Orion/How Tos/Editor Build

From Eclipsepedia

< Orion | How Tos

Contents

Editor Builds

- In order to make the Orion editor easier to consume, 2 new nightly builds have been introduced.
- One build is targeted at users who already use the RequireJS (http://requirejs.org/) module loader.
- The other build is targeted at users who want to embed the editor without any other dependencies.
- The builds are identical in functionality and are available both minified and non-minified.

Getting the Build

- Users can grab the most recent editor code from the nightly builds or they can use the latest release available on the Orion editor web page.
- Users need to download (or directly link to) one type of build (either the RequireJS or standalone) and the common CSS file.

Nightly builds & Releases

- From the Orion build page (http://download.eclipse.org/orion) , users can click on a nightly build or a release (starting with 2.0) and they will be presented with a page that now includes the built editor components:

Orion Client Components

File	Download	Description
built-editor-amd.min.js	(http)	Orion Editor (AMD minified)
built-editor-amd.js	(http)	Orion Editor (AMD non-minified)
built-editor.min.js	(http)	Orion Editor (minified)
built-editor.js	(http)	Orion Editor (non-minified)
built-editor.css	(http)	Orion Editor CSS (common)

- **build-editor-amd.min.js** and **built-editor-amd.js** are the minified and non-minified versions of the RequireJS build.
- **built-editor.min.js** and **built-editor.js** are the minified and non-minified versions of the standalone build.
- **built-editor.css** is the built css file and is needed by both builds.

Orion Editor web page

- The built editor files will also be available directly from the Orion editor web page releases section (http://eclipse.org/orion/editor/releases) .

Using the Build

The next sections will show the simplest way to embed an editor by using the orion/editor/edit function. Note if you want customize your editor (beyond the options provided by the edit function), you have access to the full editor in the build.

The following demos are going to start with the following html file then add in the built editor:

```
<!doctype html>
<html>
<head>
<title>Orion Editor Sample</title>
</head>
<body>
Here is some Javascript code:

<pre>
/*
 * This is an Orion editor sample.
 */
function() {
    var a = 'hi there!';
    window.console.log(a);
}
</pre>
</body>
</html>
```

Using the standalone build

- These instructions are going to use the builds available on eclipse.org/orion.editor. If you have downloaded the **built-editor.min.js** and **built-editor.css** files to your local server, just update the corresponding paths.
- Add in the css file:

```
<!doctype html>
<html>
<head>
<title>Orion Editor Sample</title>
<link rel="stylesheet" type="text/css" href="http://eclipse.org/orion/editor/releases/2.0/built-editor.c
</head>
<body>

... snip ...
```

- Add in the editor file. Note that you can change **className** to be whatever you want (it just has to match up the classname you use in the body for the editor):

```
<!doctype html>
<html>
<head>
<title>Orion Editor Sample</title>
<link rel="stylesheet" type="text/css" href="http://eclipse.org/orion/editor/releases/2.0/built-editor.c
<script src="http://eclipse.org/orion/editor/releases/2.0/built-editor.min.js"></script>
<script>
        /*global require*/
        require(["orion/editor/edit"], function(edit) {
                edit({className: "editor"});
        });
</script>
</head>
<body>

... snip ...
```

- Add the **className** attribute to the code. Note you can also pass in other attributes that influence the editor configuration (see Attributes):

```
... snip ...
<body>
Here is some Javascript code:

<pre class="editor" data-editor-lang="js">
/*
 * This is an Orion editor sample.
 */
function() {
    var a = 'hi there!';
    window.console.log(a);
}
</pre>
</body>
</html>
```

- Your final file should look like this:

```
<!doctype html>
<html>
<head>
<title>Orion Editor Sample</title>
<link rel="stylesheet" type="text/css" href="http://eclipse.org/orion/editor/releases/2.0/built-editor.c
<script src="http://eclipse.org/orion/editor/releases/2.0/built-editor.min.js"></script>
<script>
        /*global require*/
        require(["orion/editor/edit"], function(edit) {
                edit({className: "editor"});
        });
</script>
</head>
<body>
Here is some Javascript code:

<pre class="editor" data-editor-lang="js">
/*
 * This is an Orion editor sample.
 */
function() {
    var a = 'hi there!';
    window.console.log(a);
}
</pre>
</body>
</html>
```

Using the RequireJS build

- These instructions are going to use the builds available on eclipse.org/orion.editor. If you have downloaded the **built-editor-amd.min.js** and **built-editor.css** files to your local server, just update the corresponding paths.
- Add in the css file:

```
<!doctype html>
<html>
<head>
<title>Orion Editor Sample</title>
<link rel="stylesheet" type="text/css" href="http://eclipse.org/orion/editor/releases/2.0/built-editor.c
</head>
<body>

... snip ...
```

- Add in the RequireJS file. You can use a local one or link to one.

<script src="http://requirejs.org/docs/release/2.1.4/minified/require.js"></script>

- Add in the editor file. Note that you can change **className** to be whatever you want (it just has to match up the classname you use in the body for the editor):

```
<!doctype html>
<html>
<head>
<title>Orion Editor Sample</title>
<link rel="stylesheet" type="text/css" href="http://eclipse.org/orion/editor/releases/2.0/built-editor.c
<script src="http://requirejs.org/docs/release/2.1.4/minified/require.js"></script>
<script>
        require(["http://eclipse.org/orion/editor/releases/2.0/built-editor-amd.min.js"], function(edit)
                edit({className: "editor"});
        });
</script>
</head>
<body>

... snip ...
```

- Add the **className** attribute to the code. Note you can also pass in other options that influence the editor configuration (see : editor options):

```
... snip ...
<body>
Here is some Javascript code:

<pre class="editor" data-editor-lang="js">
/*
 * This is an Orion editor sample.
 */
function() {
    var a = 'hi there!';
    window.console.log(a);
}
</pre>
</body>
</html>
```

- Your final file should look like this:

```
<!doctype html>
<html>
<head>
<title>Orion Editor Sample</title>
<link rel="stylesheet" type="text/css" href="http://eclipse.org/orion/editor/releases/2.0/built-editor.c
<script src="http://requirejs.org/docs/release/2.1.4/minified/require.js"></script>
<script>
        require(["http://eclipse.org/orion/editor/releases/2.0/built-editor-amd.min.js"], function(edit)
                edit({className: "editor"});
        });
</script>
</head>
<body>
Here is some Javascript code:

<pre class="editor" data-editor-lang="js">
/*
 * This is an Orion editor sample.
 */
function() {
    var a = 'hi there!';
    window.console.log(a);
}
</pre>
```

```
</body>
</html>
```

Editor Options

Below is the current list of options(along with their default values) supported in the 2.0 release. The options can be specified directly in HTML by adding data attributes prefixed with "data-editor". Since HTML attributes are case insensitive, camel case letters are prefixed by a dash. For example: fullSelection becomes full-selection.

For example:

```
<pre class="editor" data-editor-full-selection="false" data-editor-readonly="true" data-editor-lang="js"
```

Note that these may change in future releases.

```
/**
* @class This object describes the options for <code>edit</code>.
* @name orion.editor.EditOptions
*
* @property {String|DOMElement} parent the parent element for the view, it can be either a DOM element o
* @property {Boolean} [readonly=false] whether or not the view is read-only.
* @property {Boolean} [fullSelection=true] whether or not the view is in full selection mode.
* @property {Boolean} [tabMode=true] whether or not the tab keypress is consumed by the view or is used
* @property {Boolean} [expandTab=false] whether or not the tab key inserts white spaces.
* @property {String} [themeClass] the CSS class for the view theming.
* @property {Number} [tabSize=4] The number of spaces in a tab.
* @property {Boolean} [wrapMode=false] whether or not the view wraps lines.
* @property {Function} [statusReporter] a status reporter.
* @property {String} [title=""] the editor title.
* @property {String} [contents=""] the editor contents.
* @property {String} [lang] the styler language. Plain text by default.
* @property {Boolean} [showLinesRuler=true] whether or not the lines ruler is shown.
* @property {Boolean} [showAnnotationRuler=true] whether or not the annotation ruler is shown.
* @property {Boolean} [showOverviewRuler=true] whether or not the overview ruler is shown.
* @property {Boolean} [showFoldingRuler=true] whether or not the folding ruler is shown.
*/
```


This page was last modified 20:24, 19 February 2013 by Bogdan Gheorghe.