Contents

[Introduction 2](#_Toc370303827)

[Editor Style Sheet 2](#_Toc370303828)

[Specifying the Style Sheet within iCM 2](#_Toc370303829)

[Defining Styles 3](#_Toc370303830)

[Defining Table Auto-Formatting Styles 3](#_Toc370303831)

[Styling Notes 5](#_Toc370303832)

[Editor Configuration File 5](#_Toc370303833)

[Customizing the Editor’s Toolbar 6](#_Toc370303834)

[Configuring the Editor for HTML Compliance 7](#_Toc370303835)

[Editor Content (Style) Rules 8](#_Toc370303836)

[Defining Content Rules 8](#_Toc370303837)

[Defining Content Rules for Table Auto-Formatting 9](#_Toc370303838)

[Generating Content Rules using an existing (Pre iCM Version 10) Editor Style Sheet 10](#_Toc370303839)

[Browser Settings 11](#_Toc370303840)

[Testing/Troubleshooting 11](#_Toc370303841)

[Appendix 1: Editor Configuration Diagram 13](#_Toc370303842)

# Introduction

The iCM HTML Text Editor relies on the presence of certain files within the iCM installation and the associated site(s) to function correctly. The visual appearance of content displayed within the editor is dictated by the ***Editor Style Sheet***. The functionality and behaviour of the editor is governed by a combination of the ***Editor Configuration File*** settings and the ***Editor Content Rules***. These settings and rules are also used to determine the html elements and attributes which are supported within article content, whether submitted within iCM via the editor, or externally via one of the APIs.

The iCM HTML Text Editor may also require the browser’s security settings to be configured to allow certain functionality. This will be covered in the ***Browser Settings*** section of this document.

Finally, some advice on troubleshooting the editor’s configuration is provided within the ***Troubleshooting*** section.

# Editor Style Sheet

The ***Editor Style Sheet*** is a standard CSS file that is loaded into the editor to style its content in the same way it will be styled on the site. It is typically a simplified version of the site style sheet.

The editor style sheet should also include the class definitions referenced by the content rules used to populate the **Style***s* drop-down within the editor’s toolbar, and by the rules used to provide Table Auto Format Styling. Refer to the [***Editor Content Rules***](#_Editor_Content_(Style)) section for further details.

### Specifying the Style Sheet within iCM

The style sheet is located within the site directory structure and can be provided on a global, site and/or template level. It is specified at a global level via the *HTML Editor Settings* within the AutoConfig utility, or via the *Content Settings* of the Management | System Configuration | iCM Settings/Configuration section of iCM itself. The style sheet provided at a global level will be applied to all articles unless subsite and/or template override settings exist.

The global style sheet setting may be overridden for a particular site via the **Settings |Display** tab within the **Management | Definitions | Subsites** section of iCM. A site's style sheet setting may, in turn, be overridden on a template by template basis within the **Management | Definitions |Templates** section of iCM. Display settings provided via a Template's **Settings | Display** tab will apply to all sites that use the template unless overridden for a particular site on the Template's **Subsite Settings** tab. Display settings provided for a site via the **Subsite Settings** tab within a template will apply to a particular site only.

If no display settings are specified anywhere, a style sheet of defaultarticletext.css is assumed; a sample defaultarticletext.css style sheet exists in the sample site included as part of the iCM installation files.

All style sheets should be specified as a path relative to the site’s root directory. For example, css/ecitizen/editor.css.

🖎 It is advisable to use forward-slashes (/) to separate the path elements, even on Windows-based servers, in order to maximise the portability of the settings; template settings in particular can be exported from one iCM to another.

### Defining Styles

The editor’s style sheet is just a standard CSS file and as such should follow standard CSS guidelines. It must, however, provide definitions for any styles that are referenced by rules within the content rules file. For example, if a content rule includes a class element with a value of “redtext” the editor style sheet must include a style definition for the class “redtext”. Refer to the [***Editor Content Rules***](#_Editor_Content_(Style)) section for further details.

🖎 Older (pre-version 10) style sheets can be re-used, without modification, for iCM Version 10. The '$name' or ‘$type’ comment blocks used previously to indicate the styles to be made available within the editor have now been superseded by the content rules; if present within the editor style sheet these comment blocks will now simply be ignored.

### Defining Table Auto-Formatting Styles

The editor’s style sheet must also cover any table auto-formatting styles that are to be made available via the **Auto Format Table** context menu option for a table within the editor. Table auto-formatting styles allow a complete “look and feel” to be applied to a table. The following class names are applied to a table and its cells when the auto-formatting option is applied within the editor:

|  |  |  |
| --- | --- | --- |
| **Class name assigned** | **Element(s) assigned to** | **Required** |
| <prefix>general | TABLE | Yes |
| <prefix>topleft | TH, TD elements in the first column of the top row | Yes |
| <prefix>topcenter | TH, TD elements in the top row that do not fall within the first or last column | Yes |
| <prefix>topright | TH, TD elements in the last column of the top row | Yes |
| <prefix>middleleft | TH, TD elements in the first column that do not fall within the top or bottom rows | Yes |
| <prefix>middlecenter | TH, TD elements that do not fall in the first or last columns or the top or bottom rows | Yes |
| <prefix>middleright | TH, TD elements in the last column that do not fall within the first or last rows | Yes |
| <prefix>bottomleft | TH, TD elements in the first column of the bottom row | Yes |
| <prefix>bottomcenter | TH, TD elements in the bottom row that do not fall within the first or last column | Yes |
| <prefix>bottomright | TH, TD elements in the last column of the bottom row | Yes |
| <prefix>odd | TR element of each odd row | No |
| <prefix>even | TR element of each even row | No |

Where <prefix> is a suitable alpha-numeric that identifies a particular auto-formatting style.

The editor style sheet should, therefore, provide suitable style definitions for the applied classes. An example set of style definitions for table auto-formatting would look something like the following, assuming a <prefix> of ‘ts1’:

table.ts1general {

width: 100%;

border-collapse: separate;

border-spacing: 5px;

}

table.ts1general td, table.ts1general th {

padding: 5px;

}

.ts1general {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

font-weight: normal;

color: #000000;

}

.ts1topleft {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

background-color: #660099; color: #ffffff;

font-weight: bold;

padding: 5px;

}

.ts1topcenter {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

background-color: #660099; color: #ffffff;

font-weight: bold;

padding: 5px;

}

.ts1topright {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

background-color: #660099; color: #ffffff;

font-weight: bold;

padding: 5px;

}

.ts1middleleft {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

font-weight: normal;

}

.ts1middlecenter {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

font-weight: normal;

}

.ts1middleright {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

font-weight: normal;

}

.ts1bottomleft {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

font-weight: normal;

}

.ts1bottomcenter {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

font-weight: normal;

}

.ts1bottomright {

font-family : Verdana, Arial, Helvetica, Geneva, sans-serif;

font-weight: normal;

}

.ts1odd {

background-color: #eeeeee;

}

.ts1even {

background-color: #eeeeee;

}

Suitable content rules that link these style definitions (by their name) to the relevant elements (table, tr, th, td) will then need to be defined. Without these, the classes applied by the editor will get filtered out whenever the article content is cleaned up i.e. on load into the editor and on submit. Additionally, the **Class** drop-downs on the table/cell property dialogs would fail to list these styles. Refer to the [***Editor Content Rules***](#_Editor_Content_(Style)) section for further details about content rules.

🖎 The table style definitions provided in versions prior to iCM 10 should not need any work. Content rules that reference them, however, will need defining. Old style $type=’table’ comment blocks will be ignored. Refer to the [***Generating Content Rules using an existing Editor Style Sheet***](#_Generating_Content_Rules) section for help on generating content rules from a pre-existing style sheet.

### Styling Notes

The following notes may be useful when working on a style sheet for the editor.

1. If applying styles to the **body** element, be aware that this may affect the margins around the content when displayed within the editor. For example, including the following within the editor style sheet will result in the text butting up against the editor’s frame and not looking particularly pretty:   
   *body {margin: 0; padding: 0;}*
2. Certain elements generated by the editor may need styling to make them easily identifiable within the editor (and on the site). At the time of writing, the **acronym** and **abbr** elements, for example, do not seem to have any obvious styling in one or more of the supported browsers.
3. If the editor has been configured to support tables, be aware that the content of table cells (**th**/**td** elements), may include **p** tags. As soon as the ENTER key has been pressed inside a table cell within the editor, **p** elements will be present.
4. Also relating to tables, and a change from the previous HTML Text Editor, table rows will now be wrapped with **tbody**/**thead** elements.

🖎 Do not include other style sheets via the **@import** command; all CSS required by the editor should be placed directly within the file.

# Editor Configuration File

The Editor Configuration File, dhtmledit\_config.js located within the icm/custom directory, helps determine the functionality and behaviour of the editor in combination with the content rules; the latter are discussed in the [***Editor Content Rules***](#_Editor_Content_(Style)) section. The configuration file determines the various tools made available for editing content within the editor, and also helps configure the HTML elements/attributes that are supported.

The configuration file format is JSON plus /\* \*/ comment blocks. It should contain a JSON object that in turn contains zero of more settings. JSONLint (<http://jsonlint.com>) or similar may be used to verify that the JSON (minus any comments) is valid.

The comment block included within the configuration file included within the iCM installation describes all of the available settings, indicating their default values, and providing examples.

🖎 Configuration file settings apply globally and so will apply to all article text editing/submissions, regardless of the site or user being used to access iCM.

### Customizing the Editor’s Toolbar

The toolbar items that are provided for editing content within the editor are determined by the **toolbar\_Full** setting. By default, the full set of toolbar items are made available.

Overriding this setting enables functionality within the article editor to be restricted. If this toolbar definition does not include a particular item or set of items, the associated editing functionality will be omitted from the editor and the relevant HTML elements/attributes will be filtered out of submitted content.

🖎 Care should be taken to ensure that the “lists”, “links”, inlines”, “tables” and “styles” toolbar button groups are always named as such. iCM allows some editing functionality to be restricted on a per user basis (via user privileges) and so may omit any one of these complete groups, by name.

An example of a simple, cut down toolbar configuration would be:

"toolbar\_Full" : [

{"name":"clipboard", "items": [ "Cut","Copy","Paste","-","Undo","Redo" ]},

{"name":"editing", "items": [ "Find","Replace","-","SelectAll" ] },

{"name":"basicstyles","items": [ "Bold","Italic ] },

{"name":"styles", "items": [ "Styles" ] },

{"name":"tools", "items": [ "RemoveFormat","ShowBlocks","Maximize" ] },

{"name":"about", "items": [ "iCMAbout" ] }

]

This would only provide support for simple styling operations, including bold, italic and whatever the content (style) rules specify as styling options for the *Styles* drop-down; see the [***Editor Content Rules***](#_Editor_Content_(Style)) section for further details about the content rules. Any additional editor functionality such as tables, lists, links, etc would be unsupported and the associated HTML elements/attributes would be filtered out of any submitted content.

🖎 Be aware that if the editor configuration has been made more restrictive, existing content will be affected the next time the article is loaded into the editor and/or submitted.

🖎 If upgrading an existing iCM installation, the toolbar prior to iCM Version 10 did not typically get configured, although it was possible. It would be worth double-checking the FCKConfig.ToolbarSets["iCM"] setting specified within the old dhtmledit\_config.js file (either within the icm/admin/articles section or the icm/custom directory).

### Configuring the Editor for HTML Compliance

By default, the HTML Text Editor generates HTML 4/XHTML 1.1 compliant content. The configuration file supports customisation of certain editor dialog items so that HTML 5 compliant content can be generated. The name of each setting that can be used to customize a dialog item takes the following format: *<dialog id>\_<tab id>\_<field id>\_<suffix>.*

🖎 If you are unsure whether a particular setting is going to customize the desired dialog item, adding the **Diagnostics** privilege to the iCM user’s account will show the relevant *<dialog id>\_<tab id>\_<field id>* identifier as you hover over fields within any of the editor’s dialogs.

The following suffixes are used for these settings, and are used to identify the type of customization to be performed:

**\_show**Settings that have *\_show* as their suffix can be used to either show or hide the associated dialog item. For example, at the time of writing, the "summary" attribute is deprecated in HTML5; the “caption” attribute should be used instead. To configure the editor so that it no longer supports the “summary” attribute, simply set the *table\_info\_txtSummary\_show* *setting* to false:

"table\_info\_txtSummary\_show" : false

🖎 Be aware that changing a *\_show* setting from true to false will result in the associated HTML attribute being filtered out of existing article content the next time it is loaded into the article editor and/or it is submitted.

**\_default**Settings that have *\_default* as their suffix can be used to change the initial default value of the associated dialog item. For example, on the table properties dialog the cell padding field defaults to 1. If a different cell padding is required, set the *table\_info\_txtCellPad\_default* setting:

"table\_info\_txtCellPad\_default" : 0

**\_regex**Settings that have *\_regex* as their suffix can be used to change the validation performed on data entered into the associated dialog item. For example, HTML 5 supports a different set of link types (relationships) to those supported in HTML 4. To configure the editor to validate against the HTML 5 types, use the *link\_advanced\_advRel\_regex* setting to override the HTML 4 defaults:

"link\_advanced\_advRel\_regex" : {

"re":"^(alternate|author|bookmark|external|help|icon|license|next|nofollow,noreferrer|pingback|prefetch|prev|search|stylesheet|tag)?$",

"msg": "Please specify one of the following relationships (link types): alternate, author, bookmark, external, help, icon, license, next, nofollow, noreferrer, pingback, prefetch, prev, search, stylesheet, tag"

}

\_**required**Settings that have *\_required* as their suffix can be used to make the associated dialog item either required or optional. For example, the default configuration requires that a table’s summary is specified. To make the table summary optional, set the table\_info\_txtSummary\_required setting to false:

"table\_info\_txtSummary\_required" : false

Examples of configuring the editor for HTML 4 and for HTML 5 are included within the icm/custom/EXAMPLES directory of the iCM installation, namely dhtmledit\_config\_html4.js and dhtmledit\_config\_html5.js.

🖎 The compliance level that used to be specified via AutoConfig, prior to iCM version 10, is no longer supported. The default configuration in the editor used by v10 (or later) is roughly equivalent to the old STRICT/HIGH compliance levels. The defaults differ from the old STRICT compliance level in that there are a few extra table options (e.g. table cell ‘nowrap’) that are now supported due to them no longer using deprecated attributes. The defaults differ from the old HIGH compliance level in that the link *target* attribute is automatically mapped to *rel=”external”*; set the *mapLinkTarget* setting to false and the *link\_target\_tab\_show* setting to true to configure the editor to mimic the old HIGH level behaviour.

# Editor Content (Style) Rules

The ***Editor Content Rules*** are used in combination with the editor’s configuration to determine which HTML elements/attributes are supported within any article content; anything not supported will be filtered out of article content whether submitted via the editor in iCM or via one of the APIs. The content rules also determine the styles that are made available via the *Styles* drop-down list within the editor’s toolbar, and for the auto-formatting of tables.

🖎 Any HTML elements and attributes deemed not to be supported by the editor via the configuration file or the content rules ***will*** be filtered out of any content submitted as article text, whether submitted via the editor or via one of the APIs.

### Defining Content Rules

The content rules can be specified via the *stylesSet* setting within the editor configuration file or as a separate content rules file; dhtmledit\_rules.js within iCM’s custom directory. The rules within the separate file will always be used in preference to any specified via the *stylesSet* configuration setting.

The content rules file format is JSON plus /\* \*/ comment blocks. It should contain a JSON array ([]) of zero or more rules, each of which should be in the format:

{

"name" : "<Style Name>",

"element" : "<HTML Element>",

"attributes" : { "class" : "<Class Name>" }

}

The *name* and *element* values are required. The *class* attribute is optional but, if present, must tie up with the name of a class (style) definition within the editor’s style sheet. The *name* must be unique across all rules.

The *<Style Name>* is the user-friendly name that will appear within the **Styles** or **Class** drop-down lists within the editor. The *<HTML Element>* is the name of the element to which the style applies. The editor will only make this style available when an element of that type is selected. The *<Class Name>* is the name of a class (style) definition present within the editor’s style sheet. This name will be set as the value of the “class” attribute on the associated element when the style is applied within the editor.

Here are some example content rules:

[

{"name": "Normal", "element": "p"},

{"name": "Heading 1", "element": "h1"},

{"name": "Heading 2", "element": "h2"},

{"name": "Heading 3", "element": "h3"},

{"name": "Heading 4", "element": "h4"},

{"name": "Heading 5", "element": "h5"},

{"name": "Heading 6", "element": "h6"},

{"name": "Preformatted Text", "element": "pre"},

{"name": "Address", "element": "address"},

{"name": "Serif", "element": "p", "attributes": {"class": "serif"}},

{"name": "Snippet", "element": "p", "attributes": {"class": "snippet"}},

{"name": "Centre", "element": "p", "attributes": {"class": "aligncentre"}},

{"name": "Red Text", "element": "span", "attributes": {"class": "redtext"}},

{"name": "Blue Text", "element": "span", "attributes": {"class": "bluetext"}}

]

The ‘serif’, ‘snippet’, ‘aligncentre’, ‘redtext’, and ‘bluetext’ classes must have associated style definitions within the editor’s style sheet.

JSONLint (<http://jsonlint.com>) or similar may be used to verify that the JSON (minus any comments) specified in the content rules file is valid.

🖎 Paragraph formats (*H1*, *H2*, and so on) must be included as a content rule if they are to be supported. Prior to iCM version 10, the supported paragraph formats were configured via the FCKConfig.BlockFormats setting within the dhtmledit\_config.js; this is no longer supported but will need checking as part of an upgrade so that the appropriate paragraph formats can be included as a content rule. *P* will always be supported and does not need to be included as a content rule.

### Defining Content Rules for Table Auto-Formatting

The content rules must also cover any table auto-formatting styles made available via the *Auto Format Table* context menu option for a table within the editor. Although the styling is handled by the editor’s style sheet (refer to the [***Editor Style Sheet***](#_Defining_Table_Auto-Formatting) section), rules linking particular elements to the style definitions (by class name) must be included to prevent the styling being filtered out whenever the editor HTML clean-up is applied. These rules will also ensure that the auto-formatting styles will be listed in the *Class* drop-downs of the table/cell properties editor dialogs.

The *<prefix>general* style must be linked to a ‘table’ rule, the *<prefix>odd* and *<prefix>even* styles each to a ‘tr’ rule, and the remaining styles each to a ‘th’ and ‘td’ rule. Here is a set of content rules that would need to be provided for the example table auto-format styling provided in the [***Editor Style Sheet***](#_Editor_Style_Sheet) section:

[{"name":"Style 1","element":"table","attributes":{"class":"ts1general"}},

{"name":"Style 1 Top Left (TH)","element":"th","attributes":{"class":"ts1topleft"}},

{"name":"Style 1 Top Left","element":"td","attributes":{"class":"ts1topleft"}},

{"name":"Style 1 Top Centre (TH)","element":"th","attributes":{"class":"ts1topcenter"}},

{"name":"Style 1 Top Centre","element":"td","attributes":{"class":"ts1topcenter"}},

{"name":"Style 1 Top Right (TH)","element":"th","attributes":{"class":"ts1topright"}},

{"name":"Style 1 Top Right","element":"td","attributes":{"class":"ts1topright"}},

{"name":"Style 1 Mid Left (TH)","element":"th","attributes":{"class":"ts1middleleft"}},

{"name":"Style 1 Mid Left","element":"td","attributes":{"class":"ts1middleleft"}},

{"name":"Style 1 Mid Centre (TH)","element":"th","attributes":{"class":"ts1middlecenter"}},

{"name":"Style 1 Mid Centre","element":"td","attributes":{"class":"ts1middlecenter"}},

{"name":"Style 1 Mid Right (TH)","element":"th","attributes":{"class":"ts1middleright"}},

{"name":"Style 1 Mid Right","element":"td","attributes":{"class":"ts1middleright"}},

{"name":"Style 1 Bot Left (TH)","element":"th","attributes":{"class":"ts1bottomleft"}},

{"name":"Style 1 Bot Left","element":"td","attributes":{"class":"ts1bottomleft"}},

{"name":"Style 1 Bot Centre (TH)","element":"th","attributes":{"class":"ts1bottomcenter"}},

{"name":"Style 1 Bot Centre","element":"td","attributes":{"class":"ts1bottomcenter"}},

{"name":"Style 1 Bot Right (TH)","element":"th","attributes":{"class":"ts1bottomright"}},

{"name":"Style 1 Bot Right","element":"td","attributes":{"class":"ts1bottomright"}},

{"name":"Style 1 Odd (TR)","element":"tr","attributes":{"class":"ts1odd"}},

{"name":"Style 1 Even (TR)","element":"tr","attributes":{"class":"ts1even"}}]

### Generating Content Rules using an existing (Pre iCM Version 10) Editor Style Sheet

***This section is only applicable when upgrading from a version prior to iCM Version 10.***

A utility script, *createstylerules.cfm*, has been provided within the iCM custom/utils directory to aid the generation of content rules for existing installations that are being upgraded. This script will prompt for an existing editor style sheet (CSS) file as input and generate a rules (JS) file as output.

The script will search for styles (identified by their $name='Style Name' comments) within the supplied pre-version 10 editor style sheet, and it will use these to generate suggested content rules. A 'general' style definition within the old style sheet will result in a content rule being generated for ***all elements*** that were supported by the pre-version 10 editor. It is unlikely that the editor will need to validly support a particular style for all of these elements, and so they should manually be rationalised to something more sensible.

A 'table' style definition will result in a content rule being generated for each of the *table*, *t*r, *th*, *td* HTML elements. Again, these should be manually rationalised to include rules for just those elements that a particular style is to be applicable to within the editor.

🖎 The content rules file that is generated by the utility script is only ever intended to be used as a *basis* for producing the definitive set of content rules. It is **not recommended** that it ever be used as it is.

Table auto-formattting styles within the old style sheet (identified by $style='Table Style Name' comments) will also be processed. Each auto-format style comprises several style definitions that will be mapped to a content rule for the relevant table-related HTML element. So the *<prefix>general* styling will be mapped to a content rule for the *table* element, the *<prefix>odd* and *<prefix>even* styling to a content rule for the *tr* element, and all other styles (*<prefix>topleft*, *<prefix>middleright*, etc) will be mapped to content rules on the *th* and the *td* elements. Refer to the other sections on [***Table Auto-Formatting***](#_Defining_Table_Auto-Formatting) for further information about setting these styles up.

🖎 The table auto-format style definitions within editor style sheets prior to iCM version 10 should not need to be changed. The content rules **must** be defined though, to prevent the editor filtering out the associated classes whenever its clean-up is performed.

# Browser Settings

Browser setting changes may be required to enable some of the editor toolbar functionality. This is due to some browsers' security options and cannot be controlled by iCM or the editor configuration files. The cut, copy, paste options, for example, may require that clipboard access is enabled. The keyboard short cuts, however, should function regardless of whether clipboard access is enabled or not. On Windows the shortcuts are Ctrl+C for copy, Ctrl+X for cut, and Ctrl+V for paste.

# Testing/Troubleshooting

**It is important that the editing of articles is tested after the editor has been fully configured.** iCM checks for the existence of the relevant configuration and content rules, and will prevent any editing (within iCM or via one of the APIs) if they are missing. Once present, however, it is important that existing content in particular is loaded into the article editor and is checked for unexpected clean-up of content. The article preview and **Show HTML** options can be used to confirm the editor’s functionality and behaviour is as intended.

🖎 iCM will not disallow editing of article content if the configuration and rules are properly set up but the editor’s style sheet is inaccessible for whatever reason. This is used for styling of the content only and is not used by the editor’s filtering/clean-up.

iCM provides a debug/diagnostics mode that is activated by assigning the **Diagnostics** privilege to the iCM user’s account. It can also be activated by enabling **Debug** mode within the AutoConfig utility but the former is the preferred way as only one particular user will be affected.

The diagnostics mode will result in additional entries being logged to the Railo application.log file (located in the WEB-INF/railo/logs directory). This logging occurs when previewing an article, and whenever clean-up is performed by the editor and so can be useful for diagnosing configuration problems.

The **About** dialog accessed from the editor’s toolbar will also include additional information about the configuration file, content rules and style sheet it is using; links to view their contents will also be present.

Finally, the diagnostics mode will cause the HTML Text Editor to use its uncompressed source files. This is particularly useful for diagnosing any javascript errors that occur within the editor as the browser debuggers will be able provide more meaningful location information.

🖎 Avoid leaving the Diagnostics/Debug mode enabled for any length of time; it will result in an increase of output to the application.log file and will mean that the editor will be loading the uncompressed source files each time it is loaded.

# Appendix 1: Editor Configuration Diagram

The following diagram illustrates the relationships between the editor, API and the various editor configuration files.

