MathJax Documentation

Release 1.1

Davide Cervone, Casey Stark, Robert Miner, Paul Topping

May 01, 2014

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

Basic Usage

1.1 What is MathJax?

MathJax is an open-source JavaScript display engine for LaTeX and MathML 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. 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, or MathML, a World Wide Web Consortium standard for representing mathematics in XML format. 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.

1.2 Getting Started

MathJax allows you to include mathematics in your web pages, either using TeX and LaTeX notation, or as MathML, and you can even use both in the same document.

There are two ways to access MathJax: the easiest way is to use the copy of MathJax available from our distributed network service at `cdn.mathjax.org`, but you can also download and install a copy of MathJax on your own server, or for use locally on your own hard disk (with no need for network access). Both 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.

1.2.1 Using the MathJax Content Delivery Network (CDN)

To use MathJax from our server, you need to do two things:

1. Link MathJax into the web pages that are to include mathematics.

2. Put mathematics into your web pages so that MathJax can display it.

You accomplish the first step by putting

```
<script type="text/javascript"
  src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</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 and MathML notation, and ask it to generate its output using MathML if the browser supports that, and otherwise use HTML-with-CSS to display the mathematics. This is the most general configuration, and should suffice for most people's needs. Other configurations are available, however, and you can also provide additional configuration parameters to taylor one of the configurations to your needs. More details can be found in the *Loading and Configuring MathJax* instructions.

The use of cdn.mathjax.org is governed by its terms of service, so be sure to read that before linking to the MathJax CDN server.

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

1.2.2 Installing Your Own Copy of MathJax

We recommend using the 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.

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 v1.1 archive from the MathJax download page (you should obtain a file named something like mathjax-MathJax-v1.1-X-XXXXXXXX.zip, where the X's are random-looking letters and numbers). 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 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. Load them in your browser using its web address 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, 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 has changed in MathJax v1.1, so if you have existing pages that use MathJax, 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

```html
<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

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

1.2.3 Putting mathematics in a web page

To put mathematics in your web page, you can use either *TeX* and *LaTeX* notation or *MathML* notation or both 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. These two 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:

```html
<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.

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

```html
<!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="http://cdn.mathjax.org/mathjax/latest/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.

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.

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.

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

```html
<!DOCTYPE html>
<html>
<head>
<title>MathJax MathML Test Page</title>
<script type="text/javascript"
  src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>
</head>
<body>

When <math><mi>a</mi><mo>&#x2260;</mo><mn>0</mn></math>,
there are two solutions to <math>
  <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 mode="display">
  <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>

</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="thinmathspace"></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. 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 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.

1.2.4 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*.

1.3 Installing and Testing MathJax

The easiest way to use MathJax is to link directly to the MathJax distributed network service (see *Using the MathJax 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 and its font package. 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.

1.3.1 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, 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 `v1.1` tagged release is `v1.1-latest`. To get this release, use

```
cd MathJax
git checkout v1.1-latest
```

and to update it when changes occur, use

```
cd MathJax
git pull origin v1.1-latest
```

1.3.2 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://svn.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, 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 either use `git` as described above, or one of the archive files as described below. You can use

```
svn checkout http://svn.github.com/mathjax/MathJax.git@nnn mathjax
```

to check out revision number *nnn*, but it is not easy to tell what svn revision number is associated with a particular release. GitHub's `svn` service doesn't appear to allow you to specify a particular tagged version.

1.3.3 Obtaining MathJax via an archive

Release versions of MathJax are available in archive files from the MathJax download page or the GitHub downloads (click the big download button on the right), where you can download the archive that you need.

You should download the v1.1 archive (which will get you a file with a name like `mathjax-MathJax-v1.1-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 (the download button at the right), 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. You can also get older tagged versions (if there are any).

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 `v1.1-latest`), and then use the download button and the `Download .tar.gz` or `Download .zip` button to get the latest patched version of that release.

1.3.4 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
```

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.

1.3.5 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)$">
<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.

1.3.6 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).

1.3.7 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, like the MathJax 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).

1.4 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:

```html
<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 the MathJax distributed network service, the tag might be

```html
<script type="text/javascript"
   src="http://cdn.mathjax.org/mathjax/latest/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

```html
<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

```html
<script type="text/javascript"
   src="http://cdn.mathjax.org/mathjax/latest/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, 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.

1.4.1 Loading MathJax from the CDN

MathJax is now available as a web service from cdn.mathjax.org, so you can obtain MathJax from there without needing to install it on your own server. The 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.

The CDN hosts the most current version of MathJax, as well as older versions, so you can either link to a version that stays up-to-date as MathJax is improved, or you can stay with one of the release versions so that your pages always use the same version of MathJax.

The URL that you use to obtain MathJax determines the version that you get. The CDN has the following directory structure:

```
mathjax/          # project-name
   1.0-latest/
   1.1-beta/      # temporary
   1.1-latest/    # the 1.1 release with any ciritical patches
   ...
   latest/        # the most current version (1.1-latest in this case)
```

Each directory corresponds to an official MathJax release; however, hotfixes (urgent bug fixes) will be applied in each release branch as necessary, even if new releases are not prepared. In other words, 1.1-latest will initially point to v1.1, but over time may be updated with patches that would correspond to releases that might be numbers 1.1a, 1.1b, etc., even if such releases are not actually prepared for distribution (they likely won't be).

We may occasionally introduce directories for betas, as indicated above, but they will be temporary, and will be removed after the official release.

To load from a particular release, use the directory for that release. For example,

```html
<script type="text/javascript" src="http://cdn.mathjax.org/mathjax/1.1-latest/MathJax.js"></script>
```

will load the stable v1.1 version, even if we release v1.2 or other later versions, while

```html
<script type="text/javascript" src="http://cdn.mathjax.org/mathjax/latest/MathJax.js"></script>
```

will always be the most current stable release, so it will go from v1.1 to v1.2 automatically when that is released. Note that all the versions available on the CDN are stable versions; the development version is not hosted on the CDN. (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.)

The use of `cdn.mathjax.org` is governed by its terms of service, so be sure to read that before linking to the MathJax CDN server.

If you wish to use the MathJax 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 configuration file* section below.

1.4.2 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 earlier versions of MathJax, version 1.1 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.

1.4.3 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, 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, or HTML-CSS output otherwise.

`Accessible.js`
> Essentially the same as `TeX-AMS-MML_HTMLorMML`, but with some settings specified to make MathJax work better with assistive technology (for the visually impaired). This includes setting the zoom trigger to be a double-click, and removing the MathMenu in Internet Explorer (which can interfere with some screen readers).

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), 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="http://cdn.mathjax.org/mathjax/latest/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.

You can also load MathJax from the MathJax CDN server but use a configuration from your own local server:

```
<script type="text/javascript"
    src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS_HTML,http://myserver.com/Math
</script>
```

Because the `local.js` file is not on the 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:

```
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 the CDN, not your local server, and so you need to tell MathJax the actual location of your configuration file.

1.4.4 Using in-line configuration options

The second way to configure MathJax is through *in-line configuration*, that puts the configuration options within the web page itself. This process has changed in version 1.1 to make it compatible with HTML5. Earlier versions of MathJax had in-line configuration included within the content of the `<script>` tag that loads `MathJax.js`, but HTML5 makes it illegal to have content for a script with a `src` attribute.

MathJax solves this problem by using separate `<script>` tags to perform configuration for 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 want, and MathJax will process them in order as 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 de-limiters. 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* sec-tion (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. Note that 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 include the TeX preprocessor, and use an in-line configuration block to enable the single-dollar signs:

```
<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>
```

1.4.5 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. 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.

1.4.6 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 process is the following:

1. Process any configuration file explicitly specified as a script parameter.

2. Process the in-line script body (deprecated), if present.

3. If delayed startup is requested, wait for the indicated signal.

4. Process `text/x-mathjax-config` config blocks.

5. 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 startup must be delayed. 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.

1.5 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.

The remaining files 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 better 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.

1.5.1 The `TeX-AMS-MML_HTMLorMML` configuration file

This configuration file is the most general of the pre-defined configurations. It loads all the important 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.

1.5.2 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.

1.5.3 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, 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.

1.5.4 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. 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"]
  },
  NativeMML: { showMathMenuMSIE: false },
  menuSettings: { zoom: "Double-Click" },
```

```
    errorSettings: { message: ["[Math Error]"] }
});
```

This turns off the MathJax contextual menu for Internet Explorer, since it can interfere with some screen readers. It also sets the zoom trigger to double-click, so that readers can see a larger version of the mathematics but 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.

1.6 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 a configuration object as its value. The 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.

1.6.1 Configuration Options by Component

The individual options are explained in the following sections, which are categorized by the component they affect.

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 number of TeX-specific extensions (see the *TeX and LaTeX input* section of the *Getting Started* document for more details). There is also 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 the only other predefined configuration file.

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 *selector: rules;* styles to be defined dynamically at startup time.

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 regular expressions, 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. 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.

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.

There are also settings for `format`, `renderer`, and `font`, 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.

`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 automatically preceeded by `'(^| )('` and followed by `')( |$)'`, so your pattern will have to match full words in the class name. Assigning an element this class name will prevent *tex2jax* from processing its contents.

`processClass: "tex2jax_process"`

This is the class name used to mark elements whose contents *should* be processed by *tex2jax*. This is used to turn on processing within tags that have been marked as ignored or skipped above. Note that this is a regular expression, and so you need to be sure to quote any *regexp* special characters. The pattern is automatically preceeded by `'(^| )('` and followed by `')( |$)'`, so your pattern will have to match full words in the class name. Use this to restart processing within an element that has been marked as ignored above.

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: "none"
  }
});
```

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

preview: "alttext"

> 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. The default is `"alttext"`, which means use the `<math>` tag's `alttext` attribute as the preview (visible until it is processed by MathJax), if the tag has one. Set it to `"none"` to prevent the previews from being inserted (the math will simply disappear until it is typeset). Set it 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 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. 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.

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. 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 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. 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 en-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.

webFont: "TeX"
> This is the web-based font to use when none of the fonts listed above are available on the user's computer. Note that currently only the *TeX* font is available in a web-based form (they 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.

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 controls whether the MathJax contextual menu will be available on the mathematics in the page. If true, then right-clicking (on the PC) or control-clicking (on the Mac) 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.

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 and 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. 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. Since the *NativeMML* output relies on the browser's native MathML support, MathJax does not control the font size used in the mathematics. You may need to set this value to compensate for the size selected by the browser. The user can also adjust this value using the contextual menu item associated with the typeset mathematics.

showMathMath: true
> This controls whether the MathJax contextual menu will be available on the mathematics in the page. If true, then right-clicking (on the PC) or control-clicking (on the Mac) 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.

showMathMenuMSIE: true
> There is a separate menu setting for MSIE since the code to handle that is a bit delicate; if it turns out to have unexpected consequences, you can turn it off without turning off the menu support in other browsers.

styles: {}
> This is a list of CSS declarations for styling the HTML-CSS 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 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. 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: "MML", 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 does process some MathML natively, its support is not sufficient to handle the more complicated output generated by MathJax, so its setting is `"HTML"` by default.

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 "Settings" submenu of the mathematics contextual menu. It allows the user to change between the *HTML-CSS* and *NativeMML* output processors for the mathematics on the page. Set to `false` to prevent this menu item from showing.

showContext: false
> This controls whether the "Contextual Menu" item will be displayed in the "Settings" submenu of the mathematics 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. (The main reason to allow pass-through to the browser's menu is to gain access to the MathPlayer contextual menu when the NativeMML output processor is used in Internet Explorer with the MathPlayer plugin.) Set to `false` to prevent this menu item from showing.

showFontMenu: false
> This controls whether the "Font Preference" item will be displayed in the "Settings" submenu of the mathematics 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.

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: {
    delay: 600
  }
});
```

would set the `delay` option to 600 milliseconds.

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).

delay: 400
> This the time (in milliseconds) that the mouse must be still over a typeset mathematical formula before the zoomed version is displayed (when the zoom trigger is set to *Hover*).

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 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.

`fadeoutSteps: 10`
> This is the number of steps to take while fading out the FontWarning message. More steps make for a smoother fade-out.

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

1.7 Using MathJax in popular web platforms

MathJax plugins are available for a growing number of wikis, blogs, and other content-management systems. These include WordPress, Blogger, Sphinx, TiddlyWiki, and MathEL-Wiki. A list of these is available in the web applications and integrations list of the MathJax web site.

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

1.7.1 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 one popular 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.

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

```
<script type="text/javascript"
  src="http://cdn.mathjax.org/mathjax/latest/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>`.

The theme files for various popular platforms are:

> **WordPress** `wp-content/themes/[current_theme]/header.php`
>
> **Movable Type** `[current_theme_templates]/html_head.mhtml`
>
> **Drupal** `themes/[current_theme]/page.tpl.php`
>
> **Joomla** `templates/[current_theme]/index.php`
>
> **MediaWiki** `skins/[current_skin].php`
>
> **TiddlyWiki** `*.php` (Whatever you call your TiddlyWiki php file)
>
> **Moodle** `theme/[current_theme]/header.html`

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.

1.7.2 Instructions for Specific Platforms

Some programs, such as WordPress and Moveable Type, allow you to edit template files from inside their administrator interfaces. Specific instructions for these are given via the links below.

Installing MathJax in WordPress

Note that there is a WordPress plugin for MathJax, so that may be a better choice than editing the theme directly.

1. Open the WordPress admin interface.

2. In the administration menu on the left, open up the *Appearance* menu and click on the *Editor* submenu option.
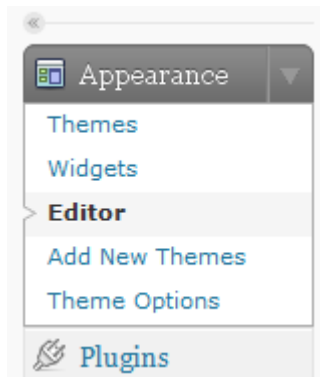
> When you click on the editor option, WordPress should open up the first stylesheet in the current theme.

3. In the template list on the right side of the page, click on the header file (it should be `header.php`).
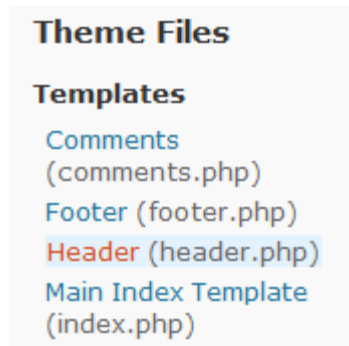
This part depends slightly on how your current theme is written. In the `header.php` file, look for the end-of-head tag, `</head>`. If you find it, insert
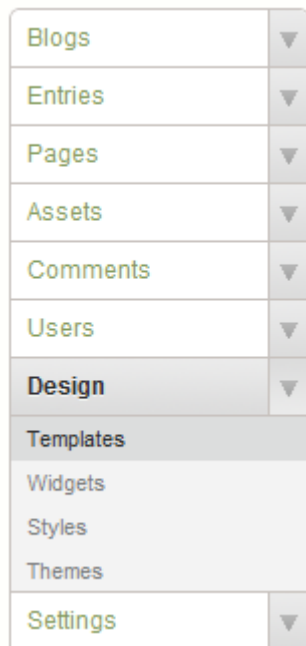
```
<script type="text/javascript"
    src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>
```

just before that. Otherwise, insert the same code at the very bottom of the file.

4. Save the file. This will enable MathJax with both TeX and MathML input, so you should be able to start adding mathematical content to your pages. If you need to adjust the configuration, see *Configuring MathJax* for more details.

Using MathJax in Movable Type

1. Open Moveable Type Admin interface for the site on which you want to enable MathJax.

2. In the dashboard menu on the left, open up the Design menu. This should show you the templates you are currently using on the site.

3. Scroll down to the Template Modules section in the template list and open the *HTML Head* template.
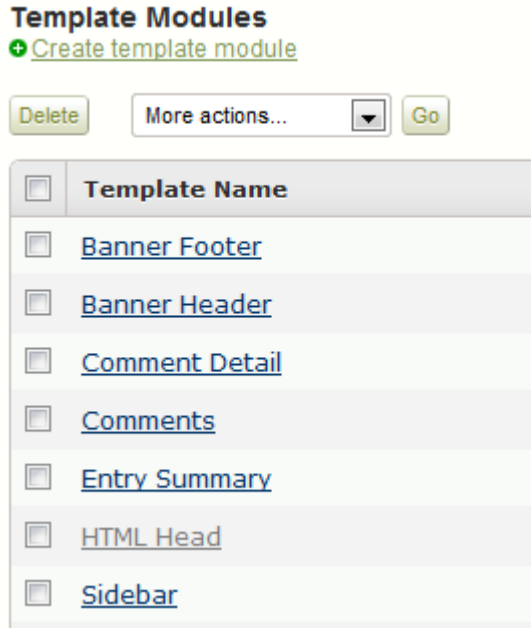
4. At the end of the file, insert

```
<script type="text/javascript"
   src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>
```

to load MathJax from our distributed network service.

5. Save the file. This will enable MathJax with both TeX and MathML input, so you should be able to start adding mathematical content to your pages. If you need to adjust the configuration, see *Configuring MathJax* for more details.

1.8 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.

1.8.1 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.

1.8.2 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 `<`, `>` and `&` to encode these characters so that the browser will not interpret them, but MathJax will. E.g.,

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

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

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

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

1.8.3 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="http://cdn.mathjax.org/mathjax/latest/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.

The main extensions are described below.

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.

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.

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-family": "serif",
      "font-size":   "80%",
      "color":       black",
      "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, as the default is "serif".

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.

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'"
  }
}
```

1.8.4 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.

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
\because	AMSsymbols
\begin	
\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		
\cap		
\Cap	AMSsymbols	
\cases		
\cdot		
\cdotp		
\cdots		
\centerdot	AMSsymbols	
\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		
\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
\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
\end
\enspace
\epsilon
\eqalign
\eqalignno
\eqcirc   AMSsymbols
\eqsim   AMSsymbols
\eqslantgtr   AMSsymbols
\eqslantless   AMSsymbols
\equiv
\eta
\eth   AMSsymbols
\exists
\exp

F

\fallingdotseq   AMSsymbols
\fbox
\Finv   AMSsymbols
\flat
\forall
\frac
\frac   AMSmath
\frak
\frown

G

\Game   AMSsymbols
\Gamma
\gamma
\gcd
\ge
\genfrac   AMSmath
\geq
\geqq   AMSsymbols
\geqslant   AMSsymbols

\gets	
\gg	
\ggg	AMSsymbols
\gggtr	AMSsymbols
\gimel	AMSsymbols
\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

\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
\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
\mathtt
\matrix
\max
\mbox
\measuredangle   AMSsymbols
\mho   AMSsymbols
\mid
\min
\mit
\mkern
\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]
\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
\renewcommand   [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
\tfrac   AMSmath
\therefore   AMSsymbols
\Theta
\theta

\thickapprox	AMSsymbols	
\thicksim	AMSsymbols	
\thinspace		
\tilde		
\times		
\tiny		
\Tiny		non-standard
\to		
\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`	
`\xleftarrow`	AMSmath
`\xrightarrow`	AMSmath

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

eqnarray
eqnarray*
equation
equation*

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

matrix
multline        [AMSmath]
multline*       [AMSmath]

pmatrix

smallmatrix      AMSmath
split           [AMSmath]
subarray         AMSmath

Vmatrix
vmatrix
```

1.9 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, 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.

1.9.1 MathML in HTML pages

For MathML that is handled via the preprocessor, you should not use the named MathML entities, but rather use the numeric entities like `√` 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 woth 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 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.

1.9.2 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`.

- Limited support for line breaking (they are only allowed in direct children of `mrow` or implied `mrow` elements).

- No support for alignment groups in table.

- No support for right-to-left rendering.

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

1.10 MathJax Output Formats

Currently, MathJax can render math in two ways:

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

- Using a browser's native MathML support.

These are implemented by the *HTML-CSS* 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 `_HTMLorMML`, then the NativeMML output processor will be chosen if the browser supports it, 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"` 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. (The HTML-CSS processor has not yet been optimized for speed, so you can expect some improvement in the future. Note that IE8 in "IE8 standards mode" is an order of magnitude slower than any other browser when processing math through the HTML-CSS output processor; see *HTML-CSS with IE8* below for some strategies to deal with this.)

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, Chrome, Konqueror, and most other browsers don't support MathML natively, but may 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 lay out 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 may rely on features that are not available in some renderers (for example, Firefox's MathML support does not implement some of the named widths, such as `negativethinmathspace`). The results using the NativeMML output processor may have spacing or other rendering problems that are outside of MathJax's control.

1.10.1 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, 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"]
```

You can customize which choice to make 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 HTML-CSS output rather than native MathML support for Firefox:

```html
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    MMLorHTML: { prefer: { Firefox: "HTML" } }
  });
</script>
<script type="text/javascript"
  src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>
```

With this configuration, MathML output will be used only for IE with the MathPlayer plugin (Firefox is the only other browser to have native MathML support that is sufficient for use with MathJax). 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.

1.10.2 HTML-CSS with IE8

Internet Explorer 8 has at least eight different rendering modes in which it can operate, and that are triggered by the *DOCTYPE* of the document being viewed. Its "quirks" mode is its fastest mode, and its "IE8 standards" mode is its slowest. This is the mode triggered by strict HTML document types, and since most modern content management systems now include a *DOCTYPE* that activates "standards" mode, IE8 will operate in its slowest manner. This is particularly apparent when MathJax is used, since IE8 in standards mode runs 20 to 30 times slower than it does in its IE7 emulation mode, and 60 times slower than in quirks mode, on the sample equations page in `test/sample.html`.

Most users find this speed reduction unacceptable when there is much mathematics on the page. To overcome this problem, you may wish to tell IE8 to use its IE7 emulation mode rather than its IE8 standards mode. You can accomplish this by including the line

```html
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7">
```

at the top of the `<head>` section of your HTML documents. This lets you keep the strict *DOCTYPE* for validation purposes, while still managing to get reasonable performance from Internet Explorer 8. Note that this line must come at the beginning of the `<head>`, before any stylesheets or other content are loaded.

Alternatively, you can use the *MMLorHTML* configuration file described above to select NativeMML output when possible, and request that your users install the MathPlayer plugin, which will render the mathematics much more quickly.

It appears that IE9 in IE9 standards mode may perform better than IE8, but since IE9 is still in beta testing as of this writing, we have yet to see exactly what the performance of MathJax in IE9 will be like.

1.11 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.

1.11.1 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.

1.11.2 Issue tracking

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

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

1.11.3 "Powered by MathJax"

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

Upgrading MathJax

2.1 What's New in MathJax v1.1

MathJax version 1.1 includes a number of important improvements and enhancements over version 1.0. We have worked hard to fix bugs, improve support for browsers and mobile devices, process TeX and MathML better, and increase MathJax's performance.

In addition to these changes, MathJax.org now offers MathJax as a network service. Instead of having to install MathJax on your own server, you can link to our content delivery network (CDN) to get fast access to up-to-date and past versions of MathJax. See *Loading MathJax from the CDN* for more details.

The following sections outline the changes in v1.1:

2.1.1 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 downlaoded, you don't have to wait so long.

- Rollover to image fonts if a web font fails to load (so you don't have to wait for *every* font to fail.

- The MathJax files are now packed only with *yuicompressor* rather than a custom compressor. The CDN serves gzipped versions, which end up being smaller than the gzipped custom-packed files.

- Improved rendering speed in IE by removing `position:relative` from the style for mathematics.

- Improved rendering speed for most browsers by isolating the mathematics from the page during typesetting (avoids full page reflows).

2.1.2 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).

2.1.3 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.

2.1.4 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.

2.1.5 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.

2.1.6 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.

2.2 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.

2.2.1 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.

2.2.2 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`.

2.2.3 Loading MathJax from the 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 the MathJax CDN service, simply load MathJax as follows:

```
<script type="text/javascript"
  src="http://cdn.mathjax.org/mathjax/latest/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 the MathJax CDN server.

2.2.4 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

```html
<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

```html
<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

```html
<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.

2.2.5 New Distribution Location

Version 1.0 of MathJax was distributed through *SourceForge*, but the development of MathJax has switched to GitHub, which is now the primary location for MathJax source code and distributions. The SourceForge repository will no longer be actively maintained (and hasn't been since November 2010), and so you will not be able to obtain updates through `svn` if you checked out MathJax from there.

You may be able to switch to using the MathJax CDN (see above) rather than hosting your own copy of MathJax, and avoid the problem of updates all together. If you must install your own copy, however, you should follow the instructions at *Installing and Testing MathJax*, using either `git` or `svn` as described to obtain your copy from GitHub. This will allow you to keep your copy of MathJax up to date as development continues.

We apologize for the inconvenience of having to switch distributions, but the git-to-svn bridge we tried to implement to keep both copies in synch turned out to be unreliable, and so the SourceForge distribution was retired in favor of the GitHub site.

2.3 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 Math-Jax'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="http://cdn.mathjax.org/mathjax/latest/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.

Advanced Topics

3.1 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* or *LaTeX*), 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*, 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, and 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).

3.1.1 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, and *math/mml* for mathematics in MathML notation. When the *tex2jax* or *mml2jax* 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.

3.1.2 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. Output jax could be produced that render the mathematics using SVG, 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.

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.

3.2 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.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 set 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
    - 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. 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

6. Load the extension files:

   - Post the `Begin Extension` startup signal

   - Load the files from the `MathJax.Hub.config.extensions` array

     - Most extensions will post a `Extension [name] Ready` startup message when they are loaded (where `[name]` is the name of the extension)

   - Post the `End Extension` startup signal

7. Set the MathJax menu's renderer value based on the jax that have been loaded

8. Wait for the onload handler to fire

9. Set `MathJax.isReady` to `true`

10. 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

      - 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

11. Post the `End` startup signal

The loading of the jax and extensions in steps 5 and 6 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 5 and 6 must complete, however, before 7 will be performed.

See the `test/sample-signals.html` file to see the signals in action.

3.3 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 used 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:

3.3.1 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, Internet Explorer does 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.

3.3.2 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>
```

3.3.3 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, 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 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.

3.4 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` 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 so MathJax will not be able to tell if it is safe for it to process the contents of the page. Indeed, it will wait forever for its `onload` handler to fire, and so will never process the page.

To solve this problem, you will need to call MathJax's `onload` handler yourself, to let it know that it is OK to typeset the mathematics on the page. You accomplish this by calling the `MathJax.Hub.Startup.onload()` method as part of your MathJax startup script. To do this, you will need to give MathJax an in-line configuration.

Here is an example of how to load and configure MathJax dynamically:

```
(function () {
  var script = document.createElement("script");
  script.type = "text/javascript";
  script.src  = "http://cdn.mathjax.org/mathjax/latest/MathJax.js";

  var config = 'MathJax.Hub.Config({' +
                 'extensions: ["tex2jax.js"],' +
                 'jax: ["input/TeX","output/HTML-CSS"]' +
               '});' +
               'MathJax.Hub.Startup.onload();';

  if (window.opera) {script.innerHTML = config}
            else {script.text = config}

  document.getElementsByTagName("head")[0].appendChild(script);
})();
```

You can adjust the `config` variable to your needs, but be careful to get the commas right. The `window.opera` test is because some versions of Opera don't handle setting `script.text` properly, while Internet Explorer doesn't handle setting the `innerHTML` of a script tag.

Here is a version that uses the `config=filename` method to configure MathJax:

```
(function () {
  var script = document.createElement("script");
```

```
    script.type = "text/javascript";
    script.src  = "http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML-full";

    var config = 'MathJax.Hub.Startup.onload();';

    if (window.opera) {script.innerHTML = config}
               else {script.text = config}

    document.getElementsByTagName("head")[0].appendChild(script);
})();
```

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.

3.4.1 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. The MathJax 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.src = "http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML-full";
    var config = 'MathJax.Hub.Startup.onload()';
    if (window.opera) {script.innerHTML = config} else {script.text = config}
    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.src = "http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML-full"
    var config = 'MathJax.Hub.Startup.onload()';
    if (window.opera) {script.innerHTML = config} else {script.text = config}
    document.getElementsByTagName("head")[0].appendChild(script);
  }
}
```

Source: mathjax_wikipedia.user.js

3.5 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.

3.5.1 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 dollar signs (it will be blank). 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.
>
> **`Reprocess()`**
> > to remove the output and reproduce it again (for example, if CSS has changed that would alter the spacing of the mathematics).
>
> **`Remove()`**
> > to remove the output for this math element (but not the original `<script>` tag).
>
> **`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");
});
```

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 (available in a more full-featured version as `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="http://cdn.mathjax.org/mathjax/latest/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>
```

3.6 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 importat, 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.

3.6.1 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 most of 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.Extensions
> Initially empty, this is where extensions can load their code. For example, the *tex2jax* preprocessor creates `MathJax.Extensions.tex2jax` for its code and variables.

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.

MathJax.isReady
> This is set to `true` when MathJax is set up and ready to perform typesetting actions (and is `null` otherwise).

3.6.2 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.

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.

   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
     }
   );
   ```

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. 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)

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`

3.6.3 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

3.6.4 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.Messssage.Clear()` method by hand to remove it.

> **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

3.6.5 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

setText (*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`

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"}`.

3.6.6 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.

..describe:: ["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,

> ```
> ["length",[1,2,3,4]]
> ```

> would call the *length* method on the array [1,2,3,4] when the callback is called, returning 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

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`

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

3.6.7 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.

3.6.8 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

3.6.9 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

Translate(*script*)
> 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 redefined, 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
> >
> > **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`

3.6.10 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"`);

Methods

Translate(*script*)
> 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 redefined, 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.
>
> > **Parameters**
> >
> > > • **script** — the `<script>` element to be translated
> >
> > **Returns** the *element jax* resulting from the translation

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`

3.6.11 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

> A reference to the input jax that created the element. (In the future, this will be changed to the name of the input jax.)

outputJax

> A reference to the output jax that has processed this element. (In the future, this will be changed to the name of the output jax.)

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

Reprocess ([*callback*])
> Removes the output and produces it again. This may be necessary if there are changes to the CSS styles that would affect the layout of the mathematics, for example. 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

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.

3.6.12 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.

Methods

Process(*script*)
> This is the method that the `MathJax.Hub` calls when it needs the input or output jax to process the given math `<script>`. 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 `Translate()` method, 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
> >
> > **Returns** an *ElementJax* object, or `null`

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. 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.

3.6.13 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"
```

Reference Pages

4.1 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>
```

4.2 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 .merror": {
    "background-color": "#FFFF88",
    color:   "#CC0000",
    border:  "1px solid #CC0000",
    padding: "1px 3px",
    "font-family": "serif",
    "font-style": "normal",
    "font-size":  "90%"
  },

  ".MathJax_Preview": {color: "#888888"},

}
```

This defines two CSS styles, one for the selector `.MathJax .merror`, which specifies a background color, foreground color, border, and so on, and a second for `.MathJax_Preview` that sets its color.

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.

4.3 Glossary

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

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

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*

- User Help Pages:
    - **MathJax Font Help**
    - **MathJax Contextual Menu**
    - **MathJax Zoom Feature**

This version of the documentation was built May 01, 2014.