PEARSON ALWAYS LEARNING

EQUELLA®

HTML Editor Plug-in Guide

Version 6.4

Document History

Document No.	Reviewed		Finalised		Published	
1	22/05/2015		22/05/2015		22/05/2015	

May 2015 edition.

Property of:

Pearson
Level 1, 11 Elizabeth Street
Hobart, Australia, 7000

Contact documentation@equella.com for matters relating to documentation.

Table of Contents

EQUELLA HTML editor plug-ins

Introduction

EQUELLA administrators can upload custom written HTML editor plug-ins into EQUELLA. The plug-ins can then be used within the HTML editors throughout the EQUELLA system.

NOTE: Although the EQUELLA HTML Editor supports the uploading of third party plug-ins, bugs or issues with the plug-ins themselves are NOT supported by the EQUELLA Support Team.

An HTML editor plug-in is essentially a thin wrapper around a third-party or custom written TinyMCE plug-in. It's important to note the distinction between a TinyMCE plug-in and an HTML editor plug-in. If you obtain a TinyMCE plug-in from a third party you will need to wrap it in your own HTML editor plug-in. This document shows how one would go about writing such a plug-in.

Structure of the plug-in

The plug-in is simply a zip file with the following folder/file structure:

```
plugin (folder)
            editor_plugin_src.js
            [+ other supporting files]


resources (folder)
            [public resources] (for published HTML content)


plugin.json
config.json
client.js
```

The top level files (plugin.json, config.json, client.js) are used to identify and configure the plug-in.

plugin folder

The plugin folder contains the TinyMCE plug-in as you obtain it from a third party (or write yourself from scratch). Note that developing TinyMCE plug-ins is beyond the scope of this document. Tutorials for creating TinyMCE plug-ins are thin on the ground, although this website has a decent tutorial http://www.januarius.net/blog/?p=298

resources folder

The resources folder contains content that is used in HTML generated by your plug-in. For example, if your plug-in inserts a random image into the HTML, that image needs to be available independent of the plug-in. The base URL of the resources folder is available under the `resourcesUrl` script variable. See the section on Scripting for more information.

plugin.json

This file defines the plug-in, and its buttons, to the EQUELLA system.

The file is in JSON format (http://en.wikipedia.org/wiki/JSON) and may contain the following values:

```
{
     "id": "plugin_id",
     "name": "Plugin Name",
     "author": "Plugin Author",
     "buttons": [
            {
                   "id": "button id",
                   "title": "button hover text",
                   "image": "button image"
            },
            ...
     ]
}
```

The "id" property is mandatory and **must** match the value that the plug-in is registered as in TinyMCE. E.g. in the example plug-in found in the integration pack, the plugin/editor_plugin_src.js file registers the plug-in as "bacon_example":

```
// Register plugin
tinymce.PluginManager.add('bacon_example', tinymce.plugins.BaconExample);
```

The "name" property is displayed in the HTML editor plug-in list within EQUELLA. This property is optional, the ID field will be used if no name is supplied.

The "author" property is displayed in the HTML editor plug-in list within EQUELLA. This property is optional, "Unknown Author" will be displayed if no author is supplied.

The "buttons" property is an array value. Each array value is an object with an id, title and image property. EQUELLA will use the buttons array to place the buttons on the HTML editor toolbar editor. Failure to list any buttons will mean you cannot add any of the buttons that this plug-in requires to the HTML editor toolbar.

In the bacon_example editor_plugin_src.js file the button is registered as "bacon_example_receivebacon":

```
// Register example buttons
ed.addButton('bacon_example_receivebacon', {
     title : 'Receive Bacon',
     cmd : 'mceReceiveBacon',
     image : url + '/images/receivebacon.png'
});
```

For each button used in the plug-in editor_plugin_src.js file the id, title and image properties must be replicated in the buttons array in the plugin.json file:

```
"buttons": [
            {
                  "id":"bacon_example_receivebacon",
                  "title":"Receive Bacon",
                  "image":"images/receivebacon.png"
            }
      ]
```

config.json

The values in this file are sent to the initialisation method of the TinyMCE editor. The file is in JSON format (http://en.wikipedia.org/wiki/JSON).

The recommended method of supplying config values is to include a property with a name matching your plug-in ID, and with an object value containing the config values themselves.

E.g.

```
{
      "bacon_example":
      {
            "numberOfBaconStrips": 1,
            "image1": "${resourcesUrl}/images/bacon1.png",
            "keywords": "${xml.get('metadata/general/keyword')}"
      }
}
```

If you haven't written the plug-in yourself then it's likely the plug-in will require top level values, which seems to be standard in the TinyMCE world.

E.g.

```
{
      "someprefix_param": "some value",
      "someprefix_arg": "something else",
      "someprefix_thing": 22.3
}
```

The config.json file allows Freemarker markup as seen in the bacon_example above. See the section on Scripting for more information.

client.js

The Javascript in this file will be executed on the browser when your plug-in is loaded. You can perform additional initialisation using this file. client.js allows Freemarker markup, which will be evaluated before sending the script to the user's browser. See the section on Scripting for more information.

A contrived example could be:

```
alert('${xml.get('my/node')}');
```

This would retrieve a value from the XML on the server side and output the following script to the browser:

```
alert('node value');
```

Scripting

Both the config.json file and the client.js file allow Freemarker markup. In addition to the usual EQUELLA script objects (see "EQUELLA Scripting Guide (Advanced)" in the Integration Pack) the following script variables are also available:

- `pluginUrl`: The URL to plug-in folder of this plug-in.
- `resourcesUrl`: The URL to resources folder of this plug-in. Note that resources are copied to a central, publicly accessible, location and so the `resourcesUrl` has no relation to the `pluginUrl`.
- `institutionUrl`: The base URL of the institution.

An example config.js file could look like:

```
{
    "myplugin":
    {
        "user": {
            name:"${user.firstName} ${user.lastName}",
            "id": "${user.getID()}"
        },
        "placeholderImage": "${pluginUrl}images/placeholder.png",
        "contentImage": "${resourcesUrl}images/content.png"
    }
}
```

Uploading plug-ins to EQUELLA

To upload a third-party plug-in package

1. Select **Settings** from the navigation menu, and either type *html* in the filter box, then select **HTML editor** from the results or scroll down to **HTML editor** on the Settings list. An example is shown in Figure 1**.**

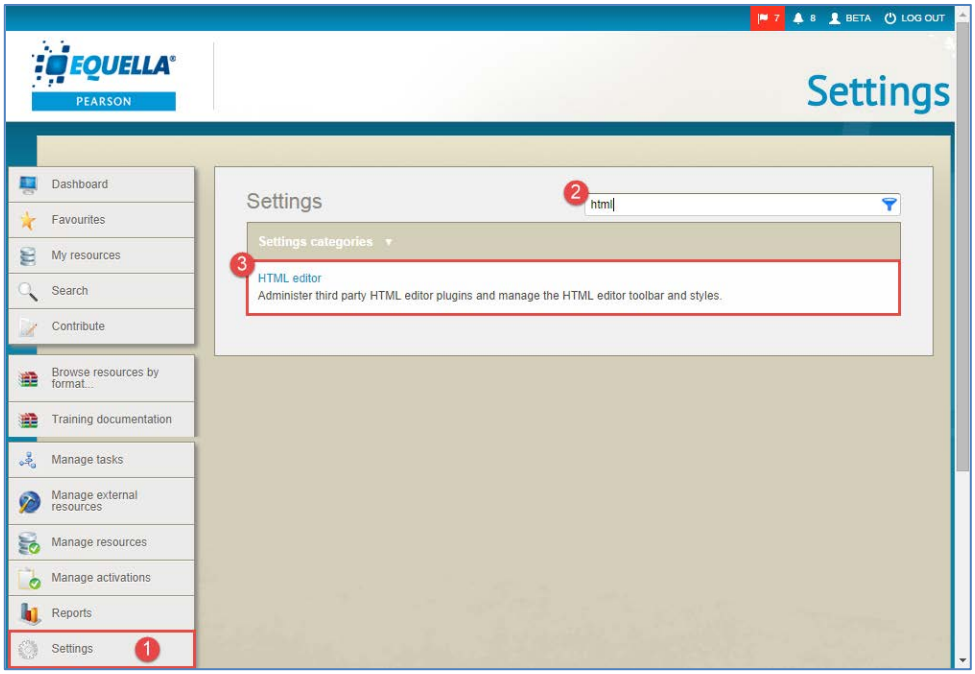

Figure 1 Settings - HTML editor

2. Select the **Plugins** link from the **HTML editor** settings page. The **HTML editor plugins** page displays. An example is shown in Figure 2.

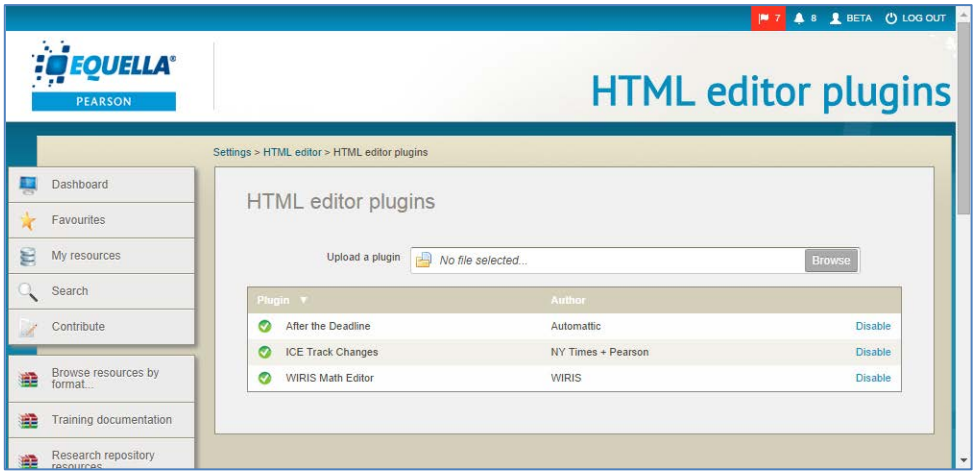

Figure 2 HTML editor plugins page

3. Click Browse to browse to and select the third-party plug-in zip file. The plug-in is uploaded and enabled.

NOTE: Once the plug-in is uploaded, the buttons specific to the plug-in will need to the added to the HTML Editor toolbar. See the EQUELLA HTML Editor User Guide *for further information.*

Once plug-ins have been uploaded into the HTML editor, they can also be disabled by clicking the **Disable** link. This makes the plug-in functionality unavailable to the HTML editor.

Contact client support

We are always happy to help.

If your organisation has a support agreement with EQUELLA then help is available at http://equella.custhelp.com/.