for example, or to use a `data::` URL). The file must be relative to the MathJax home directory and can not contain `../` file path components.

When the file is completely loaded and run, the *callback*, if provided, will be executed passing it the status of the file load. If there was an error while loading the file, or if the file fails to load within the time limit given by `MathJax.Ajax.timout`, the status will be `MathJax.Ajax.STATUS.ERROR` otherwise it will be `MathJax.Ajax.STATUS.OK`. If the file is already loaded, the callback will be called immediately and the file will not be loaded again.

Parameters

- **file** — name of the file to be loaded
- **callback** — the callback specification

Returns the callback object

Load(*file*[, *callback*])
Used internally to load a given file without checking if it already has been loaded, or where it is to be found.

Parameters

- **file** — name of the file to be loaded
- **callback** — the callback specification

Returns the callback object

loadComplete(*file*)
Called from within the loaded files to inform MathJax that the file has been completely loaded and initialized. The *file* parameter is the name of the file that has been loaded. This routine will cause any callback functions registered for the file or included in the `MathJax.Ajax.Require()` calls to be executed, passing them the status of the load (`MathJax.Ajax.STATUS.OK` or `MathJax.Ajax.STATUS.ERROR`) as their last parameter.

Parameters

- **file** — name of the file that has been loaded

Returns `null`

loadTimeout(*file*)
Called when the timeout period is over and the file hasn't loaded. This indicates an error condition, and the `MathJax.Ajax.loadError()` method will be executed, then the file's callback will be run with `MathJax.Ajax.STATUS.ERROR` as its parameter.

Parameters

- **file** — name of the file that timed out

Returns `null`

loadError(*file*)
The default error handler called when a file fails to load. It puts a warning message into the MathJax message box on screen.

Parameters

- **file** — the name of the file that failed to load

Returns `null`

LoadHook(*file*, *callback*)
Registers a callback to be executed when the given file is loaded. The file load operation needs to be started when this method is called, so it can be used to register a hook for a file that may be loaded in the future.

Parameters

- **file** — the name of the file to wait for
- **callback** — the callback specification

Returns the callback object

Preloading (*file1*[, *file2...*])
Used with combined configuration files to indicate what files are in the configuration file. Marks the files as loading (since there will never be an explicit *Load()* or *Require()* call for them), so that load-hooks and other load-related events can be properly processed when the *loadComplete()* occurs.

Parameters

- **file1, file2, ...** — the names of the files in the combined file

Returns `null`

Styles (*styles*[, *callback*])
Creates a stylesheet from the given style data. *styles* can either be a string containing a stylesheet definition, or an object containing a *CSS Style Object*. For example:

```
MathJax.Ajax.Styles("body {font-family: serif; font-style: italic}");
```

and

```
MathJax.Ajax.Styles({
  body: {
    "font-family": "serif",
    "font-style":  "italic"
  }
});
```

both set the body font family and style.

The callback routine is called when the stylesheet has been created and is available for use.

Parameters

- **styles** — CSS style object for the styles to set
- **callback** — the callback specification

Returns the callback object

Note: Internet Explorer has a limit of 32 dynamically created stylesheets, so it is best to combine your styles into one large group rather than making several smaller calls.

---

fileURL (*file*)
Returns a complete URL to a file (replacing `[MathJax]` with the actual root URL location).

Parameters

- **file** — the file name possibly including `[MathJax]`

Returns the full URL for the file

The MathJax.Message Object

The `MathJax.Message` object contains the methods used to manage the small message area that appears at the lower-left corner of the window. MathJax uses this area to inform the user of time-consuming actions, like loading files and fonts, or how far along in the typesetting process it is.

The page author can customize the look of the message window by setting styles for the `#MathJax_Message` selector (which can be set via

```
MathJax.Hub.Config({
  styles: {
    "#MathJax_Message": {
      ...
    }
  }
});
```

Because of a bug in Internet Explorer, in order to change the side of the screen where the message occurs, you must also set the side for `#MathJax_MSIE_Frame`, as in

```
MathJax.Hub.Config({
  styles: {
    "#MathJax_Message": {left: "", right: 0},
    "#MathJax_MSIE_Frame": {left: "", right: 0}
  }
});
```

It is possible that a message is already being displayed when another message needs to be posted. For this reason, when a message is displayed on screen, it gets an id number that is used when you want to remove or change that message. That way, when a message is removed, the previous message (if any) can be redisplayed if it hasn't been removed. This allows for intermittent messages (like file loading messages) to obscure longer-term messages (like "Processing Math" messages) temporarily.

Methods

Set (*message*[, *n*[, *delay*]])

This sets the message being displayed to the given *message* string. If *n* is not `null`, it represents a message id number and the text is set for that message id, otherwise a new id number is created for this message. If *delay* is provided, it is the time (in milliseconds) to display the message before it is cleared. If *delay* is not provided, the message will not be removed automatically; you must call the `MathJax.Messsage.Clear()` method by hand to remove it. If *message* is an array, then it represents a localizable string, as described in the *Localization strings* documentation.

Parameters

- **message** — the text to display in the message area
- **n** — the message id number
- **delay** — amout of time to display the message

Returns the message id number for this message.

Clear (*n*[, *delay*])

This causes the message with id *n* to be removed after the given *delay*, in milliseconds. The default delay is 600 milliseconds.

Parameters

- **n** — the message id number
- **delay** — the delay before removing the message

Returns `null`

Remove ()

This removes the message frame from the window (it will reappear when future messages are set, however).

Returns `null`

File (*file*)
This sets the message area to a "Loading *file*" message, where *file* is the name of the file (with `[MathJax]` representing the root directory).

Parameters

- **file** — the name of the file being loaded

Returns the message id number for the message created

filterText (*text*, *n*)
This method is called on each message before it is displayed. It can be used to modify (e.g., shorten) the various messages before they are displayed. The default action is to check if the `messageStyle` configuration parameter is `simple`, and if so, convert loading and processing messages to a simpler form. This method can be overridden to perform other sanitization of the message strings.

Parameters

- **text** — the text of the message to be posted
- **n** — the id number of the message to be posted

Returns the modified message text

Log ()
Returns a string of all the messages issued so far, separated by newlines. This is used in debugging MathJax operations.

Returns string of all messages so far

The MathJax.HTML Object

The `MathJax.HTML` object provides routines for creating HTML elements and adding them to the page, and in particular, it contains the code that processes MathJax's *HTML snippets* and turns them into actual DOM objects. It also implements the methods used to manage the cookies used by MathJax.

Properties

Cookie.prefix: "mjx"
The prefix used for names of cookies stored by MathJax.

Cookie.expires: 365
The expiration time (in days) for cookies created by MathJax.

Methods

Element (*type*[, *attributes*[, *contents*]])
Creates a DOM element of the given type. If *attributes* is non-`null`, it is an object that contains *key:value* pairs of attributes to set for the newly created element. If *contents* is non-`null`, it is an *HTML snippet* that describes the contents to create for the element. For example

```
var div = MathJax.HTML.Element(
  "div",
  {id: "MathDiv", style:{border:"1px solid", padding:"5px"}},
  ["Here is math: \\(x+1\\)",["br"],"and a display $$x+1\\over x-1$$"]
);
```

Parameters

- **type** — node type to be created
- **attributes** — object specifying attributes to set
- **contents** — HTML snippet representing contents of node

Returns the DOM element created

addElement (*parent*, *type*[, *attributes*[, *content*]])
Creates a DOM element and appends it to the *parent* node provided. It is equivalent to

```
parent.appendChild(MathJax.HTML.Element(type,attributes,content))
```

Parameters

- **parent** — the node where the element will be added
- **attributes** — object specifying attributes to set
- **contents** — HTML snippet representing contents of node

Returns the DOM element created

TextNode (*text*)
Creates a DOM text node with the given text as its content.

Parameters

- **text** — the text for the node

Returns the new text node

addText (*parent*, *text*)
Creates a DOM text node with the given text and appends it to the *parent* node.

Parameters

- **parent** — the node where the text will be added
- **text** — the text for the new node

Returns the new text node

setScript (*script*, *text*)
Sets the contents of the `script` element to be the given `text`, properly taking into account the browser limitations and bugs.

Parameters

- **script** — the script whose content is to be set
- **text** — the text that is to be the script's new content

Returns `null`

getScript (*script*)
Gets the contents of the `script` element, properly taking into account the browser limitations and bugs.

Parameters

- **script** — the script whose content is to be retrieved

Returns the text of the `script`

Cookie.Set(name,data)
Creates a MathJax cookie using the MathJax.HTML.Cookie.prefix and the *name* as the cookie name, and the *key:value* pairs in the *data* object as the data for the cookie. For example,

```
MathJax.HTML.Cookie.Set("test",{x:42, y:"It Works!"});
```

will create a cookie named "mjx.test" that stores the values of x and y provided in the *data* object. This data can be retrieved using the MathJax.HTML.Cookie.Get() method discussed below.

Parameters

- **name** — the name that identifies the cookie
- **data** — object containing the data to store in the cookie

Returns null

Cookie.Get(name[,obj])
Looks up the data for the cookie named *name* and merges the data into the given *obj* object, or returns a new object containing the data. For instance, given the cookie stored by the example above,

```
var data = MathJax.HTML.Cookie.Get("test");
```

would set data to {x:42, y:"It Works!"}, while

```
var data = {x:10, z:"Safe"};
MathJax.HTML.Cookie.Get("test",data);
```

would leave data as {x:42, y:"It Works!", z:"Safe"}.

The MathJax.Callback Class

The MathJax.Callback object is one of the key mechanisms used by MathJax to synchronize its actions with those that occur asynchronously, like loading files and stylesheets. A *Callback* object is used to tie the execution of a function to the completion of an asynchronous action. See *Synchronizing with MathJax* for more details, and *Using Callbacks* in particular for examples of how to specify and use MathJax *Callback* objects.

Specifying a callback

When a method includes a callback as one of its arguments, that callback can be specified in a number of different ways, depending on the functionality that is required of the callback. The easiest case is to simply provide a function to be called, but it is also possible to include data to pass to the function when it is executed, and even the object that will be used as the javascript *this* object when the function is called.

Most functions that take callbacks as arguments accept a *callback specification* rather than an actual callback object, though you can use the MathJax.Callback() function to convert a callback specification into a Callback object if needed.

A callback specification is any one of the following:

fn
A function that is to be called when the callback is executed. No additional data is passed to it (other than what it is called with at the time the callback is executed), and *this* will be the window object.

[fn]
An array containing a function to be called when the callback is executed (as above).

[fn, data...]
An array containing a function together with data to be passed to that function when the callback is executed; *this* is still the window object. For example,

```
[function (x,y) {return x+y}, 2, 3]
```

would specify a callback that would pass `2` and `3` to the given function, and it would return their sum, `5`, when the callback is executed.

[object, fn]
An array containing an object to use as *this* and a function to call for the callback. For example,

```
[{x:'foo', y:'bar'}, function () {this.x}]
```

would produce a callback that returns the string `"foo"` when it is called.

[object, fn, data...]
Similar to the previous case, but with data that is passed to the function as well.

["method", object]
Here, *object* is an object that has a method called *method*, and the callback will execute that method (with the object as *this*) when it is called. For example,

```
["toString",[1,2,3,4]]
```

would call the *toString* method on the array `[1,2,3,4]` when the callback is called, returning `1,2,3,4`.

["method", object, data...]
Similar to the previous case, but with data that is passed to the method. E.g.,

```
["slice",[1,2,3,4],1,3]
```

would perform the equivalent of `[1,2,3,4].slice(1,3)`, which returns the array `[2,3]` as a result.

{hook: fn, data: [...], object: this}
Here the data for the callback are given in an associative array of *key:value* pairs. The value of *hook* is the function to call, the value of *data* is an array of the arguments to pass to the function, and the value of *object* is the object to use as *this* in the function call. The specification need not include all three *key:value* pairs; any that are missing get default values (a function that does nothing, an empty array, and the window object, respectively).

"string"
This specifies a callback where the string is executed via an `eval()` statement. The code is run in the global context, so any variables or functions created by the string become part of the global namespace. The return value is the value of the last statement executed in the string.

Executing a Callback Object

The *Callback* object is itself a function, and calling that function executes the callback. You can pass the callback additional parameters, just as you can any function, and these will be added to the callback function's argument list following any data that was supplied at the time the callback was created. For example

```
var f = function (x,y) {return x + " and " +y}
var cb = MathJax.Callback([f, "foo"]);
var result = cb("bar");  // sets result to "foo and bar"
```

Usually, the callback is not executed by the code that creates it (as it is in the example above), but by some other code that runs at a later time at the completion of some other activity (say the loading of a file), or in response to a user action. For example:

```
function f(x) {alert("x contains "+x)};
function DelayedX(time) {
    var x = "hi";
    setTimeout(MathJax.Callback([f, x], time));
}
```

The `DelayedX` function arranges for the function `f` to be called at a later time, passing it the value of a local variable, `x`. Normally, this would require the use of a closure, but that is not needed when a *MathJax.Callback* object is used.

Callback Object Properties

hook
: The function to be called when the callback is executed.

data
: An array containing the arguments to pass to the callback function when it is executed.

object
: The object to use as *this* during the call to the callback function.

called
: Set to `true` after the callback has been called, and undefined otherwise. A callback will not be executed a second time unless the callback's *`reset()`* method is called first, or its `autoReset` property is set to `true`.

autoReset
: Set this to `true` if you want to be able to call the callback more than once. (This is the case for signal listeners, for example).

isCallback
: Always set to `true` (used to detect if an object is a callback or not).

Callback Object Methods

reset ()
: Clears the callback's *called* property.

MathJax.Callback Methods

Delay (*time*[, *callback*])
: Waits for the specified time (given in milliseconds) and then performs the callback. It returns the Callback object (or a blank one if none was supplied). The returned callback structure has a *timeout* property set to the result of the `setTimeout()` call that was used to perform the wait so that you can cancel the wait, if needed. Thus `MathJax.Callback.Delay()` can be used to start a timeout delay that executes the callback if an action doesn't occur within the given time (and if the action does occur, the timeout can be canceled). Since `MathJax.Callback.Delay()` returns a callback structure, it can be used in a callback queue to insert a delay between queued commands.

  Parameters

  - **time** — the amount of time to wait
  - **callback** — the callback specification

Returns the callback object

Queue(*[callback, ...]*)
Creates a *MathJax.CallBack.Queue* object and pushes the given callbacks into the queue. See *Using Queues* for more details about MathJax queues.

Parameters

- **callback** — one or more callback specifications

Returns the *Queue* object

Signal(*name*)
Looks for a named signal, creates it if it doesn't already exist, and returns the signal object. See *Using Signals* for more details.

Parameters

- **name** — name of the signal to get or create

Returns the *Signal* object

ExecuteHooks(*hooks*[, *data*[, *reset*]])
Calls each callback in the *hooks* array (or the single hook if it is not an array), passing it the arguments stored in the data array. If *reset* is `true`, then the callback's *reset()* method will be called before each hook is executed. If any of the hooks returns a *Callback* object, then it collects those callbacks and returns a new callback that will execute when all the ones returned by the hooks have been completed. Otherwise, `MathJax.Callback.ExecuteHooks()` returns `null`.

Parameters

- **hooks** — array of hooks to be called, or a hook
- **data** — array of arguments to pass to each hook in turn
- **reset** — `true` if the *reset()* method should be called

Returns callback that waits for all the hooks to complete, or `null`

Hooks(*reset*)
Creates a prioritized list of hooks that are called in order based on their priority (low priority numbers are handled first). This is meant to replace `MathJax.Callback.ExecuteHooks()` and is used internally for signal callbacks, pre- and post-filters, and other lists of callbacks.

Parameters

- **reset** — `true` if callbacks can be called more than once

Returns the *Hooks* object

The list has the following methods:

Add(*hook*[, *priority*])
Add a callback to the prioritized list. If `priority` is not provided, the default is 10. The `hook` is a *Callback* specification as described above.

Parameters

- **hook** — callback specification to add to the list
- **priority** — priority of the hook in the list (default: 10)

Returns the callback object being added

Remove(*hook*)
Remove a given hook (as returned from *Add()* above) from the prioritized list.

Parameters

- **hook** — the callback to be removed

Returns `null`

`Execute``()`
Execute the list of callbacks, resetting them if requested. If any of the hooks return callbacks, then `Execute()` returns a callback that will be executed when they all have completed.

Returns a callback object or `null`

The MathJax.Callback.Queue Class

The `MathJax.Callback.Queue` object is one of the key mechanisms used by MathJax to synchronize its actions with those that occur asynchronously, like loading files and stylesheets. A *Queue* object is used to coordinate a sequence of actions so that they are performed one after another, even when one action has to wait for an asynchronous process to complete. This guarantees that operations are performed in the right order even when the code must wait for some other action to occur. See *Synchronizing with MathJax* for more details, and *Using Queues* in particular for examples of how to specify and use MathJax *Queue* objects.

Properties

`pending`
This is non-zero when the queue is waiting for a command to complete, i.e. a command being processed returns a *Callback* object, indicating that the queue should wait for that action to complete before processing additional commands.

`running`
This is non-zero when the queue is executing one of the commands in the queue.

`queue`
An array containing the queued commands that are yet to be performed.

Methods

`Push`(*callback*, ...)
Adds commands to the queue and runs them (if the queue is not pending or running another command). If one of the callbacks is an actual *Callback* object rather than a callback specification, then the command queued is an internal command to wait for the given callback to complete. That is, that callback is not itself queued to be executed, but a wait for that callback is queued. The `Push()` method returns the last callback that was added to the queue (so that it can be used for further synchronization, say as an entry in some other queue).

Parameters

- **callback** — the callback specifications to be added to the queue

Returns the last callback object added to the queue

`Process``()`
Process the commands in the queue, provided the queue is not waiting for another command to complete. This method is used internally; you should not need to call it yourself.

`Suspend``()`
Increments the *running* property, indicating that any commands that are added to the queue should not be executed immediately, but should be queued for later execution (when its `Resume()` is called). This method is used internally; you should not need to call it yourself.

Resume()
Decrements the *running* property, if it is positive. When it is zero, commands can be processed, but that is not done automatically — you would need to call `Process()` to make that happen. This method is used internally; you should not need to call it yourself.

wait(*callback*)
Used internally when an entry in the queue is a *Callback* object rather than a callback specification. A callback to this function (passing it the original callback) is queued instead, and it simply returns the callback it was passed. Since the queue will wait for a callback if it is the return value of one of the commands it executes, this effectively makes the queue wait for the original callback at that point in the command queue.

Parameters

- **callback** — the function to complete before returning to the queue

Returns the passed callback function

call()
An internal function used to restart processing of the queue after it has been waiting for a command to complete.

The MathJax.Callback.Signal Class

The `MathJax.Callback.Signal` object is one of the key mechanisms used by MathJax to synchronize its actions with those that occur asynchronously, like loading files and stylesheets. A *Signal* object is used to publicize the fact that MathJax has performed certain actions, giving other code running the web page the chance to react to those actions. See *Synchronizing with MathJax* for more details, and *Using Signals* in particular for examples of how to specify and use MathJax *Signal* objects.

The *Callback Signal* object is a subclass of the *Callback Queue* object.

Properties

name
The name of the signal. Each signal is named so that various components can access it. The first one to request a particular signal causes it to be created, and other requests for the signal return references to the same object.

posted
Array used internally to store the post history so that when new listeners express interests in this signal, they can be informed of the signals that have been posted so far. This can be cleared using the signal's `Clear()` method.

listeners
Array of callbacks to the listeners who have expressed interest in hearing about posts to this signal. When a post occurs, the listeners are called, each in turn, passing them the message that was posted.

Methods

Post(*message*[, *callback*])
Posts a message to all the listeners for the signal. The listener callbacks are called in turn (with the message as an argument), and if any return a *Callback* object, the posting will be suspended until the callback is executed. In this way, the `Post()` call can operate asynchronously, and so the *callback* parameter is used to synchronize with its operation; the *callback* will be called when all the listeners have responded to the post.

If a `Post()` to this signal occurs while waiting for the response from a listener (either because a listener returned a *Callback* object and we are waiting for it to complete when the `Post()` occurred, or because the listener itself called the `Post()` method), the new message will be queued and will be posted after the current

message has been sent to all the listeners, and they have all responded. This is another way in which posting can be asynchronous; the only sure way to know that a posting has occurred is through its *callback*. When the posting is complete, the callback is called, passing it the signal object that has just completed.

Returns the callback object (or a blank callback object if none was provided).

Parameters

- **message** — the message to send through the signal
- **callback** — called after the message is posted

Returns the callback or a blank callback

Clear(*[callback]*)

This causes the history of past messages to be cleared so new listeners will not receive them. Note that since the signal may be operating asynchronously, the `Clear()` may be queued for later. In this way, the `Post()` and `Clear()` operations will be performed in the proper order even when they are delayed. The *callback* is called when the `Clear()` operation is completed.

Returns the callback (or a blank callback if none is provided).

Parameters

- **callback** — called after the signal history is cleared

Returns the callback or a blank callback

Interest(*callback*[, *ignorePast*])

This method registers a new listener on the signal. It creates a *Callback* object from the callback specification, attaches it to the signal, and returns that *Callback* object. When new messages are posted to the signal, it runs the callback, passing it the message that was posted. If the callback itself returns a *Callback* object, that indicates that the listener has started an asynchronous operation and the poster should wait for that callback to complete before allowing new posts on the signal.

If *ignorePast* is `false` or not present, then before `Interest()` returns, the callback will be called with all the past messages that have been sent to the signal.

Parameters

- **callback** — called whenever a message is posted (past or present)
- **ignorePast** — `true` means ignore previous messages

Returns the callback object

NoInterest(*callback*)

This removes a listener from the signal so that no new messages will be sent to it. The callback should be the one returned by the original `Interest()` call that attached the listener to the signal in the first place. Once removed, the listener will no longer receive messages from the signal.

Parameters

- **callback** — the listener to be removed from signal

Returns `null`

MessageHook(*message*, *callback*)

This creates a callback that is called whenever the signal posts the given message. This is a little easier than having to write a function that must check the message each time it is called. Although the *message* here is a string, if a message posted to the signal is an array, then only the first element of that array is used to match against the message. That way, if a message contains an identifier plus arguments, the hook will match the identifier and still get called with the complete set of arguments.

Returns the *Callback* object that was produced.

Parameters

- **message** — the message to look for from the signal
- **callback** — called when the message is posted

Returns the callback object

`ExecuteHook` (*message*)
Used internally to call the listeners when a particular message is posted to the signal.

Parameters

- **message** — the posted message

Returns `null`

The MathJax.Localization Class

Beginning in version 2.2 of MathJax, all of MathJax's messages, menus, dialog boxes, and so are are localizable (meaning they can be presented in languages other than English). This is accomplished through the *MathJax.Localization* object. This object stores the data about the available languages, and the selected language, together with the routines needed to obtain the translated strings for the messages used by MathJax, and the ones used to register translations with the system.

Localizable strings in MathJax are identified by a unique ID (a character string used to obtain the translation), and MathJax has functions that obtain the translated message associated with the ID. Some messages need values inserted into them (like file names, or TeX macro names), and MathJax can insert those values into the translated string automatically. The localization system has support for plural and number forms, which differ from language to language. These issues are described in more detail in the *Localization Strings* documentation.

A number of MathJax's messaging functions handle localization of their messages automatically. For example, the `MathJax.Message.Set()` function and the TeX input jax's `Error()` function both will look up localization strings automatically.

Because the localization data needs to be downloaded over the network, MathJax only loads this data when it is actually needed (many users will only see mathematical expressions and will never need an actual translated message string, so there is no need to waste time downloading the localization data for them). Since MathJax loads files asynchonously, there is a synchronization issue that you need to be aware of when using localized message strings. There are support routines to help make this easier (these are described in more detail below).

Finally, MathJax consists of a number of relatively separate components, and can be extended by third-party plug-ins, it is possible that there would be name collisions with the ID's used to identify localizable strings. To make it easier to manage the string ID's, and to break up the localization data into smaller chunks that can be loaded quickly when needed, MathJax breaks up the messages into *domains*, each with its own set of ID's for the messages in that domain. Typically, a component (like the math menu, or the TeX input jax) has its own domain, so it can keep its message ID's separate from other components.

Getting a Translated String

The basic means of obtaining the string to use for a message to display to the user is to call the *_()* method of the *MathJax.Localization* object, passing the string id and the English phrase. For example,

```
MathJax.Localization._("TC","Typsetting Complete");
```

would return the string for "Typesettings Complete" in the currently selected language. This can be facilitated by defining the function

```
var _ = function () {return MathJax.Localization._.apply(MathJax.Localization,
↪arguments)}
```

so that you only need to use

```
_("TC","Typesetting Complete");
```

to obtain the translated string.

Both these examples take the translation from the default domain (the _ domain), but most components will want to use their own domain. For example, the TeX input jax uses the `TeX` domain. To request a translation from a specific domain, replace the ID with an array consisting of the domain and ID. For example

```
MathJax.Localization._(["TeX","MissingBrace"],"Missing Close Brace");
```

would get the string associated with the ID `MissingBrace` from the `TeX` domain in the current language. To make this easier, the TeX input jax could define

```
var _ = function (id) {
  return  MathJax.Localization._.apply(MathJax.Localization,
                  [["TeX",id]].concat([].slice.call(arguments,1)));
};
```

which appends the `TeX` domain automatically. With this definition, you could use the simpler form

```
_("MissingBrace","Missing Close Brace");
```

to get the `MissingBrace` message from the `TeX` domain.

Parameter Substitution

Some messages may want to include values (like file names, or TeX macro names) as part of their strings. The MathJax localization system provides a means of including such values in the translated strings. In addition to the ID and message strings, you pass the values that need to be substituted into the message, and use the special sequences `%1`, `%2`, etc. to indicate where they go within the message. For example

```
MathJax.Localization._("NotFound","File %1 not found",filename)
```

would obtain the translation for "File %1 not found" and insert the filename at the location of *%1* in the translated string.

There are also mechanisms of handling plural forms (which differ from language to language) and number forms. See the *Localization Strings* documentation for complete details.

HTML Snippets

MathJax allows you to encode HTML snippets using javascript data (see the *HTML snippets* documentation for details), and these often contain textual data that needs to be localized. You can pass HTML snippets to the *_()* function and a domain in which the strings are to be looked up. You then use a localization string (an array consisting of the ID and string, plus optional parameters to be substituted into the string) in place of a normal string in the HTML snippet. For example,

```
[
  "Follow this link: ",
  ["a",{href:"http://www.mathjax.org"},[
```

```
    ["img",{src:"external.gif"}]
  ]]
]
```

could be localized as

```
MathJax.Localization._("myDomain",[
  ["FollowLink","Follow this link"],": ",
  ["a",{href:"http://www.mathjax.org"},[
    ["img",{src:"external.gif"}]
  ]]
])
```

where the `FollowLink` ID is looked up in the `myDomain` domain of the current language.

See the HTML snippets section of the *Localization Strings* documentation for complete details.

Synchronization Issues

Because the translation data are stored in files that are loaded only when they are needed, and since file loading in MathJax is asynchronous, you need to take this loading process into account when you use *_()* to obtain a localized string. If this is the first string obtained from the language, or the first one from the requested domain, MathJax may have to load the data file or that language or domain (or both). In that case, you need to be prepared to wait for that file to load and retry obtaining the translation string. The localization system provides you with two functions to make this easier, but you do have to keep in mind that obtaining translation strings may be an asynchronous action.

The first method is `MathJax.Localization.loadDomain()`, which takes a domain name and an optional callback, and forces MathJax to load the language data for that domain (and the main language data file, if needed), then calls the callback. In this way, the callback function knows that the localization data that it needs will be available, and it doesn't have to worry about the possibility that *_()* will start a file loading operation. The *loadDomain()* function returns the callback object, which can be used in callback queues, for example, to coordinate further actions.

For example, suppose you want to perform the check

```
if (!url.match(/^https?:/)) {
  alert("Your url must use the http protocol");
  url = null;
}
```

and want to localize the error message. The naive approach would be

```
if (!url.match(/^https?:/)) {
  alert(_("BadProtocol","Your url must use the http protocol"));
  url = null;
}
```

(provided you have defined *_()* for your domain as described above). The problem is that *_()* might need to load the language data for your message, and that causes *_()* to throw a restart error. That would cause an error message to appear on the javascript console, and your alert would never occur. Instead, you want to make sure that the localization data are available before calling *_()*.

Suppose the domain for your message ID is `myDomain`, then one way to do this would be

```
if (!url.match(/^https?:/)) {
  MathJax.Localization.loadDomain("myDomain",function () {
    alert(_("BadProtocol","Your url must use the http or https protocol"));
  });
```

```
  url = null;
}
```

This uses `loadDomain` to force the `myDomain` data to be loaded before attemptin the *`_()`* call, so you are sure the call will succeed. If several localized string are needed, you may want to use `loadDomain` around the entire function:

```
MathJax.Localization.loadDomain("myDomain",function () {
  if (!url.match(/^https?:/)) {
    alert(_("BadProtocol","Your url must use the http or https protocol"));
    url = null;
  }
  if (url && !url.match(/\.js$/)) {
    alert(_("BadType","Your url should refer to a javascript file"));
  }
});
```

It is also possible to use *`loadDomain()`* as part of a callback queue:

```
MathJax.Callback.Queue(
  MathJax.Localization.loadDomain("myDomain"),
  function () {
    if (!url.match(/^https?:/)) {
      alert(_("BadProtocol","Your url must use the http or https protocol"));
      url = null;
    }
  }
);
```

Here the function will not be performed until after the `myDomain` domain is loaded.

The second tool for synchronizing with the localization system is the `MathJax.Localization.Try()` function. This method takes a callback specification (for example, a function, though it could be any valid callback data) and runs the callback with error trapping. If the callback throws a restart error (due to loading a localization data file), *`Try()`* will wait for that file to load, then rerun the callback (and will continue to do so if there are additional file loads).

Using this approach, you don't have to worry about loading the domains explicitly, as *`_()`* will throw a restart error when one is needed, and *`Try()`* will catch it and restart after the load. For example,

```
MathJax.Localization.Try(function () {
  if (!url.match(/^https?:/)) {
    alert(_("BadProtocol","Your url must use the http or https protocol"));
    url = null;
  }
});
```

Note that, as with *`loadDomain()`*, *`Try()`* may return before the callback has been run successfully, so you should consider this to be an asynchronous function. You can use callbacks to synchronize with other actions, if needed.

Also note that your function may be called multiple times before it succeeds (if localization data needs to be loaded). So you need to write the function in such a way that it doesn't matter if it gets partway through and fails. For example, you might not want to create structures or modify values that affect what happens if the function has to be rerun from the beginning when one of its *`_()`* causes a file load.

A number of functions in MathJax are able to accept localization strings as their inputs, and these already take care of the synchronization issues for you. For example, `MathJax.Message.Set()` can accept either a plain (untranslated) string, or a localization string (array with ID, string, and substitution parameters). It uses *`Try()`* internally to

make sure your message is properly translated before posting it to the screen. That means you don't have to worry about that yourself when you use `MathJax.Message.Set()`, though you shoud be aware that the posting of the message may be asynchronous, so the message might not be visible when *`Set()`* returns. Fortunately, `MathJax.Message.Clear()` coordinates with *`Set()`* so that even if you call *`Clear()`* before the original message posts, MathJax won't get confused). Similarly, the TeX input jax's `Error()` function handles the calling of *`_()`* and its synchronization for you.

The Localization Data

The `MathJax.Localization` object holds the data for the various translations, as well as the service routines for adding to the translations and retrieving translations.

Methods

The methods in `MathJax.Localization` include:

_(*id*, *message*[, *arguments*])
: The function (described in detail above) that returns the translated string for a given *id*, substituting the given *arguments* as needed.

  Parameters

  - **id** — the ID of the message to translate, or an array `[domain,ID]`
  - **message** — the English phrase to use as fallback if there is no translation, or an HTML snippet to be localized
  - **arguments** — values to be inserted into the translated string

  Returns the translated string or HTML snippet

`setLocale`(*locale*)
: Sets the selected locale to the given one, e.g.

```
MathJax.Localization.setLocale("fr");
```

  Parameters

  - **locale** — the two-character identifier for the desired locale

  Returns `null`

`addTranslation`(*locale*, *domain*, *def*)
: Defines (or adds to) the translation data for the given *locale* and *domain*. The *def* is the definition to be merged with the current translation data (if it exists) or to be used as the complete definition (if not). The data format is described below.

  Parameters

  - **locale** — the two-letter identifier for the locale to update or create
  - **domain** — the name of the domain to add or modify
  - **def** — the definition of the domain (see below)

  Returns `null`

`setCSS`(*div*)
: Sets the CSS for the given *div* to reflect the needs of the locale. In particular, it sets the font-family, if needed, and the direction (for right-to-left languages).

Parameters

- **div** — the DOM element whose CSS is to be modified

Returns the *div*

fontFamily()
Get the `font-family` needed to display text in the selected language. Returns `null` if no special font is required.

fontDirection()
Get the `direction` needed to display text in the selected language. Returns `null` if no special font is required.

plural(*n*)
The method that returns the index into the list of plural texts for the value *n*. See the [CLDR rules](http://unicode.org/cldr/charts/supplemental/language_plural_rules.html) for more information. This calls the locale's *plural()* method, if there is one, otherwise it defaults to the English version.

number(*n*)
The method that returns the localized version of the number *n*. This calls the locale's *number()* method, if there is one, otherwise it defaults to the English version.

loadDomain(*domain*[, *callback*])
This causes MathJax to load the data file for the given *domain* in the current language, and calls the *callback* when that is complete. If the domain is already loaded, the *callback* is called immediately. This lets you synchronize actions that require localization with the loading of the needed data so that you are sure that the needed translations are available. See the section on synchonization above for details.

Parameters

- **domain** — the name of the domain to load
- **callback** — the callback object to be run after loading

Returns the callback object (or a blank one if none specified)

Try(*fn*)
This method runs the function *fn* with error trapping and if an asynchronous file load is performed (for loading localizaton data), reruns the function again after the file loads. This lets you synchronize actions that require localization with the loading of the needed data (see the section on synchronization above for details). Note that the function should be one that can be run multiple times, if needed. Also note that *Try()* can return *before* the *fn* has been completed, so you should consider *fn* to be running asynchronously (you can use callbacks to synchronize with other actions, if needed).

Parameters

- **fn** — a callback specification for a function that uses localization data

Returns `null`

Properties

locale
The currently selected locale, e.g., `"fr"`. This is set by the *setLocale()* method, and should not be modified by hand.

directory
The URL for the localization data files. This can be overridden for individual languages or domains (see below). The default is `[MathJax]/localization`.

strings
: This is the main data structure that holds the translation strings. It consists of an entry for each language that MathJax knows about, e.g., there would be an entry with key `fr` whose value is the data for the French translation. Initially, these simply reference the files that define the translation data, which MathJax will load when needed. After the file is loaded, they will contain the translation data as well. This is described in more detail below.

Translation Data

Each language has its own data in the *MathJax.Localization.strings* structure. This structure holds data about the translation, plus the translated strings for each domain.

A typical example might be

```
fr: {
  menuTitle: "Fran\u00E7ais",                 // title used in language menu
  version: "1.0",
  directory: "[MathJax]/localization/fr",     // optional
  file: "fr.js",                              // optional (file contains the data␣
↪below)
  isLoaded: true,                             // set when loaded
  fontFamily: "...",                          // optional
  plural: function (n) {...},                 // optional implementation of plural␣
↪forms
  number: function (n) {...},                 // optional implementation of number␣
↪forms

  domains: {
    "_": {
      version: "1.0",
      file: "http://somecompany.com/MathJax/localization/fr/hub.js",  //  optional␣
↪(contains the rest of the data)
      isLoaded: true,
      strings: {
        fnf: "File '%1' not found",
        fl: "%1 %{plural:%1|file|files|} loaded",
        ...
      }
    },
    TeX: {
      ...
    },
    MathMenu: {
      ...
    }
    ...
  }
}
```

The fields have the following meanings:

menuTitle
: The string used for the menu item in the language submenu (it should be in the language itself, not English).

version
: The version of the translation data.

directory

An optional value that can be used to override the directory where the translation files for this language are stored. The default is to add the locale identifier to the end of `MathJax.Localization.directory`, so the value given in the example above is the default value, and could be omitted.

file
The name of the file containing the translation data for this language. The default is the locale identifier with `.js` appended, so the value given in the example above is the default value, and could be omitted.

isLoaded
This is set to `true` when MathJax has loaded the data for this language. Typically, when a language is registered with MathJax, the data file isn't loaded at that point. It will be loaded when it is first needed, and when that happens, this value is set.

fontFamily
This is a CSS font-family (or list of font-families) that should be used when text in this language is displayed. If not present, then no special font is needed.

fontDirection
This is a string `ltr` or `rtl` that specifies if the language is left-to-right or right-to-left. If not present, `ltr` will be assumed.

plural(n)
This is an optional function that returns the index into the list of plural values apropriate for the given integer n. If not provided, the English *`plural()`* function is used.

plural(n)
This is an optional function that returns the index into the list of plural values apropriate for the given integer *n*. If not provided, the English *`plural()`* function is used.

number(n)
This is an optional function that returns the a string representing the decimal number *n* in the format used by the given locale. If not provided, the English *`number()`* function is used.

domains
This is an object that contains the translation strings for this language, grouped by domain. Each domain has an entry, and its value is an object that contains the translation strings for that domain. The format is described in more detail below.

Domain Data

Each domain for which there are translations has an entry in the locale's domains object. These store the following information:

version
The version of the data for this domain.

file
If the domain data is stored in a separate file from the rest of the language's data (e.g., a third-party extension that is not stored on a CDN may have translation data that is provied by the thrid-party), this property tells where to obtain the translation data. In the example above, the data is provided by another company via a complete URL. The default value is the locale's directory with the domain name appended and *.js* appended to that.

isLoaded
This is set to `true` when the data file has been loaded.

strings
This is an object that contains that actual translated strings. The keys are the message identifiers described in the overview section above, and the values are the translations

Registering a Translation

Typically, for languages stored on a CDN, MathJax will register the language with a call like

```
MathJax.Localization.addTranslation("fr",null,{});
```

which will create an `fr` entry in the localization data that will be tied to the `[MathJax]/localization/fr` directory, and the `[MathJax]/localization/fr/fr.js` file. That directory could contain individual files for the various domains, or the `fr.js` file itself could contain combined data that includes the most common domains, leaving only the lesser-used domains in separate files.

An example `fr.js` file could be

```
MathJax.Localization.addTranslation("fr",null,{
  menuTitle: "Fran\u00E7ais",
  version: "1.0",
  domains: {
    "_": {},
    TeX: {},
    MathMenu: {}
  }
});
```

This would declare that there are translation files for the _, `TeX`, and `MathMenu` domains, and that these will be loaded individually from their default file names in the default directory of `[MathJax]/localization/fr`. Other domains will not be translated unless they register themselves via a command like

```
MathJax.Localization.addTranslation("fr","HelpDialog",{});
```

in which case the domain's data file will be loaded automatically when needed.

One could preload translation strings by including them in the `fr.js` file:

```
MathJax.Localization.addTranslation("fr",null,{
  menuTitle: "Fran\u00E7ais",
  version: "1.0",
  domains: {
    "_": {
      isLoaded: true,
      strings: {
        'NotFound': "Fichier `%1` non trouvé",
        ...
      }
    },
    TeX: {
      isLoaded: true,
      strings: {
        'MissingBrace': "Accolade de fermeture manquante",
        ...
      }
    },
    MathMenu: {}
  }
});
```

Here the _ and `TeX` strings are preloaded, while the `MathMenu` strings will be loaded on demand.

A third party extension could include

```
MathJax.Localization.addTranslation("fr","myExtension",{
  file: "http://myserver.com/MathJax/localization/myExtension/fr.js"
});
```

to add French translations for the `myExtension` domain (used by the extension) so that they would be obtained from the third-party server when needed.

A third party could provide a translation for a language not covered by a CDN by using

```
MathJax.Localization.addTranslation("kr",null,{
  menuTitle: "\uD55C\uAD6D\uB9D0",
  fontFamily: "Butang, 'Arial unicode MS', AppleMayungjo",
  directory: "http://mycompany.com/MathJax/localization/kr"
});
```

and providing a `kr.js` file in their `MathJax/localization/kr` directory that defines the details of their translation. If the Korean (kr) locale is selected, MathJax will load `http://mycompany.com/MathJax/localization/kr/kr.js` and any other domain files when they are needed.

See the subdirectories in the `MathJax/localization` directory for examples of language files. The English directory (en) is not actually used by MathJax (because the English strings a built in), but it can serve as an example and starting point for producing your own translations.

The Translation Files

Version 2.2 of MathJax comes with translations for French and German. Additional languages will be made available as they are developed. We hope to use community-based websites like Transifex to help produce these translations. Currently, however, the language data files are not in a form that can be used by these sites, so the only way to generate new translations is to copy the English data files and modify them for the new language.

In the future, MathJax will provide conversion programs that create the files needed for such sites in the formats they require (e.g., YAML), and that convert the translated versions back into the data files needed by MathJax, but these programs are not yet ready.

In addition, there will be a program that scans the MathJax files to obtain the ID's and English strings that are needed for the translation files. This will make maintenance of language files easier in the future, but these are not available yet.

The MathJax.InputJax Class

Input jax are the components of MathJax that translate mathematics from its original format (like *TeX* or *MathML*) to the MathJax internal format (an *element jax*).

An input jax is stored as a pair of files in a subdirectory of the `jax/input` directory, with the subdirectory name being the name of the input jax. For example, the TeX input jax is stored in *jax/input/TeX*. The first file, `config.js`, is loaded when MathJax is being loaded and configured, and is indicated by listing the input jax directory in the *jax* array of the MathJax configuration. The `config.js` file creates a subclass of the *MathJax.InputJax* object for the new input jax and registers that with MathJax, along with the MIME-type that will be used to indicate the mathematics that is to be processed by the input jax.

The main body of the input jax is stored in the second file, `jax.js`, which is loaded when the input jax is first called on to translate some mathematics. This file augments the original input jax subclass with the additional methods needed to do the translation. MathJax calls the input jax's *`Translate()`* method when it needs the input jax to translate the contents of a math `<script>` tag.

The *MathJax.InputJax* class is a subclass of the *MathJax Jax* class, and inherits the properties and methods of that class. Those listed below are the additional or overridden ones from that class.

Properties

id
The name of the jax.

version
The version number of the jax.

directory
The directory where the jax files are stored (e.g., "[MathJax]/jax/input/TeX").

elementJax
The name of the ElementJax class that this input jax will produce (typically mml, as that is the only ElementJax at the moment).

Methods

Process (*script*, *state*)
This is the method that the MathJax.Hub calls when it needs the input jax to process the given math <script>. Its default action is to do the following:

1. Start loading any element jax specified in the elementJax array;
2. Start loading the jax's jax.js file;
3. Start loading the required output jax (so it is ready when needed); and
4. Redefine itself to simply return the callback for the load operation (so that further calls to it will cause the processing to wait for the callback).

Once the jax.js file has loaded, this method is replaced by the jax's Translate() method (see below), so that subsequent calls to Process() will perform the appropriate translation.

Parameters

- **script** — reference to the DOM <script> object for the mathematics to be translated
- **state** — a structure containing information about the current proccessing state of the mathematics (internal use)

Returns an *ElementJax* object, or null

Translate (*script*, *state*)
This is the main routine called by MathJax when a <script> of the appropriate type is found. The default *Translate()* method throws an error indicating that *Translate()* hasn't been defined, so when the jax.js file loads, it should override the default *Translate()* with its own version that does the actual translation.

The translation process should include the creation of an *Element Jax* that stores the data needed for this element.

Parameters

- **script** — the <script> element to be translated
- **state** — a structure containing information about the current proccessing state of the mathematics (internal use)

Returns the *element jax* resulting from the translation

Register (*mimetype*)
This registers the MIME-type associated with this input jax so that MathJax knows to call this input jax when it sees a `<script>` of that type. An input jax can register more than one type, but it will be responsible for distinguishing elements of the various types from one another.

Parameters

- **mimetype** — the MIME-type of the input this jax processes

Returns `null`

needsUpdate (*jax*)
This implements the element jax's `needsUpdate()` method, and returns `true` if the `jax` needs to be rerendered (i.e., the text has changed), and `false` otherwise.

Perameters

- **jax** — the element jax to be checked

Returns `true` if the jax's text has changed, `false` otherwise

The MathJax.OutputJax Class

Output jax are the components of MathJax that translate mathematics from the MathJax internal format (an *element jax*) to whatever output is required to represent the mathematics (e.g., MathML elements, or HTML-with-CSS that formats the mathematics on screen).

An output jax is stored as a pair of files in a subdirectory of the the `jax/output` directory, with the subdirectory name being the name of the output jax. For example, the NativeMML output jax is stored in *jax/output/NativeMML*. The first file, `config.js`, is loaded when MathJax is being loaded and configured, and is indicated by listing the input jax directory in the *jax* array of the MathJax configuration. The `config.js` file creates a subclass of the *MathJax.OutputJax* object for the new output jax and registers it with MathJax, along with the MIME-type of the element jax that it can process.

The main body of the output jax is stored in the second file, `jax.js`, which is loaded when the output jax is first called on to translate some mathematics. This file augments the original output jax subclass with the additional methods needed to produce the output. MathJax calls the input jax's *Translate()* method when it needs the output jax to translate an element jax to produce output.

The *MathJax.OutputJax* class is a subclass of the *MathJax Jax* class, and inherits the properties and methods of that class. Those listed below are the additional or overridden ones from that class.

Properties

id
The name of the jax.

version
The version number of the jax.

directory
The directory where the jax files are stored (e.g., `"[MathJax]/jax/output/HTML-CSS"`);

fontDir
The directory where the fonts are stored (e.g., `"[MathJax]/fonts"`)

imageDir
The directory where MathJax images are found (e.g. `"[MathJax]/images"`)

Methods

preProcess(*state*)

This is called by `MathJax.Hub` to ask the output processor to prepare to process math scripts. Its default action is to start loading the jax's `jax.js` file, and redefine itself to simply return the callback for the load operation (so that further calls to it will cause the processing to wait for the callback).

Once the `jax.js` file has loaded, this method is replaced by the jax's *preTranslate()* method, so that subsequent calls to *preProcess()* will perform the appropriate translation.

Parameters

- **state — a structure containing information about the** current proccessing state of the mathematics

Returns `null`

preTranslate(*state*)

This routine replaces *preProcess()* above when the jax's `jax.js` file is loaded. It is called by `MathJax.Hub` to ask the output processor to prepare to process math scripts. (For example, the HTML-CSS output jax uses this to determine em-sizes for all the mathematics at once, to minimize page reflows that slow down Internet Explorer.)

The routine can use `state.jax[this.id]` to obtain the array of element jax that are to be processed. The output jax can use the `state` variable to maintain its own state information, but any properties that it adds to the variable should have a prefix that is the output jax's ID. For example, the HTML-CSS output jax might use `state.HTMLCSSlast` to keep track of the last equation it processed, or could add `state.HTMLCSS = {...}` to create an object of its own within the state variable.

Parameters

- **state** — a structure containing information about the current proccessing state of the mathematics

Returns `null`

Translate(*script*, *state*)

This is the main routine called by MathJax when an element jax is to be converted to output. The default *Translate()* method throws an error indicating that *Translate()* hasn't been defined, so when the `jax.js` file loads, it should override the default *Translate()* with its own version that does the actual translation.

You should use `MathJax.Hub.getJaxFor(script)` to obtain the element jax for the given script. The translation process may modify the element jax (e.g., if it has data that needs to be stored with the jax), and may insert DOM elements into the document near the jax's `<script>` tag. The output jax can use the `state` variable to maintain information about its processing state, but see *preTranslate()* above for naming conventions for properties that are added.

Parameters

- **script** — the `<script>` element to be translated
- **state** — a structure containing information about the current proccessing state of the mathematics

Returns the *element jax* resulting from the translation

postTranslate(*state*)

This routines is called by `MathJax.Hub` when the translation of math elements is complete, and can be used by the output processor to finalize any actions that it needs to complete. (For example, making the mathematics visible, or forcing a reflow of the page.)

The routine can use `state.jax[this.id]` to obtain the array of element jax that were processed, or can use the `state` variable to store its own state information (see *preProcess()* above for caveats about naming properties).

Parameters

- **state** — a structure containing information about the current proccessing state of the mathematics

Returns `null`

Register(*mimetype*)

This registers the MIME-type for the element jax associated with this output jax so that MathJax knows to call this jax when it wants to display an element jax of that type. Several output jax may register for the same input jax, in which case the first one to register will be the default one for that type.

Parameters

- **mimetype** — the MIME-type of the input this jax processes

Returns `null`

Remove(*jax*)

Removes the output associated with the given element jax. The routine can use `jax.SourceElement()` to locate the `<script>` tag associated with the element jax.

Parameters

- **jax** — the element jax whose display should be removed

Returns `null`

If an output jax wants its output to handle the contextual menu item and zooming, then it needs to tie into the event-handling code (*MathEvents*) and the zoom-handling code (*MathZoom*). That requires the following methods.

getJaxFromMath(*math*)

This is called by the event-handling code (*MathEvents*) to get the element jax associated with the DOM element that caused an event to occur. The output jax will have attached event handlers to some DOM element that is part of its output, and the *MathEvents* code uses this routine to map back to the jax associated with that output.

Parameters

- **math** — a DOM element that triggered a DOM event (e.g., a mouse click)

Returns the *ElementJax* structure associated with the DOM element

Zoom(*jax*, *span*, *math*, *Mw*, *Mh*)

This routine is called by the zoom-handling code (*MathZoom*) when an expression has received its zoom trigger event (e.g., a double-click). The `jax` is the math that needs to be zoomed, `span` is a `<span>` element in which the zoomed version of the math should be placed, `math` is the DOM element that received the zoom trigger event, and `Mw` and `Mh` are the maximum width and height allowed for the zoom box (the `span`).

The return value is an object with the following properties:

- **`Y` — the vertical offset from the top of the `span` to** the baseline of the mathematics
- `mW` — the width of the original mathematics element
- `mH` — the height of the original mathematics element
- `zW` — the width of the zoomed math
- `zH` — the height of the zoomed math

All of these values are in pixels.

Parameters

- **jax** — the jax to be zoomed
- **span** — the `<span>` in which to place the zoomed math
- **math** — the DOM element generating the zoom event
- **Mw** — the maximum width of the zoom box
- **Mh** — the maximum height of the zoom box

Returns a structure as described above

The MathJax.ElementJax Class

The element jax is the bridge between the input and output jax, and contains the data produced by the input jax needed by the output jax to display the results. It is tied to the individual `<script>` tag that produced it, and is the object used by JavaScript programs to interact with the mathematics on the page.

An element jax is stored in the `jax.js` file in a subdirectory of the `jax/element` directory, with the subdirectory name being the name of the element jax. Currently, there is only one element jax class, the *mml* element jax, and it is stored in `jax/element/mml`.

The *MathJax.ElementJax* class is a subclass of the *MathJax Jax* class, and inherits the properties and methods of that class. Those listed below are the additional or overridden ones from that class.

Class Properties

`id`
The name of the jax.

`version`
The version number of the jax.

`directory`
The directory where the jax files are stored (e.g., `"[MathJax]/jax/element/mml"`).

Instance Properties

`inputJax`
The name of the input jax that created the element.

`outputJax`
The name of the output jax that has processed this element.

`inputID`
The DOM *id* of the `<script>` tag that generated this element (if it doesn't have one initially, the MathJax hub will supply one). Note that this is not a reference to the element itself; that element will have a reference to this element jax, and if *inputID* were a reference back, that would cause a reference loop, which some browsers would not free properly during trash collection, thus causing a memory leak.

`originalText`
A string indicating the original input text that was processed for this element. (In the future, this may be managed by the input jax rather than `MathJax.Hub`.)

`mimeType`
The MIME-type of the element jax (*jax/mml* in the case of an *mml* element jax).

Other data specific to the element jax subclass may also appear here.

Methods

Text (*text*[, *callback*])
Sets the input text for this element to the given text and reprocesses the mathematics. (I.e., updates the equation to the new one given by *text*). When the processing is complete, the *callback*, if any, is called.

Parameters

- **text** — the new mathematics source string for the element
- **callback** — the callback specification

Returns the callback object

Rerender ([*callback*])
Removes the output and produces it again (for example, if CSS has changed that would alter the spacing of the mathematics). Note that the internal representation isn't regenerated; only the output is. The *callback*, if any, is called when the process completes.

Parameters

- **callback** — the callback specification

Returns the callback object

Reprocess ([*callback*])
Removes the output and then retranslates the input into the internal form and reredners the output again. The *callback*, if any, is called when the process completes.

Parameters

- **callback** — the callback specification

Returns the callback object

Remove()
Removes the output for this element from the web page (but does not remove the original `<script>`). The `<script>` will be considered unprocessed, and the next call to `MathJax.hub.Typeset()` will re-display it.

Returns `null`

SourceElement()
Returns a reference to the original `<script>` DOM element associated to this element jax.

Returns the `<script>` element

needsUpdate()
Indicates whether the mathematics has changed so that its output needs to be updated.

Returns `true` if the mathematics needs to be reprocessed, `false` otherwise

Output jax may add new methods to the base element jax class to perform exporting to other formats. For example, a MathML output jax could add `toMathML()`, or an accessibility output jax could add `toAudible()`. These could be made available via the MathJax contextual menu.

The Base Jax Class

The *MathJax.InputJax*, *MathJax.OutputJax* and *MathJax.ElementJax* classes are all subclasses of the base *Jax* class in MathJax. This is a private class that implements the methods common to all three other jax classes.

Unlike most MathJax.Object classes, calling the class object creates a *subclass* of the class, rather than an instance of the class. E.g.,

```
MathJax.InputJax.MyInputJax = MathJax.InputJax({
  id: "MyInputJax",
  version: "1.0",
  ...
});
```

creates `MathJax.InputJax.MyInputJax` as a subclass of `MathJax.InputJax`.

Class Properties

directory
: The name of the jax directory (usually `"[MathJax]/jax"`). Overridden in the subclass to be the specific directory for the class, e.g. `"[MathJax]/jax/input"`.

extensionDir
: The name of the extensions directory (usually `"[MathJax]/extensions"`).

Instance Properties

id
: The name of the jax.

version
: The version number of the jax.

directory
: The directory for the jax (e.g., `"[MathJax]/jax/input/TeX"`).

require: null
: An array of files to load before the `jax.js` file calls the `MathJax.Ajax.loadComplete()` method.

config: {}
: An object that contains the default configuration options for the jax. These can be modified by the author by including a configuration subsection for the specific jax in question.

JAXFILE: "jax.js"
: The name of the file that contains the main code for the jax.

Methods

Translate(*script*)
: This is a stub for a routine that should be defined by the jax's `jax.js` file when it is loaded. It should perform the translation action for the specific jax. For an input jax, it should return the *ElementJax* object that it created. The *Translate()* method is never called directly by MathJax; during the `loadComplete()` call, this function is copied to the *Process()* method, and is called via that name. The default *Translate()* method throws an error indicating that the *Translate()* method was not redefined. That way, if the `jax.js` file fails to load for some reason, you will receive an error trying to process mathematics with this jax.

Parameters

- **script** — reference to the DOM `<script>` object for the mathematics to be translated

Returns an *ElementJax* object, or `null`

Register(*mimetype*)
This method is overridden in the *InputJax*, *OutputJax* and *ElementJax* subclasses to handle the registration of those classes of jax.

Parameters

- **mimetype** — the MIME-type to be associated with the jax

Returns `null`

Config()
Inserts the configuration block for this jax from the author's configuration specification into the jax's `config` property. If the configuration includes an `Augment` object, that is used to augment the jax (that is, the configuration can override the methods of the object, as well as the data). This is called automatically during the loading of the `jax.js` file.

Startup()
This is a method that can be overridden in the subclasses to perform initialization at startup time (after the configuration has occurred).

loadComplete(*file*)
This is called by the `config.js` and `jax.js` files when they are completely loaded and are ready to signal that fact to MathJax. For `config.js`, this simply calls the `MathJax.Ajax.loadComplete()` method for the `config.js` file. For `jax.js`, the actions performed here are the following:

1. Post the "[name] Jax Config" message to the startup signal.
2. Perform the jax's *`Config()`* method.
3. Post the "[name] Jax Require" message to the startup signal.
4. Load the files from the jax's `require` and `config.extensions` arrays.
5. Post the "[name] Jax Startup" message to the startup signal.
6. Perform the jax's *`Startup()`* method.
7. Post the "[name] Jax Ready" message to the startup signal.
8. Copy the `preTranslate`, `Translate`, and `postTranslate` functions to `preProcess`, `Process`, and `postProcess`.
9. Perform the `MathJax.Ajax.loadComplete()` call for the `jax.js` file.

Note that the configuration process (the *`Config()`* call) can modify the `require` or `config.extensions` arrays to add more files that need to be loaded, and that the *`Startup()`* method isn't called until those files are completely loaded.

The MathJax Object-Oriented Programming Model

MathJax uses an object-oriented programming model for its main components, such as the *Input jax*, *Output jax*, and *Element jax*. The model is intended to be light-weight and is based on JavaScript's prototype inheritance mechanism. Object classes are created by making subclasses of *MathJax.Object* or one of its subclasses, and are instantiated by calling the object class as you would a function.

For example:

```
MathJax.Object.Foo = MathJax.Object.Subclass({
  Init: function (x) {this.setX(x)},
  getX: function () {return this.x},
  setX: function (x) {this.x = x}
});
```

```
var foo = MathJax.Object.Foo("bar");
foo.getX();                 // returns "bar"
foo.setX("foobar");
foo.getX();                 // returns "foobar"
```

Object classes can have static properties and methods, which are accessed via the object class variable. E.g., `MathJax.Object.Foo.SUPER` or `MathJax.Object.Foo.Augment()` for the object in the example above. Static values are not inherited by subclasses.

Static Properties

SUPER
Pointer to the super class for this subclass. (It is a reference to *MathJax.Object* in the example above.)

Static Methods

Subclass (*def*[, *static*])
Creates a subclass of the given class using the contents of the *def* object to define new methods and properties of the object class, and the contents of the optional *static* object to define new static methods and properties.

Parameters

- **def** — object that defines the properties and methods
- **static** — object that defines static properties and methods

Returns the new object class

Augment (*def*[, *static*])
Adds new properties and methods to the class prototype. All instances of the object already in existence will receive the new properties and methods automatically.

Parameters

- **def** — object that defines the properties and methods
- **static** — object that defines static properties and methods

Returns the object class itself

Properties

constructor
Pointer to the constructor function for this class. E.g., `foo.constructor` would be a reference to `MathJax.Object.Foo` in the example above.

Methods

Init ([*data*])
An optional function that is called when an instance of the class is created. When called, the *this* variable is set to the newly instantiated object, and the *data* is whatever was passed to the object constructor. For instance, in the example above, the variable `foo` is created by calling `MathJax.Object.Foo("bar")`, which calls the `MathJax.Object.Foo` object's *Init()* method with *data* equal to `"bar"`. If desired, the *Init()* method can create a *different* object, and return that, in which case this becomes the return value for the object constructor.

Parameters

- **data** — the data from the constructor call

Returns `null` or the object to be returned by the constructor

isa (*class*)
Returns `true` if the object is an instance of the given class, or of a subclass of the given class, and `false` otherwise. So using the `foo` value defined above,

```
foo.isa(MathJax.Object);       // returns true
foo.isa(MathJax.Object.Foo);   // returns true
foo.isa(MathJax.InputJax);     // returns false
```

can (*method*)
Checks if the object has the given *method* and returns `true` if so, otherwise returns `false`. This allows you to test if an object has a particular function available before trying to call it (i.e., if an object implements a particular feature). For example:

```
foo.can("getX");  // returns true
foo.can("bar");   // returns false
```

has (*property*)
Checks if the object has the given *property* and returns `true` if so, otherwise returns `false`. This allows you to test if an object has a particular property available before trying to use it. For example:

```
foo.has("getX");  // returns true
foo.has("x");     // returns true
foo.has("bar");   // returns false
```

Accessing the Super Class

If a subclass overrides a method of its parent class, it may want to call the original function as part of its replacement method. The semantics for this are a bit awkward, but work efficiently. Within a method, the value `arguments.callee.SUPER` refers to the super class, so you can access any method of the superclass using that. In order to have *this* refer to the current object when you call the super class, however, you need to use `call()` or `apply()` to access the given method.

For example, `arguments.callee.SUPER.method.call(this,data)` would call the superclass' *method* and pass it *data* as its argument, properly passing the current object as *this*. Alternatively, you can use `this.SUPER(arguments)` in place of `arguments.callee.SUPER`. It is also possible to refer to the super class explicitly rather than through `arguments.callee.SUPER`, as in the following example:

```
MathJax.Class1 = MathJax.Object.Subclass({
  Init: function(x) {this.x = x},
  XandY: function(y) {return "Class1: x and y = " + this.x + " and " + y}
});

MathJax.Class2 = MathJax.Class1.Subclass({
  XandY: function (y) {return "Class2: "+arguments.callee.SUPER.XandY.call(this,y)}
});

MathJax.Class3 = MathJax.Class2.Subclass({
  XandY: function (y) {return "Class3: "+MathJax.Class2.prototype.XandY.call(this,y)}
});

MathJax.Class4 = MathJax.Class1.Subclass({
```

```
  XandY: function (y) {return "Class4: "+this.SUPER(arguments).XandY.call(this,y)}
});

var foo = MathJax.Class2("foo");
foo.XandY("bar");   // returns "Class2: Class1: x and y = foo and bar"
var bar = MathJax.Class3("bar");
bar.XandY("foo");   // returns "Class3: Class2: Class1: x and y = bar and foo"
var moo = MathJax.Class4("moo");
moo.XandY("cow");   // returns "Class4: Class1: x and y = moo and cow"
```

Since both of these mechanisms are rather awkward, MathJax provides an alternative syntax that is easier on the programmer, but at the cost of some inefficiency in creating the subclass and in calling methods that access the super class.

Since most calls to the super class are to the overridden method, not to some other method, the method name and the `call()` are essentially redundant. You can get a more convenient syntax by wrapping the *def* for the *`Subclass()`* call in a call to `MathJax.Object.SimpleSUPER()`, as in the following example:

```
MathJax.Class1 = MathJax.Object.Subclass({
  Init: function (x) {this.x = x},
  XandY: function (y) {return "Class1: x and y = " + this.x + " and " + y}
});

MathJax.Class2 = MathJax.Class1.Subclass(
  MathJax.Object.SimpleSUPER({
    XandY: function (y) {return "Class2: "+this.SUPER(y)},
    AnotherMethod: function () {return this.x}              // it's OK if a method
↪doesn't use SUPER
  })
);

var foo = MathJax.Class2("foo");
foo.XandY("bar");     // returns "Class2: Class1: x and y = foo and bar"
```

CHAPTER 5

Reference Pages

Describing HTML snippets

A number of MathJax configuration options allow you to specify an HTML snippet using a JavaScript object. This lets you include HTML in your configuration files even though they are not HTML files themselves. The format is fairly simple, but flexible enough to let you represent complicated HTML trees.

An HTML snippet is an array consisting of a series of elements that format the HTML tree. Those elements are one of two things: either a string, which represents text to be included in the snippet, or an array, which represents an HTML tag to be included. In the latter case, the array consists of three items: a string that is the tag name (e.g., "img"), an optional object that gives attributes for the tag (as described below), and an optional HTML snippet array that gives the contents of the tag.

When attributes are provided, they are given as *name:value* pairs, with the *name* giving the attribute name, and *value* giving its value. For example

```
[["img",{src:"/images/mypic.jpg"}]]
```

represents an HTML snippet that includes one element: an `<img>` tag with `src` set to `/images/mypic.jpg`. That is, this is equivalent to

```
<img src="/images/mypic.jpg">
```

Note that the snippet has two sets of square brackets. The outermost one is for the array that holds the snippet, and the innermost set is because the first (and only) element in the snippet is a tag, not text. Note that the code `["img", {src:"/images/mypic.jpg"}]` is invalid as an HTML snippet. It would represent a snippet that starts with "img" as text in the snippet (not a tag), but the second item is neither a string nor an array, and so is illegal. This is a common mistake that should be avoided.

A more complex example is the following:

```
[
  "Please read the ",
  ["a",{href:"instructions.html"},["instructions"]],
```

```
  " carefully before proceeding"
]
```

which is equivalent to

```
please read the <a href="instructions.html">instructions</a> carefully
before proceeding.
```

A final example shows how to set style attributes on an object:

```
[["span",
  {
    id:"mySpan",
    style: {color:"red", "font-weight":"bold"}
  },
  [" This is bold text shown in red "]
]]
```

which is equivalent to

```
<span id="mySpan" style="color: red; font-weight: bold;">
This is bold text shown in red
</span>
```

Since HTML snippets contain text that is displayed to users, it may be important to localize those strings to be in the language selected by the user. See the *Localization Strings* documentation for details of how to accomplish that.

CSS Style Objects

Many MathJax components allow you to specify CSS styles that control the look of the elements they create. These are described using *CSS style objects*, which are JavaScript objects that represent standard CSS declarations. The main CSS style object is a collection of *name:value* pairs where the *name* is the CSS selector that is being defined, and the *value* is an object that gives the style for that selector. Most often, the selector will need to be enclosed in quotation marks, as it will contain special characters, so you would need to use `"#myID"` rather than just `#myID` and `"ul li"` rather than just `ul li`.

The value used to define the CSS style can either be a string containing the CSS definition, or a javascript object that is itself a collection of *name:value* pairs, where the *name* is the attribute being defined and *value* is the value that attribute should be given. Note that, since this is a JavaScript object, the pairs are separated by commas (not semi-colons) and the values are enclosed in quotation marks. If the name contains dashes, it should be enclosed in quotation marks as well.

For example, `jax/output/HTML-CSS/config.js` includes the following declaration:

```
styles: {

  ".MathJax_Display": {
    "text-align": "center",
    margin:       "1em 0em"
  },

  ".MathJax .merror": {
    "background-color": "#FFFF88",
    color:   "#CC0000",
    border:  "1px solid #CC0000",
```

```
    padding: "1px 3px",
    "font-style": "normal",
    "font-size":  "90%"
  }

}
```

This defines two CSS styles, one for the selector `.MathJax_Display`, which specifies its text alignment and margin settings, and a second for `.MathJax .merror`, which specifies a background color, foreground color, border, and so on.

You can add as many such definitions to a `styles` object as you wish. Note, however, that since this is a JavaScript object, the selectors must be unique (e.g., you can't use two definitions for `"img"`, for example, as only the last one would be saved). If you need to use more than one entry for a single selector, you can add comments like `/* 1 */` and `/* 2 */` to the selector to make them unique.

It is possible to include selectors like `"@media print"`, in which case the value is a CSS style object. For example:

```
styles: {
  "@media print": {
    ".MathJax .merror": {
       "background-color": "white",
       border: 0
    }
  }
}
```

The various extensions and output processors include more examples of CSS style objects, so see the code for those files for additional samples. In particular, the `extensions/MathMenu.js`, `extensions/MathZoom.js`, `extensions/FontWarnsing.js`, and `jax/output/HTML-CSS/jax.js` files include such definitions.

Localization Strings

In MathJax v2.2 and later, the user interface can be localized to use languages other than English. This includes all information strings, menu items, warning messages, and so on. To make this possible, each string is given an ID that is used to obtain the localized version of the string. So the "File not found" message might have the ID `NotFound`. The localization data for each language associates the ID `NotFound` with the proper translation of the English phrase "File not found".

Some of MathJax's functions, like `MathJax.Message.Set()`, can accept localized strings as their parameters. To use a localized string in this case, use an array consisting of the ID followed by the English string (followed by any substitution arguments for the string, see below). For example,

```
MathJax.Message.Set(["NotFound","File not found"]);
```

would cause the "File not found" message to be displayed (in the currently selected language) in the MathJax message area at the bottom of left of the browser window. Note that `MathJax.Message.Set()` can still accept unlocalized strings, as it has traditionally:

```
MathJax.Message.Set("File not found");
```

Here the message will always be in English, regardless of the selected language.

The reason that the English string is also given (in addition to the ID), is because MathJax needs to have a fallback string to use in case the localization data doesn't translate that ID, or if the localization data has failed to load. Providing the English string in addition to the ID guarantees that a fallback is available.

MathJax's localization system is documented more fully in the *Localization API* documentation.

Parameter Substitution

Localized messages may need to include information, like file names, that are not known until the message is needed. In this case, the message string acts as a template and MathJax will insert the needed values into it at the appropriate places. To use such substitutions, you include the values in the localization string array following the English phrase, and use the special sequences `%1`, `%2`, etc., to refer to the parameters at the locations where they should appear in the message. For example,

```
MathJax.Message.Set(["NotFound","File %1 not found",filename]);
```

would cause the name stored in the variable `filename` to be inserted into the localized string at the location of the `%1`. Note that the localized string could use the parameters in a different order from how they appear in English, so MathJax can handle languages where the word order is different.

Although it would be rare to need more than 9 substitution parameters, you can use `%10`, `%11`, etc., to get the 10-th, 11-th, and so on. If you need a parameter to be followed directly by a number, use `%{1}0` (rather than `%10`) to get the first parameter followed directly by a zero.

A `%` followed by anything other than a number or a `{` generates just the character following the percent sign, so `%%` would produce a single `%`, and `%:` would produce just `:`.

Plural forms

Some languages handle plural forms differently from how English does. In English, there are two forms: the one used for a single item, and the one used for everything else. For example, you would say "You have one new message" for a single message, but "You have three new messages" of there were three messages (or two, or zero, or anything other than one).

To handle plurals, you use a special "plural" directive within your message string. The format is

```
%{plural:%n|text1|text2}
```

where `%n` is the reference to the parameter (which should be a number) that controls which text to use. For English, there would be two texts, one (`text1`) for when the number is 1, and one (`text2`) for when it is anything else. Other languages may have more forms (e.g., Welsh has six different plural forms, a different one for 0 (zero), 1 (one), 2 (two), 3 (few), 6 (many), and anything else, so Welsh translation plural forms would have six different texts). The details of how to map the numeric value to the text strings is handled by the translation data for the selected language.

As an example, you might use

```
MathJax.Message.Set(["NewMessages","You have %1 new %{plural:%1|message|messages}",
↪n]);
```

where `n` is a variable holding the number of new messages.

Alternatively,

```
MathJax.Message.Set(["NewMessages","You have %{plural:%1|a new message|%1 new␣
↪messages}",n]);
```

shows how you can do substitution within the plural texts themselves.

Note that the translation string may contain such constructs even if the original English one doesn't. For example

```
MathJax.Message.Set(["alone","We are %1 in this family but alone in this World.",n]);
```

could be translated into French by

```
"Nous sommes %1 dans cette famille mais %{plural:%1|seul|seuls} en ce monde."
```

Note that if one of the options for the plural forms requires a literal close brace, it can be quoted with a percent. For instance,

```
%{plural:%1|One {only%}|Two {or more%}}
```

would produce `One {only}` when the first argument is 1, and `Two {or more}` otherwise.

If a message needs to include a literal string that looks like one of these selectors, the original `%` can be quoted. So `%%{plural:%%1|A|B}` would be the literal string `%{plural:%1|A|B}`.

Number forms

Decimal numbers are represented differently in different languages. For example, 3.14159 is an English representation of an approximation to the mathematical constant pi, while in European countries, it would be written 3,14159. MathJax will convert a number to the proper format before inserting it into a localized string. For example

```
MathJax.Message.Set(["pi","The value of pi is approximately %1",3.14159]);
```

would show the value as `3.14159` in English, but `3,14159` if French is the selected language.

ID's and Domains

Because MathJax consists of a number of separate components and can be extended by third party code, it is possible for two different components to want to use the same ID value for a string, leading to an ID name collision. To help avoid this, MathJax allows identifier *domains* that are used to isolate collections of identifiers for one component from those for another component. For example, each input jax has its own domain, as do many of the MathJax extensions. This means you only have to worry about collisions within your own domain, and so can more easily manage the uniqueness if the ID's in use.

To use a domain with your ID, replace the ID with an array consisting of the domain and the ID. For example, the TeX input jax uses the domain `TeX`, so

```
MathJax.Message.Set([["TeX","MissingBrace"],"Missing Close Brace"]);
```

would set the message to the translation associated with the ID `MissingBrace` in the `TeX` domain.

Some functions that take localization strings automatically prepend the domain to the ID (if one isn't already given). For example, the `Error()` function of the TeX input jax uses the `TeX` domain if one isn't supplied, so

```
TEX.Error(["MissingBrace","Missing Close Brace"]);
```

will generate the `MissingBrace` error from the `TeX` domain without having to specify the `TeX` domain explicitly.

HTML Snippets

MathJax provides a means of specifiying HTML code in javascript called *HTML snippets*. These frequently include text that needs to be localized, so you can include localization strings (like those described above) within an HTML snippet in any location where you would normally have a regular text string. For example, the snippet

```
[
  "Follow this link: ",
  ["a",{href:"http://www.mathjax.org"},[
    ["img",{src:"external.gif"}]
  ]]
]
```

includes the text "Follow this link:" which should be localized. You can change it to a localization string to cause it to be translated to the selected langauge:

```
[
  ["FollowLink","Follow this link"],": ",
  ["a",{href:"http://www.mathjax.org"},[
    ["img",{src:"external.gif"}]
  ]]
]
```

(Here we use the ID `FollowLink` to obtain the translation). Note that you can include substitution parameters as usual:

```
[
  ["ClickMessages","Click for %1 new %{plural:%1|messsage|messages}",n],": ",
  ["a",{href:"messages.html"},[
    ["img",{src:"external.gif"}]
  ]]
]
```

It is even possible to substitute HTML snippets into a localized string (when it is within an HTML snippet):

```
[
  ["MathJaxLink","This is documented at the %1 website",[
    ["a",{href:"http://docs.mathjax.org"},["MathJax]]
  ]]
]
```

Note, however, that a better approach to this exampe is given in the next section.

Since an HTML snippet might contain several strings that need to be localized, you may want to be able to specify the domain to use for *all* the strings within the snippet. Within a snippet, you can use an entry of the form `[domain, snippet]` to force the snippet to be processed with default domain `domain`. E.g.

```
[
  ["TeX",[
    ["ul",{},[
      ["li",{},[["MissingBrace","Missing close brace"]]],
      ["li",{},[["ExtraBrace","Extra close brace"]]],
    ]]
  ]],
  ["MathML",[
    ["ul",{},[
      ["li",{},[["UnknownNode","Unknown node type: %1",type]]],
      ["li",{},[["BadAttribute","Illegal attribute: %1",attr]]],
    ]]
  ]
]
```

would create two undordered lists, one with translations from the `TeX` domain, and one from the `MathML` domain.

To summarize the format of an HTML snippet, it is an array with each entry being one of the following:

- A text string, which becomes text in the resulting HTML; this is untranslated.
- An array of the form `["tag"]`, `["tag",{properties}]`, or `["tag",{properties},HTML-snippet]`, which becomes the given HTML tag, with the given properties, containing the given HTML-snippet as its children.
- An array of the form `[id,message]` or `[id,message,parameters]`, which is first translated, then parameter substitution performed, and the result added to the HTML (either as text or as HTML tags if the message included Markdown syntax). Note that the `id` can be either an id or an array `[domain,id]`, and that the parameters could be HTML snippets themselves.
- An array of the form `[domain,HTML-snippet]`, which becomes the HTML-snippet with its localizations done from the given domain.

Markdown Notation

HTML snippets allow you to create styled markup, like bold or italics, but this requires breaking the text up into smaller strings that fall in between HTML tags. That makes it hard to translate, since the strings are not full phrases. To make the creation of strings with bold, italics, and hyperlinks easier to localize, MathJax allows the strings within HTML snippets to be written in a limited Markdown syntax (*very* limited). You can use `*bold*`, `**italics**`, `***bold-italics***`, `[link-text](url)`, and `` `code` `` to obtain bold, italics, bold-italics, hyperlinks, and code blocks. For instance, the link example above could be more easily handled via

```
[
  ["MathJaxLink","This is documented at the [MathJax](%1) website",
                 "http://docs.mathjax.org"]
]
```

while

```
[
  ["Renderer","*Renderer*: lets you select the output renderer"]
]
```

will produce the equivalent of `<b>Renderer</b>: lets you select the output render` in the appropriate language.

Glossary

AsciiMath A notation for mathematics that uses characters commonly available on all computer keyboards to represent the math in an algebra-like syntax that should be intuitive and easily read.

See also:

AsciiMath home page

Callback A JavaScript function that is used to perform actions that must wait for other actions to complete before they are performed.

Callback Queue MathJax uses *Queues* to synchronize its activity so that actions that operate asynchronously (like loading files) will be performed in the right order. *Callback* functions are pushed onto the queue, and are performed in order, with MathJax handling the synchronization if operations need to wait for other actions to finish.

Callback Signal A JavaScript object that acts as a mailbox for MathJax events. Like an event handler, but it also keeps a history of messages. Your code can register an "interest" in a signal, or can register a *callback* to be called when a particular message is sent along the signal channel.

HTML-CSS MathJax output form that relys only on HTML and CSS 2.1, allowing MathJax to remain compatible across all browsers.

jax MathJax's input and output processors are called "jax", as is its internal format manager. The code for the jax are in the `MathJax/jax` directory.

LaTeX LaTeX is a variant of *TeX* that is now the dominant TeX style.

See also:

LaTeX Wikipedia entry

Markdown A text format commonly used in blogs and wikis for creating web pages without the need for complicated markup notation. It is intended to be an easy-to-read and easy-to-write format that still gives you the ability to specify a rich text result (including things like bold, italics, bullet lists, and so on).

See also:

Markdown home page

MathML An XML specification created to describe mathematical notations and capture both its structure and content. MathML is much more verbose than *TeX*, but is much more machine-readable.

See also:

MathML Wikipedia entry

STIX The Scientific and Technical Information Exchange font package. A comprehensive set of scientific glyphs.

See also:

STIX project

SVG Acronym for *Scalable Vector Graphics*. SVG is a graphics format that allows images to be described as a collection of graphics objects (like lines, rectangles, etc) rather than as a bitmap of colored pixels. MathJax can use this format to display mathematics as an alternative to its HTML-CSS or NativeMML output.

See also:

SVG Wikipedia entry

TeX A document markup language with robust math markup commands developed by Donald Knuth in the late 1970's, but still in extensive use today. It became the industry standard for typesetting of mathematics, and is one of the most common formats for mathematical journals, articles, and books.

See also:

TeX Wikipedia entry

- Search

This version of the documentation was built Apr 10, 2017.

Index

P

Q

R

S

T

U

W

Z