Using Zen Mojo Plugins

Version 1.1.2
2020-04-16

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using Zen Mojo Plugins
Caché Version 1.1.2 2020-04-16

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This book describes how to use Zen Mojo plugins. It is a supplement to the book *Using Zen Mojo*, which describes all other aspects of creating and distributing Zen Mojo applications.

The following chapters contain important introductory material, and describe special plugin classes that provide certain generic services to other plugins:

- Working with Plugins — provides an overview of plugin structure and features, and describes common tasks that must be performed when adding plugins to your application.
- Zen Mojo Default Page Manager and Helper Plugins — describes the generic page manager and a utility helper plugin that provides features used by most other helper plugins.

The following chapters provide detailed information on plugins for specific JavaScript libraries or frameworks:

- Helper Plugin for Bootstrap — describes support for the Twitter Bootstrap library.
- Helper Plugin for Chart.js — describes support for the Chart.js library.
- Helper Plugin for Google Maps — describes support for the Google Maps API.
- Helper Plugin for Highcharts — describes support for the Highcharts library.
- Helper Plugin for HTML5 — describes generic support for HTML5 elements.
- Page Manager and Helper Plugins for ChocolateChip-UI — describes support for the ChocolateChip-UI framework.
- Page Manager and Helper Plugins for Dojo — describes support for the Dojo Toolkit, including Dojo Dijit, Dojo 2D chart, and Dojo GridX.
- Page Manager and Helper Plugins for jQuery Mobile — describes page manager and helper plugins for the jQuery Mobile framework.

The following chapters provide useful information that isn't required to understand the earlier chapters:

- Creating Plugins — describes the basic requirements and procedures for creating your own plugin classes
- Using the Plugin Documentation and Widget Reference Apps — describes two Zen Mojo applications that provide additional documentation resources: *Plugin Documentation* provides detailed information on layout objects for each plugin, and the *Widget Reference* provides interactive widget samples with detailed descriptions and source code.

For a detailed outline, see the table of contents.

See *Using Zen Mojo* for details on creating and distributing Zen Mojo applications.

For general information about InterSystems documentation, see the *InterSystems Documentation Guide*.

1

Working with Plugins

Zen Mojo *plugins* are utility classes containing layout objects that provide access to several popular third-party JavaScript libraries and frameworks. Zen Mojo currently provides plugins for the Twitter Bootstrap library, the Chart.js library, the ChocolateChip-UI framework, the Dojo Toolkit, the Google Maps API, the Highcharts library, the jQuery Mobile framework, and HTML5 standard. You can also extend Zen Mojo by creating your own plugins for other third-party libraries.

This chapter discusses the following topics:

- Overview — provides a simple description of helper plugins, which define layout objects, and page manager plugins, which contain specified sets of helper plugins.
- Choosing Plugins for Your Application — lists the available helper and page manager plugins, and discusses how each one might be used.
- Adding Plugins to the Page Class — describes the basic procedures for specifying plugins and related resources in your page class.
- Detecting and Resolving Plugin Conflicts — describes how to resolve conflicts when two plugins use the same name for a layout object.
- Implementing Required Methods in the Page Class — explains when and how to implement the **adjustContentSize()** and **onPageShow()** methods.
- Using Plugin Resources in a Template Class — demonstrates how to use special methods provided by helper plugins and layout objects.

1.1 Overview

A Zen Mojo application consists of the following primary components:

- one or more *template classes*, which determine what is displayed on your web page
- a set of *plugin classes*, which define the layout objects used by the templates
- the *page class*, which specifies the plugin classes and other resources available to your template classes

A set of plugins must by specified for each documentView defined in the **pageContents** XData block of your page class (for details, see "Adding Plugins to the Page Class" later in this chapter).

Each documentView element must specify a single *page manager* plugin, which will contain one or more *helper plugins*. Each helper plugin defines a set of *layout objects*. These elements perform the following functions:

- *A page manager plugin* provides the interface between the templates and a set of helper plugins.

  The page manager creates a list of all layout objects defined in the specified list of helper plugins, and resolves any naming conflicts (for example, two helper plugins may define two different button layout objects). Page managers may also provide library-specific utility functions that can be used by all layout objects associated with a given JavaScript library or framework.

- *Helper plugins* define layout objects and other library-specific resources used by template classes.

  When you specify a helper plugin in your page class, the layout objects defined in that helper become available to your template classes. A helper plugin may also provide additional resources, such as utility methods for common tasks.

- *Layout objects* use library functions to produce HTML strings.

  A template class uses one or more layout objects to determine the appearance of your web page. A typical layout object renders a graphical element, usually by using a corresponding element in the JavaScript library. For example, a button layout object might render itself by calling a **button** function in the library.

1.2 Choosing Plugins for Your Application

Zen Mojo provides plugins for several popular JavaScript libraries and frameworks. When designing your Zen Mojo application, pick the helpers that you will need, and then chose a page manager that supports all of them.

The following helper plugins can be used in any application:

- Zen Mojo Default helper — provides utility layout objects used by most helper plugins. It can be used by any page manager (not just the Default page manager).
- HTML5 helper — provides layout objects for most standard HTML5 elements.
- Bootstrap helper — provides layout objects for many of Twitter Bootstrap's enhanced versions of HTML5 elements.
- Chart.js helper — provides layout objects for the lightweight Chart.js library.
- Highcharts helper — similar to Chart.js, but Highcharts provides a much greater variety of charts.
- Google Maps helper — provides layout objects that support an online connection to the Google Maps API.

These helper plugins can be used with any page manager.

All other helper plugins require support for a specific JavaScript library or framework, and can only be used by the page manager that provides that support. The following page managers and specialized helpers are available:

- The Default page manager supports only the previously listed helpers.
- The ChocolateChip-UI page manager supports a helper containing layout objects for the enhanced ChocolateChip-UI versions of HTML5 elements. This page manager is best for relatively simple mobile applications.
- The jQuery Mobile page manager supports a helper containing layout objects for the long list jQuery Mobile enhanced HTML5 elements. This page manager may be preferable for complex mobile applications.
- The Dojo page manager supports three helpers for three different APIs in the Dojo Toolkit. This page manager is best for complex desktop applications.

1.3 Adding Plugins to the Page Class

Plugins are added to your application by specifying some or all of the following items in the page class:

- *External JavaScript and CSS files*

  Most helper plugins depend on external JavaScript and CSS files. All required files should be located in or under `<install-dir>`/csp/broker (where `<install-dir>` is the is directory in which Caché is installed). Lists of installed files are specified as comma-delimited strings in the *JSINCLUDES* and *CSSINCLUDES* (or *CSS3INCLUDES*) parameters. The following example demonstrates how to list the required files for the Bootstrap helper plugin:

  ```
  Parameter JSINCLUDES As STRING = "jquery-1.11.3.min.js,bootstrap-3.3.5/js/bootstrap.min.js"
  Parameter CSSINCLUDES As STRING = "bootstrap-3.3.5/css/bootstrap.min.css"
  ```

  Caché assumes that /csp/broker is the root directory for each of these paths and filenames.

  The order of the *JSINCLUDES* list determines the order in which the JavaScript libraries will be loaded, from left to right. In this example, the jQuery library must be loaded first because the Bootstrap library depends on it.

- *Plugin page manager and helper classes*

  In the **pageContents** XData block of your page class, you must register one or more plugins for use in each documentView (see "Basic Definition of a Zen Mojo Page" in the book *Using Zen Mojo*). To do so:

  – Include only one page manager plugin as a child element of `<mojo:documentView>`.

  – Include one or more helper plugins as child elements of the page manager plugin.

  The following example shows the general structure (with optional indentation for clarity):

  ```
  <mojo:documentView id="mainView" ongetlayout = "return zenPage.getContent('layout',key);">
    <mojo:mojoDefaultPageManager>
      <mojo:bootstrap-3.3.x-Helper/>
      <mojo:mojoDefaultHelper/>
    </mojo:mojoDefaultPageManager>
  </mojo:documentView>
  ```

  The order in which you list helper plugins is important. When two plugins use the same name for a layout object, your application will use the layout object from the plugin listed first. Later instances of layout objects with the same name will be ignored. See "Detecting and Resolving Plugin Conflicts" for details.

- *Required page methods*

  Depending on the plugins you choose, your page class may have to implement certain required methods. See "Implementing Required Methods in the Page Class" for details.

For each plugin or set of plugins, this book describes all page class requirements at the beginning of the relevant chapter. See the Bootstrap chapter for the information used in the previous examples.

1.4 Detecting and Resolving Plugin Conflicts

It is important to consider the order in which you list the helper plugins, in the case of *plugin conflict*. A plugin conflict occurs when a single documentView uses multiple helper plugins and those plugins have layout objects with the same name. A plugin conflict is not an error condition, but rather a situation that requires special handling. There is a default behavior and a way to override it.

In the case of a plugin conflict, by default, Zen Mojo uses the layout object rendering logic as given in the helper plugin that is listed *first*. If you do not want the default behavior, you can modify it.

To override the default handling, do the following:

- Specify the `onresolvepluginconflicts` callback attribute of the documentView. Use a value like the following:

  ```
  onresolvepluginconflicts="zenPage.onResolvePluginConflicts(zenThis, conflicts);"
  ```

  In this callback, you can use the variable *conflicts*, which Zen Mojo provides. This variable is an array that contains the names of layout objects defined by more than one plugin.

- Implement the method to which your callback attribute refers. In this method, use the **setPluginMapping()** method of the documentView instance. The following shows a simple example:

  ```
  ClientMethod onResolvePluginConflicts(docView, conflicts) [ Language = javascript ]
  {
    for (prop in conflicts) {
        if (conflicts[prop].indexOf('HTML5') > -1) {
            docView.setPluginMapping(prop, 'HTML5');
        }
        else if (conflicts[prop].indexOf('html5') > -1) {
            docView.setPluginMapping(prop, 'html5');
        }
    }
  }
  ```

  For any given layout object type, your implementation can specify which helper plugin should perform the rendering for that layout object type.

1.5 Implementing Required Methods in the Page Class

Depending on the plugins you choose, your application may have to implement the **adjustContentSize()** method or the **onPageShow()** callback, as described in the following sections:

- Overriding the adjustContentSize() Method — this method specifies the size and position of each documentView. This is an existing page method that requires an override when using the Default or Dojo page managers.
- Implementing the onPageShow() Callback — this callback method performs extra processing before the page is displayed. It is required when the Google Maps helper is used, and optional in all other cases.

1.5.1 Overriding the adjustContentSize() Page Method

An override for the basePage.**adjustContentSize()** page method must be implemented in your page class if it uses the Default page manager or the Dojo page manager.

The purpose of the **adjustContentSize()** method is to specify the size and position of each documentView, based on the current width and height of the **pageContents** pane, which varies depending on the screen size and current rotation. This method is called when the basePage.**onlayoutHandler()** event is triggered. It has the following signature:

```
ClientMethod adjustContentSize(load, width, height) [ Language = javascript ]
```

Where *load* indicates if the page is being loaded. This argument is 1 when the page is loaded and is 0 at other times. *width* and *height* are the current width and height, respectively, of the **pageContents** pane, in pixels.

In this method, it is typically necessary to do the following for each documentView on this page:

- Get a reference to the documentView and call its **setSize()** method. The following example gets an instance of documentView with id `mainView`, and uses the *width* and *height* arguments received when **adjustContentSize()** is called from **onlayoutHandler()**:

```
var mainView = zen('mainView');
mainView.setSize(width, height);
```

  where *width* and *height* are the desired width and height in pixels.

- If there is more than one documentView on the page, specify the position of the top, left corner of each documentView. Use the documentView.**getEnclosingDiv()** instance method to get the `<div>` that contains the instance, then specify the style.top and style.left properties of the `<div>`. Specify the values as `'nnnpx'` where *nnn* is an integer. For example:

```
var mainDiv = mainView.getEnclosingDiv();
mainDiv.style.top =  '0px';
mainDiv.style.left = '0px';
```

Example: Implementing adjustContentSize() for a page with two documentView instances

In the following example, there are two documentView instances. The left one is one-fourth of the width of the **pageContents** area, and the right one uses the rest of the space.

```
ClientMethod adjustContentSize(load, width, height) [ Language = javascript ]
{
    var leftView = zen('leftView');
    var mainView = zen('mainView');
    var leftWidth = Math.floor(width/4);

    // This method should have an if{} block for each component.

    if (leftView) {
        leftView.setSize(leftWidth-2,height);
        var leftDiv = leftView.getEnclosingDiv();
        leftDiv.style.top = '0px';
    }
    if (mainView) {
        mainView.setSize(width - leftWidth - 2, height);
        var mainDiv = mainView.getEnclosingDiv();
        mainDiv.style.top = '0px';
        mainDiv.style.left = leftWidth + 'px';
    }
}
```

1.5.2 Implementing the onPageShow() Callback

You can specify the optional `onPageShow` callback attribute in the page manager element of your **pageContents** XData block, as demonstrated in the following example:

```
<mojo:chui-3.5.2-PageManager onPageShow="zenPage.onPageShow(layoutkey,documentkey);">
```

This attribute defines the callback method to be invoked after rendering the layout objects for the plugin. For example, you could implement a method that adjusts the display for a layout object provided by a different plugin. Each page manager plugin contains appropriate code to invoke the callback if it has been defined.

Currently, the callback is only required when using the Google Maps plugin, which needs a way to resize the map after rendering the other page items. The following example implements a callback method that triggers a Google Maps resize event:

Example: Implementing onPageShow() for Google Maps

Specify the `onPageShow` callback attribute of the page manager element. This example uses the jQuery Mobile page manager:

```
<mojo:jQM-1.4.5-PageManager onPageShow="zenPage.onPageShow(layoutkey,documentkey);">
```

In your page class, implement a method similar to the following:

```
ClientMethod onPageShow(layoutkey, documentkey) [ Language = javascript ]
{
  if (layoutkey == 'maps-demo') {
    zen('mainView').getPluginByLayoutObjectType('$map').resizeMap();
  }
}
```

This example assumes that the application defines the `'maps-demo'` layout key, and that the documentView `id` attribute is `'mainView'`. See "Custom Google Maps Helper Plugin Methods" for information on the **resizeMap()** method.

1.6 Using Plugin Resources in a Template Class

Both helper plugins and layout objects sometimes contain methods that provide object-specific utilities, as described in the following sections:

- Helper Plugin Methods — typically provide access to utilities from the supporting JavaScript library.
- Layout Object Methods — provide extra functionality to individual layout objects.

1.6.1 Helper Plugin Methods

Some plugin classes provide special methods that directly invoke utilities defined in the supporting JavaScript library. To use them, get an instance of the plugin object (via documentView methods such as **getPluginByLayoutObjectType()** or **getPluginByName()**), and then call the plugin method. The following example gets a Google Maps plugin object and calls its **resizeMap()** method. The plugin method then uses the current map instance to directly access the Google Maps API and trigger a **resize** event:

```
zen('mainView').getPluginByLayoutObjectType('$map').resizeMap();
```

For each plugin, information on available plugin methods is listed in the relevant chapter under the heading "<*Plugin-name*> Helper Plugin Methods". Some chapters also include a more detailed "Custom <*Plugin-name*> Helper Plugin Methods" section (for example, the **resizemap()** function used in this example is described under "Custom Google Maps Helper Plugin Methods" in the Google Maps chapter).

1.6.2 Layout Object Methods

Individual layout objects may also contain special methods. For example, many layout objects have a **$refresh** method. The following call would refresh the display of the layout object identified by key `'person1'`:

```
zen('mainView').getItemByKey('person1').$refresh();
```

For each plugin, details on available layout object methods are listed in the relevant chapter under the heading "Custom <*Plugin-name*> Layout Object Methods".

2

Zen Mojo Default Page Manager and Helper Plugins

This section describes two light-weight plugins that can be used in many different configurations:

- The *Default page manager* plugin can be used with all helper plugins except those provided specifically for Dojo, ChocolateChip-UI, or jQuery Mobile.
- The *Default helper plugin* can be used with any page manager, and in combination with any helper plugin. It provides the $loop, $if, and $content utility layout objects used extensively by many other helper plugins.

These plugins are suitable for use on both mobile devices and desktop computers. Neither of them require any external JavaScript or CSS files.

2.1 Adding the Default Plugins to the Page Class

Adding Required Files

These plugins do not require any external CSS or JavaScript files.

Registering the Default Plugins

The following example demonstrates how the Default page manager and helper can be registered in your **pageContents** XData block (as described in "Adding Plugins to the Page Class"). The HTML5 helper in this example is not required, but is frequently useful:

```
<mojo:mojoDefaultPageManager>
  <mojo:HTML5Helper/>
  <mojo:mojoDefaultHelper/>
</mojo:mojoDefaultPageManager>
```

The Default page manager can be used with all helper plugins except those provided specifically for ChocolateChip-UI, Dojo, or jQuery Mobile.

The Default helper can be used with any page manager, and is compatible with all helper plugins. It should be listed *after* all other helpers in the **pageContents** XData block.

Other Options and Requirements

- Your page class must implement an override for page method **adjustContentSize()**. This method must specify the width, height, and position of each documentView. For a details and examples, see "Overriding the adjustContentSize() Method" in the "Working with Plugins" chapter.

For more information on these plugins, see %ZEN.Mojo.Plugin.mojoDefaultPageManager and %ZEN.Mojo.Plugin.mojoDefaultHelper in the class reference.

2.2 Using the Default Helper in a Template Class

Default Helper Layout Objects

This plugin contains the $content, $if, and $loop utility layout objects. See "Default Helper Layout Objects" later in this chapter for details.

Custom Default Helper Layout Object Methods

This plugin does not provide any layout object methods.

Default Helper Plugin Methods

The Default helper plugin does not include any utility methods other than those inherited from %ZEN.Mojo.Plugin.baseHelperPlugin.

Direct Access to the Default Helper Library

Not applicable — this plugin does not use an external library.

Event Handling

This plugin has no special event handling requirements.

2.3 Default Helper Layout Objects

The Zen Mojo Default helper plugin contains a set of layout objects that provide utility functions for layout objects in other plugins:

- The $content Layout Object — can contain arbitrary HTML.
- The $if Layout Object — controls whether a page displays specific objects.
- The $loop Layout Object — is a general purpose iterator used in most other helper plugins.

For reference information on the layout objects provided by this plugin, use the Plugin Documentation app (see "Using the Plugin Documentation and Widget Reference Apps").

2.3.1 The $content Layout Object

The $content layout object can contain arbitrary HTML, as long as that HTML is well-formed. For example:

```
var htmlstring =        '<h2>The Demo Team</h2>'
htmlstring=htmlstring + '<p><strong>You have been invited to a meeting in Boston</strong></p>';
htmlstring=htmlstring + '<p>Are you available tomorrow at 10am?</p>';
htmlstring=htmlstring + '<p class="ui-li-aside"><strong>6:24 </strong>PM</p>'
myLayout = {
  children: [ { type: '$content', title: '$content demo', content:htmlstring } ]
}
```

2.3.2 The $if Layout Object

The $if layout object is used to control whether a page displays specific objects. The $if object provides the properties `value`, `expectedValue`, and `children`. If `value` equals `expectedValue`, then Zen Mojo renders the layout objects provided in the `children` property; otherwise, Zen Mojo does not render anything for the $if object.

2.3.3 The $loop Layout Object

The $loop layout object is a general-purpose iterator. Use this, for example, to create a table that has a varying number of rows; see an example in ZMdemo.html5.baseTemplate. Many of the other plugins use $loop objects internally. Note that loops can be nested.

In special cases, a layout object provides variables that you can use within the layout graph. For example, the default Mojo helper plugin provides the layout object $loop, which provides several variables including $loopindex, $loopnumber, and $loopValue.

To refer to the variable *variablename*, use the syntax `'=[variablename]'`

The following example shows a reference to the $loopValue variable. This example also uses the $div layout object from the HTML5 helper plugin.

```
myLayout = {
    children: [
        { type: '$loop', title: 'Basic $loop demo', value:['apples','bananas','coconuts'],
         children: [
          {type:'$div', $content:'=[$loopValue]'}
         ]}
    ]
}
```

In general, to use the $loop object:

- Specify its `value` property, which is an array. The number of items in the `value` array determines the number of items in the loop.

  Each array item can be an object, an array, or a literal value. If the array items are objects, each object should typically have the same set of properties. Similarly, if the array items are arrays, each array should have the same structure.

- Specify its `children` property, which is an array of layout objects. The loop will contain one set of these objects for each item in the `value` array.

  For these layout objects, you can refer the current item of the `value` array. To do so, use the syntax `'=[value to obtain]'`.

  For example, if the array items are objects, use `'=[property]'`, where *property* is a property of any object in the array. (You can refer to properties of properties, and so on, as well, via dot syntax.)

Example: Basic $loop demo

The following shows an example that also uses the $div object from the Zen Mojo HTML5 helper plugin:

```
myLayout = { children: [
  { type: '$loop', title: 'Basic $loop demo',
    value:[
      {product:'apples',  sku:'SKU-001'},
      {product:'bananas', sku:'SKU-002'},
      {product:'coconuts',sku:'SKU-003'}
    ],
    children: [ {type:'$div', title:'=[product]', $content:'=[sku]'} ]
  }
]}
```

2.3.3.1 Variables Available within a $loop

When you use $loop, the following variables are available within your layout graph, in addition to any variables you create:

- *$loopIndex* is the 0–based index for the loop iteration.
- *$loopNumber* is the loop count. This variable equals `$loopIndex + 1`.
- *$loopValue* is the current array item from the `value` property.
- *$loopKey* is tbd

  For any layout object within a $loop, Zen Mojo provides the following composite value for the `key` property of that object: *elementkey*:*index* where *elementkey* is the value of the `key` property for this layout object, as defined in your method, and *index* is the 0–based index for the loop iteration.

Example: $loop used with the $loopValue variable

The following example uses an array of strings for the `value` property and uses the *$loopValue* variable to display each of them in turn. Like the previous example, it uses the $div layout object from the HTML5 helper plugin.

```
myLayout = { children: [
  { type: '$loop', title: 'Basic $loop demo',
    value:['apples','bananas','coconuts'],
    children: [ {type:'$div', $content:'=[$loopValue]'} ]
  }
]}
```

2.4 Using Default Helper and Page Manager Sample Code

Many sample applications use these plugins. The Bootstrap sample is typical (see “Using Bootstrap Plugin Sample Code” for details).

3

Helper Plugin for Bootstrap

The Bootstrap helper plugin provides support for the Twitter Bootstrap library. This plugin can be used with any page manager, and is best suited for use on mobile devices. For detailed information about Bootstrap, see http://getbootstrap.com/getting-started/.

3.1 Adding Bootstrap to the Page Class

Adding Required Bootstrap Files

The files listed in this section are contained in the Bootstrap 3.3.5 release and the jQuery 1.11.3 release. If they are not already installed on your system, you can get Bootstrap at https://github.com/twbs/bootstrap/archive/v3.3.5.zip and jQuery at https://code.jquery.com/jquery/.

These instructions assume that the Bootstrap files are in a subdirectory of `<install-dir>`/csp/broker named bootstrap-3-3-5 (see "Adding Plugins to the Page Class" for details). Note that the top-level directory name in the Bootstrap zip file is bootstrap-3.3.5, which must be changed to bootstrap-3-3-5 in /csp/broker. The following files must be available:

- dist/js/bootstrap.min.js
- dist/css/bootstrap.min.css

Bootstrap also requires the jQuery library. The following file must be in /csp/broker:

- jquery-1.11.3.min.js

Add the following filename strings to the *JSINCLUDES* and *CSSINCLUDES* (or *CSS3INCLUDES*) parameters of your page class. Caché assumes that /csp/broker is the root directory for all relative paths:

- *CSSINCLUDES* filenames:

  ```
  "bootstrap-3-3-5/dist/css/bootstrap.min.css"
  ```

- *JSINCLUDES* filenames:

  ```
  "jquery-1.11.3.min.js, bootstrap-3-3-5/dist/js/bootstrap.min.js"
  ```

  Bear in mind that files are loaded in the order that they are listed in the parameters. In this case, the jQuery library must be listed first because the Bootstrap library depends on it.

Registering the Bootstrap Helper Plugin

The following example demonstrates how the Bootstrap helper can be registered in your **pageContents** XData block (as described in "Adding Plugins to the Page Class"). This example uses the Default page manager, but the Bootstrap helper will work with any page manager:

```
<mojo:mojoDefaultPageManager>
  <mojo:bootstrap-3.3.x-Helper/>
  <mojo:mojoDefaultHelper/>
</mojo:mojoDefaultPageManager>
```

The Default helper plugin is required by most Bootstrap helper layout objects, and should always be the last helper in the list. Other helpers may be added to this list.

Plugin conflicts are likely when the Bootstrap helper is used with other helpers (such as HTML5 or ChocolateChip-UI) that also provide extended HTML5 elements. To resolve all plugin conflicts in favor of Bootstrap, list the Bootstrap helper first (see "Detecting and Resolving Plugin Conflicts").

Other Options and Requirements

Most Bootstrap helper layout objects require the $loop layout object provided by the Default helper plugin. These layout objects will display the string `"($loop)"` in place of the expected output if the Default helper is not included.

For more information on this plugin, see %ZEN.Mojo.Plugin.bootstrap33xHelper in the class reference.

3.2 Using Bootstrap in a Template Class

Bootstrap Layout Objects

See "Bootstrap Helper Layout Objects" later in this chapter for a complete list of available layout objects.

Custom Bootstrap Layout Object Methods

The following methods are available within all appropriate layout objects:

- **$hide()** — Hides the layout object.
- **$refresh()** — Re-renders the HTML for the layout object.
- **$show()** — Shows the layout object.

Bootstrap Helper Plugin Methods

This helper plugin includes several utility methods other than those inherited from %ZEN.Mojo.Plugin.baseHelperPlugin. See %ZEN.Mojo.Plugin.bootstrap33xHelper in the class reference for details.

Direct Access to the Bootstrap Library

This plugin does not provide direct access to Bootstrap library functions.

Event Handling

This plugin has no special event handling requirements.

3.2.1 Bootstrap Helper Layout Objects

The following list is intended as a quick reference to the layout options offered by the Bootstrap helper plugin. For complete reference information, use the Plugin Documentation app (see "Using the Plugin Documentation and Widget Reference Apps"):

$alert	$dropdown	$inputGroupAddon	$ol	$span
$badge	$dropdownMenuItem	$jumbotron	$option	$table
$breadcrumb	$fieldset	$label	$pageHeader	$tableBody
$break	$form	$li	$pager	$tableCell
$button	$formGroup	$link	$pagination	$tableColumn
$buttonGroup	$glyphicon	$listGroup	$panel	$tableFooter
$buttonToolbar	$gridColumn	$listGroupItem	$panelBody	$tableHeader
$caret	$gridSystem	$mediaObject	$panelFooter	$tableRow
$close	$h1	$modal	$panelHeading	$text
$container	$image	$nav	$progressBar	$thumbnail
$containerFluid	$input	$navbar	$raw	$ul
$div	$inputGroup	$navbarForm	$responsiveEmbed	$well

Since the Bootstrap helper provides extended versions of HTML5 elements, plugin conflicts are likely when the it is used with other helpers that extend HTML5 elements (see "Detecting and Resolving Plugin Conflicts" earlier in this book).

3.3 Using Bootstrap Plugin Sample Code

The Widget Reference offers interactive widget samples for the Bootstrap helper. The reference includes working examples, detailed descriptions, and source code that you can cut and paste into your own applications.

A complete sample application is also available for the Bootstrap helper. In Studio, go to ZMdemo.bootstrap in the SAMPLES namespace. Compile the classes in that package if necessary. To run the application, open ZMdemo.bootstrap.HomePage and select `View > Web Page`. When running in debug mode, remember to set the debug target URL to csp/samples/ZMdemo.bootstrap.HomePage.cls.

In addition to the Bootstrap helper, the sample application also uses the Default page manager, the Default helper, and the HTML5 helper.

4

Helper Plugin for Chart.js

The Chart.js helper plugin provides support for the Chart.js library. This plugin is best suited for use on mobile devices. For detailed information about Chart.js, see http://www.chartjs.org.

4.1 Adding Chart.js to the Page Class

Adding Required Chart.js Files

If the following required file has not already been installed, you can get it from the Chart.js version 1.0.1 source available at https://github.com/nnnick/Chart.js/releases:

- Chart.min.js

These instructions assume that this file is in `<install-dir>`/csp/broker (see "Adding Plugins to the Page Class" for details). Add the following filename string to the *JSINCLUDES* parameter of your page class. Caché assumes that /csp/broker is the root directory for all relative paths:

- *JSINCLUDES* filenames:

  ```
  "Chart.min.js"
  ```

Bear in mind that files are loaded in the order that they are listed in the parameters. If you also use the Dojo 2D Chart helper plugin, be sure to list the Chart.js library before the Dojo library in the *JSINCLUDES* parameter.

Registering the Chart.js Helper Plugin

The following example demonstrates how the Chart.js helper can be registered in your **pageContents** XData block (as described in "Adding Plugins to the Page Class"). This example uses the Default page manager, but any page manager will work:

```
<mojo:mojoDefaultPageManager>
  <mojo:charts-1.0.1-Helper/>
</mojo:mojoDefaultPageManager>
```

The Chart.js helper plugin can be used with other page manager and helper plugins. If the Highcharts helper plugin is also used, some plugin conflicts will occur (see "Chart.js Layout Objects" in the following section for details).

Other Options and Requirements

For more information on this plugin, see %ZEN.Mojo.Plugin.charts101Helper in the class reference.

4.2 Using Chart.js in a Template Class

Chart.js Layout Objects

The following list is intended as a quick reference to the layout options offered by the Chart.js helper plugin. For complete reference information, use the Plugin Documentation app (see "Using the Plugin Documentation and Widget Reference Apps"):

- $chart displays a chart specified by type ($barchart by default). The other layout objects display specific chart types:

- $barchart
- $doughnutchart
- $linechart
- $piechart
- $polarareachart
- $radarchart

If both the Chart.js helper and the Highcharts helper are registered in the same **pageContents** XData block, plugin conflicts should be resolved for the following objects: $chart, $barchart, $linechart, $piechart. To resolve all conflicts in favor of Chart.js, just list it before Highcharts in **pageContents**. See "Detecting and Resolving Plugin Conflicts" for more information.

Custom Chart.js Layout Object Methods

The following methods are available within all appropriate layout objects:

- **$hide()** — Hides the layout object.
- **$refresh()** — Re-renders the HTML for the layout object.
- **$show()** — Shows the layout object.

Chart.js Helper Plugin Methods

See "Custom Chart.js Helper Plugin Methods" later in this chapter for a list of Chart.js utility methods.

Direct Access to the Chart.js Library

This plugin does not provide direct access to Chart.js library functions.

Event Handling

The **onselect()** event handler must be implemented for each Chart.js chart type that your application will use. For details, see "Implementing onselect() for Chart.js Events" later in this chapter.

4.2.1 Custom Chart.js Helper Plugin Methods

In addition to methods inherited from %ZEN.Mojo.Plugin.baseHelperPlugin, the Chart.js helper plugin provides the following utility methods. Each of these methods will call the corresponding Chart.js library prototype method for the appropriate chart type.

- **addData()**

```
method addData(key, data, labelOrIndex) [ Language = javascript ]
```

Adds data to the chart specified by *key*. The *labelOrIndex* parameter is expected to be a dataset index in doughnut, pie, and polar charts.

- **removeData()**

```
method (key, index) [ Language = javascript ]
```

Removes data from the chart specified by *key*. The *index* parameter only applies to doughnut, pie, and polar chart types. If *index* is not specified for those chart types, the last data element is removed. The other chart types simply remove the first variable.

- **update()**

```
method update(key) [ Language = javascript ]
```

Updates and re-renders the chart specified by *key*.

4.2.2 Implementing onselect() for Chart.js Events

The Chart.js helper plugin supports event handling for each chart type. To handle a selection event, your template must implement an override of the contentTemplate.**onselect()** method (see "Event Handling" in *Using Zen Mojo*). When a chart layout object is selected, the plugin passes the following arguments to **onselect()**:

- *key* — the key of the selected layout object
- *value* — an array of values
- *docViewId* — the id of the documentView

The chart is driven by the *value* data object. The *value* argument passed to **onselect()** is an array returned by a Chart.js library prototype method of the chart instance, and has a structure that varies according to the chart type:

- $linechart and $radarchart — *value* is an array returned by the **getPointsAtEvent()** method (see http://www.chartjs.org/docs/#line-chart-prototype-methods).
- $barchart — *value* is an array returned by the **getBarsAtEvent()** method (see http://www.chartjs.org/docs/#bar-chart-prototype-methods).
- $piechart, $doughnutchart, and $polarareachart — *value* is an array returned by the **getSegmentsAtEvent()** method (see http://www.chartjs.org/docs/#polar-area-chart-prototype-methods).
- $chart — *value* is an array returned by the method appropriate to the specified chart type.

4.3 Using Chart.js Plugin Sample Code

The Widget Reference offers interactive widget samples for the Chart.js helper. The reference includes working examples, detailed descriptions, and source code that you can cut and paste into your own applications.

5

Helper Plugin for Google Maps

The Google Maps helper plugin provides support for the Google Maps API. This plugin is suitable for use on mobile devices or on desktop computers. For detailed information about the Google Maps API, see https://developers.google.com/maps/documentation/javascript/.

5.1 Adding Google Maps to the Page Class

Accessing the Google Maps API

This plugin does not require any downloaded files, but it must have online access to the Google Maps API. In the *JSINCLUDES* parameter of your page class, add the following URL in place of a local filename (see "Adding Plugins to the Page Class"):

```
https://maps.googleapis.com/maps/api/js?sensor=true
```

Registering the Google Maps Helper Plugin

The following example demonstrates how the Google Maps helper can be registered in your **pageContents** XData block (as described in "Adding Plugins to the Page Class"). This example uses the Default page manager, but any page manager will work:

```
<mojo:mojoDefaultPageManager onPageShow="zenPage.onPageShow(key);">
  <mojo:googleMaps-3-Helper/>
  <mojo:mojoDefaultHelper/>
</mojo:mojoDefaultPageManager>
```

Important: **onPageShow() is Required**

Regardless of the page manager used, the `onPageShow` attribute must be specified (as shown in this example) and **onPageShow()** must be implemented to ensure that the web page is properly displayed. See "Implementing the onPageShow() Callback" for an example that uses the Google Maps plugin.

The Google Maps helper plugin can be used with other page manager and helper plugins. The Default helper plugin is required only by the `$map` layout object.

Other Options and Requirements

The `$map` layout object requires the `$loop` layout object provided by the Default helper plugin. This layout object will display the string `"($loop)"` in place of the expected output if the Default helper is not included.

For more information on this plugin, see %ZEN.Mojo.Plugin.googleMaps3Helper in the class reference.

Note: **Deprecated Google Maps Plugin Class**

The %ZEN.Mojo.Plugin.googleMapsHelper helper plugin class is currently available for backward compatibility with earlier versions of Zen Mojo. Usage is identical to the current plugin except that the older class is registered using the following elements:

```
<mojo:mojoDefaultPageManager onPageShow="zenPage.onPageShow(key);">
  <mojo:googleMapsHelper/>
  <mojo:mojoDefaultHelper/>
</mojo:mojoDefaultPageManager>
```

This older class is not being updated and will be removed in a later release.

5.2 Using Google Maps in a Template Class

Google Maps Layout Objects

The following list is intended as a quick reference to the layout options offered by the Google Maps helper plugin. For complete reference information, use the Plugin Documentation app (see "Using the Plugin Documentation and Widget Reference Apps"):

- $map — represents a map. Requires $loop from the Default helper plugin.
- $marker — represents a marker on a map.
- $directions — represents directions between two map points.
- $infowindow — represents an info window on a map. The Default helper plugin's $content layout object can be used to specify custom text content. HTML tags are recognized by the Google Maps API.

Custom Google Maps Layout Object Methods

The following method is available within all appropriate layout objects:

- **$refresh()** — Re-renders the HTML for the layout object. All Google Maps layout objects can be refreshed.

Google Maps Helper Plugin Methods

See "Custom Google Maps Helper Plugin Methods" later in this chapter for a list of Google Maps utility methods.

Direct Access to the Google Maps API

This plugin provides direct access to the Google Maps API by using the **getGMap()** method to get the associated Google Map object (see the **getGMap()** entry in the following section for details).

Event Handling

This plugin has no special event handling requirements.

5.2.1 Custom Google Maps Helper Plugin Methods

In addition to methods inherited from %ZEN.Mojo.Plugin.baseHelperPlugin, the Google Maps helper plugin provides the following methods:

- **closeInfoWindow()**

```
ClientMethod closeInfoWindow(infoKey) [ Language = javascript ]
```

Closes an info window specified by its key.

- **getGMap()**

```
ClientMethod getGMap() [ Language = javascript ]
```

Returns the Google Map object of the current level. Direct access to the Google Maps API is provided by the methods and properties of the returned object. Returns null if the map object does not exist.

- **getMarkers()**

```
ClientMethod getMarkers() [ Language = javascript ]
```

Returns an array of the `$marker` objects on the current level of the application. If the map object does not exist, this method returns null.

- **openInfoWindow()**

```
ClientMethod openInfoWindow(infoKey, markerKey) [ Language = javascript ]
```

Opens an info window specified by its key. If the key of a marker is specified as well, the info window will open on that marker.

- **refreshMap()**

```
ClientMethod refreshMap() [ Language = javascript ]
```

Triggers the refresh event for the map.

- **resizeMap()**

```
ClientMethod resizeMap() [ Language = javascript ]
```

Triggers the resize event for the map. For example:

```
var map = zen('mainView').getPluginByLayoutObjectType('$map');
map.resizeMap();
```

Note: This plugin also has several other utility methods not listed here. See %ZEN.Mojo.Plugin.googleMaps3Helper in the class reference for details.

5.3 Using Google Maps Plugin Sample Code

The Google Maps helper plugin is demonstrated in the jQuery Mobile sample application. In Studio, go to ZMdemo.JQM145 in the SAMPLES namespace. Compile the classes in that package if necessary. To run the application, open ZMdemo.JQM145.HomePage and select `View > Web Page`.

On the web page, select `...or jump directly` to expand the demo menu, and select the `Google Maps` demo. You must be online to run this demo successfully.

When running in debug mode, remember to set the debug target URL to csp/samples/ZMdemo.JQM145.HomePage.cls.

This sample application also uses the jQuery Mobile page manager and helper plugins, the HTML5 helper plugin, and Default helper plugin.

6

Helper Plugin for Highcharts

The Highcharts helper plugin provides support for the Highcharts library. This plugin is best suited for use on mobile devices. For detailed information about Highcharts, see http://www.highcharts.com/.

6.1 Adding Highcharts to the Page Class

Adding Required Highcharts Files

The files listed in this section are contained in the Highcharts 4.0.4 release and the jQuery 2.0.3 release. If they are not already installed on your system, you can get Highcharts at http://code.highcharts.com/zips/Highcharts-4.0.4.zip and jQuery at https://code.jquery.com/jquery/.

These instructions assume that the Highcharts files are in a subdirectory of `<install-dir>`/csp/broker named highcharts-4-0-4 (see "Adding Plugins to the Page Class" for details). The following file must be available:

- /js/highcharts.js

Highcharts also requires the jQuery 2.0.3 library. The following file must be in /csp/broker:

- jquery-2.0.3.min.js

Add the following filename strings to the *JSINCLUDES* parameter of your page class. Caché assumes that /csp/broker is the root directory for all relative paths:

- *JSINCLUDES* filenames:

  ```
  "jquery-2.0.3.min.js, js/highcharts.js"
  ```

  Bear in mind that files are loaded in the order that they are listed in the parameters. In this case, the jQuery library must be listed first because the Highcharts library depends on it.

Registering the Highcharts Helper Plugin

The following example demonstrates how the Chart.js helper can be registered in your **pageContents** XData block (as described in "Adding Plugins to the Page Class"). This example uses the Default page manager, but any page manager will work:

```
<mojo:mojoDefaultPageManager>
  <mojo:highCharts-4.0.4-Helper/>
</mojo:mojoDefaultPageManager>
```

The Highcharts helper plugin can be used with other page manager and helper plugins. If the Chart.js helper plugin is also used, some plugin conflicts will occur (see "Highcharts Helper Layout Objects" later in this chapter for details).

Other Options and Requirements

For more information on this plugin, see %ZEN.Mojo.Plugin.highCharts404Helper in the class reference.

Note: **Support for Highcharts version 4.0.1**

The %ZEN.Mojo.Plugin.highCharts401Helper plugin class is also available, providing support for earlier Highcharts version 4.0.1. Usage is identical to version 4.0.4 except that version 4.0.1 of highcharts.js is used, and the following element is used to register the plugin in the **pageContents** XData block:

```
<mojo:highCharts-4.0.1-Helper/>
```

6.2 Using Highcharts in a Template Class

Highcharts Layout Objects

The $chart layout object displays a chart specified by type ($barchart by default). See "Highcharts Helper Layout Objects" for a list of layout objects that display specific chart types.

Custom Highcharts Layout Object Methods

The following method is available within all appropriate layout objects:

- **$refresh()** — Re-renders the HTML for the layout object.

Highcharts Helper Plugin Methods

This helper plugin does not include any utility methods other than those inherited from %ZEN.Mojo.Plugin.baseHelperPlugin.

Direct Access to the Highcharts Library

This plugin does not provide direct access to Highcharts library functions.

Event Handling

See "Implementing onselect() for Highcharts Events" later in this chapter.

6.2.1 Highcharts Helper Layout Objects

The $chart layout object displays a chart specified by type ($barchart by default). The Highcharts helper plugin also provides layout objects for the individual chart types in the following list. This list is intended as a quick reference to the layout options offered by the Highcharts helper plugin. For complete reference information, use the Plugin Documentation app (see "Using the Plugin Documentation and Widget Reference Apps").

$areachart	$boxplotchart	$errorbarchart	$piechart	$splinechart
$arearangechart	$bubblechart	$funnelchart	$pyramidchart	$waterfallchart
$areasplinechart	$chart	$gaugechart	$scatterchart	
$areasplinerangechart	$columnchart	$heatmapchart	$serieschart	
$barchart	$columnrangechart	$linechart	$solidgaugechart	

If both the Highcharts helper and the Chart.js helper are registered in the same **pageContents** XData block, plugin conflicts should be resolved for the following objects: $chart, $barchart, $linechart, $piechart. To resolve all conflicts in favor of Highcharts, just list it before Chart.js in **pageContents**. See "Detecting and Resolving Plugin Conflicts" for more information.

6.2.2 Implementing onselect() for Highcharts Events

Every HighCharts layout object is assigned default event handlers for click events. You can change these defaults by implementing an override of the contentTemplate.**onselect()** method to handle the event (see "Event Handling" in *Using Zen Mojo*).

When a chart layout object is clicked, the plugin passes the following arguments to **onselect()**:

- *key* — the key of the selected layout object
- *value* — an array of values
- *docViewId* — the id of the documentView

The chart is driven by the *value* data object. The default *value* argument passed to **onselect()** varies according to where the layout object was clicked:

- If a point in the chart was clicked, *value* has the properties seriesIndex and dataIndex, which contain the indexes of the series and data points, respectively.

  To override this behavior, define the options.plotOptions.series.point.events property of the layout object (see http://api.highcharts.com/highcharts#plotOptions.series.point.events.click for option details).

- If the chart background was clicked (rather than a point on the chart), *value* has the properties chartX and chartY, which contain the X and Y position of the clicked spot.

  To override this behavior, define the options.chart.events.click property of the layout object (see http://api.highcharts.com/highcharts#chart.events.click for option details).

The Highcharts code in the Widget Reference contains working examples of these event overrides.

6.3 Using Highcharts Plugin Sample Code

The Widget Reference offers interactive widget samples for the Highcharts helper. The reference includes working examples, detailed descriptions, and source code that you can cut and paste into your own applications.

7

Helper Plugin for HTML5

The HTML5 helper plugin provides easy access to HTML5 standard objects. This plugin is suitable for use on both mobile devices and desktop computers.

7.1 Adding HTML5 to the Page Class

Adding Required HTML5 Files

The HTML5 helper plugin does not require any external JavaScript or CSS files.

Registering the HTML5 Helper Plugin

The following example demonstrates how the HTML5 helper can be registered in your **pageContents** XData block (as described in "Adding Plugins to the Page Class"). This example uses the Default page manager, but any page manager will work:

```
<mojo:mojoDefaultPageManager>
  <mojo:HTML5Helper/>
  <mojo:mojoDefaultHelper/>
</mojo:mojoDefaultPageManager>
```

The HTML5 helper plugin can be used with other page manager and helper plugins. The Default helper plugin is required by several HTML5 helper layout objects.

If the HTML5 helper is used together with another helper that provides extended versions of HTML5 elements (such as Bootstrap or Dojo), plugin conflicts will occur. To resolve these conflicts in favor of the extended versions, the HTML5 helper should be listed *after* all other helper plugins except the Default helper (see "Detecting and Resolving Plugin Conflicts").

Other Options and Requirements

Many HTML5 helper layout objects require the $loop layout object provided by the Default helper plugin. These layout objects will display the string `"($loop)"` in place of the expected output if the Default helper is not included.

For more information on this plugin, see %ZEN.Mojo.Plugin.HTML5Helper in the class reference.

7.2 Using HTML5 in a Template Class

HTML5 Layout Objects

See "HTML5 Helper Layout Objects" later in this chapter for a complete list of available layout objects.

Custom HTML5 Layout Object Methods

The following methods are available within all appropriate layout objects:

- **$hide()** — Hides the layout object.
- **$refresh()** — Re-renders the HTML for the layout object.
- **$setAttribute()** — Sets the value of an attribute of this layout object. This method takes two arguments. First is the name of the attribute, and second is the value.
- **$show()** — Shows the layout object.
- **$toggleVisibility()** — Hides the layout object if it is currently displayed, or displays the object if it is currently hidden.

HTML5 Helper Plugin Methods

This helper plugin does not include any utility methods other than those inherited from %ZEN.Mojo.Plugin.baseHelperPlugin.

Direct Access to the HTML5 Library

Not applicable — this plugin does not use any external library files.

Event Handling

This plugin has no special event handling requirements.

7.2.1 HTML5 Helper Layout Objects

The following list is intended as a quick reference to the layout options offered by the HTML5 helper plugin. For complete reference information, use the Plugin Documentation app (see "Using the Plugin Documentation and Widget Reference Apps"):

$a	$caption	$em	$hr	$menu	$rt	$tbody
$abbr	$cite	$embed	$i	$meter	$ruby	$td
$address	$code	$fieldset	$iframe	$nav	$s	$textarea
$area	$col	$figcaption	$img	$object	$samp	$tfoot
$article	$colgroup	$figure	$input	$ol	$section	$th
$aside	$datalist	$footer	$ins	$optgroup	$select	$thead
$audio	$dd	$form	$kbd	$option	$small	$time
$b	$del	$h1	$keygen	$output	$source	$tr
$bdi	$details	$h2	$label	$p	$span	$track
$bdo	$dfn	$h3	$legend	$param	$strong	$u

$blockquote	$dialog	$h4	$li	$pre	$sub	$ul
$br	$div	$h5	$main	$progress	$summary	$var
$button	$dl	$h6	$map	$q	$sup	$video
$canvas	$dt	$header	$mark	$rp	$table	$wbr

7.3 Using HTML5 Plugin Sample Code

A complete sample application is available for the Zen Mojo HTML5 helper plugin. In Studio, go to ZMdemo.html5 in the SAMPLES namespace. Compile the classes in that package if necessary. To run the application, open ZMdemo.html5.HomePage and select `View > Web Page`. When running in debug mode, remember to set the debug target URL to csp/samples/ZMdemo.html5.HomePage.cls.

In addition to the HTML5 helper, the sample application uses the Dojo page manager and helper plugins and the Default helper plugin.

8

Page Manager and Helper Plugins for ChocolateChip-UI

The ChocolateChip-UI page manager and helper plugins provide support for the ChocolateChip-UI framework. Because the ChocolateChip-UI framework was designed for mobile devices, these plugins are especially suitable for use on mobile devices, but could also be used on desktop computers. For detailed information about ChocolateChip-UI, see http://chocolatechip-ui.com/.

8.1 Adding ChocolateChip-UI to the Page Class

Adding Required ChocolateChip-UI Files

The files listed in this section are contained in the ChocolateChip-UI 3.5.2 release and the jQuery 2.0.3 release. If ChocolateChip-UI is not already installed on your system, you can generate the required files by building the version 3.5.2 source available at https://github.com/chocolatechipui/chocolatechip-ui/releases. The jQuery library is available at https://code.jquery.com/jquery/.

These instructions assume that the ChocolateChip-UI files are in `<install-dir>`/csp/broker (see "Adding Plugins to the Page Class" for details). The following files must be available:

- chui-3.5.2.js
- chui-ios-3.5.2.css

ChocolateChip-UI also requires the jQuery 2.0.3 library. The following file must be in /csp/broker:

- jquery-2.0.3.min.js

Add the following filename strings to the *JSINCLUDES* and *CSSINCLUDES* (or *CSS3INCLUDES*) parameters of your page class. Caché assumes that /csp/broker is the root directory for all relative paths:

- *CSSINCLUDES* filenames:

  ```
  "chui-ios-3.5.2.css"
  ```

- *JSINCLUDES* filenames:

  ```
  "jquery-2.0.3.min.js, chui-3.5.2.js"
  ```

  Bear in mind that files are loaded in the order that they are listed in the parameters. In this case, the jQuery library must be listed first because the ChocolateChip-UI library depends on it.

Registering the ChocolateChip-UI Plugins

The following example demonstrates how the ChocolateChip-UI page manager and helper can be registered in your **pageContents** XData block (as described in "Adding Plugins to the Page Class"):

```
<mojo:chui-3.5.2-PageManager>
  <mojo:chui-3.5.2-Helper/>
  <mojo:mojoDefaultHelper/>
</mojo:chui-3.5.2-PageManager>
```

Note: A page class that uses the ChocolateChip-UI page manager can include only one documentView in its **pageContents** XData block.

The ChocolateChip-UI page manager is required when you use ChocolateChip-UI helper plugin, and can also be used with most other helper plugins. The Default helper plugin is required only by the ChocolateChip-UI $list layout object.

The ChocolateChip-UI helper provides extended versions of HTML5 elements. If it is used together with the HTML5 helper, the ChocolateChip-UI helper should be listed first. Plugin conflicts are also likely when the ChocolateChip-UI helper is used with other helpers that extend HTML5 elements (see "Detecting and Resolving Plugin Conflicts").

Other Options and Requirements

The $list layout object requires the $loop layout object provided by the Default helper plugin. This layout object will display the string `"($loop)"` in place of the expected output if the Default helper is not included.

For more information on these plugins, see %ZEN.Mojo.Plugin.chui352PageManager and %ZEN.Mojo.Plugin.chui352Helper in the class reference.

Note: **Deprecated ChocolateChip-UI Plugin Classes**

The %ZEN.Mojo.Plugin.chuiPageManager and %ZEN.Mojo.Plugin.chuiHelper plugin classes are currently available for backward compatibility with earlier versions of Zen Mojo. These classes also use ChocolateChip-UI version 3.5.2, and usage is identical to the current plugins except that the older classes are registered using the following elements:

```
<mojo:chuiPageManager>
  <mojo:chuiHelper/>
</mojo:chuiPageManager>
```

These older classes are not being updated and will be removed in a later release.

8.2 Using ChocolateChip-UI in a Template Class

Important: **Required Layout Objects**

A ChocolateChip-UI layout graph must always contain at least one $article layout object with at least one $section layout object in it. This requirement is imposed by the ChocolateChip-UI framework.

ChocolateChip-UI Layout Objects

See "ChocolateChip-UI Helper Layout Objects" later in this chapter for a complete list of available layout objects.

Custom ChocolateChip-UI Layout Object Methods

The following method is available within all appropriate layout objects:

- **$refresh()** — Re-renders the HTML for the layout object.

ChocolateChip-UI Helper Plugin Methods

The ChocolateChip-UI helper plugin does not include any utility methods other than those inherited from %ZEN.Mojo.Plugin.baseHelperPlugin.

Direct Access to ChocolateChip-UI Libraries

When you use the ChocolateChip-UI plugins, your client methods can access variables defined in the ChocolateChip-UI JavaScript files. These variables abstract the user input interaction from the device input. For details, see http://chocolatechip-ui.com/documentation#/chuijs.

Event Handling

ChocolateChip-UI plugins have no special event handling requirements.

8.2.1 ChocolateChip-UI Helper Layout Objects

The following list is intended as a quick reference to the layout options offered by the ChocolateChip-UI helper plugin. For complete reference information, use the Plugin Documentation app (see "Using the Plugin Documentation and Widget Reference Apps"):

$article	$div	$h3	$link	$navbar
$aside	$h1	$h4	$list	$p
$button	$h2	$h5	$listitem	$section

Since the ChocolateChip-UI helper provides extended versions of HTML5 elements, plugin conflicts are likely when it is used with other helpers that extend HTML5 elements (see "Detecting and Resolving Plugin Conflicts" earlier in this book).

8.3 Using ChocolateChip-UI Plugin Sample Code

A complete sample application is available for the ChocolateChip-UI page manager and helper plugins. In Studio, go to ZMdemo.chui in the SAMPLES namespace. Compile the classes in that package if necessary. To run the application, open ZMdemo.chui.HomePage and select `View > Web Page`. When running in debug mode, remember to set the debug target URL to csp/samples/ZMdemo.chui.HomePage.cls.

In addition to the ChocolateChip-UI page manager and helper plugins, the sample application also uses the HTML5 and Default helper plugins.

9

Page Manager and Helper Plugins for Dojo

The Dojo page manager plugin and related helper plugins provide support for the Dojo Toolkit. Helper plugins include *Dojo Dijit* helper for the Dijit user interface framework, *Dojo 2D Chart* helper for dojox/charting/Chart2D, and *Dojo GridX* helper for the separately packaged GridX library. These plugins are best suited for use on desktop computers. For detailed information about the Dojo Toolkit, see https://dojotoolkit.org/.

9.1 Adding Dojo to the Page Class

Adding Required Dojo Files

The files listed in this section are in either Dojo Toolkit 1.9.1 or GridX 1.3.0. If they are not already installed on your system, you can download Dojo at http://download.dojotoolkit.org/release-1.9.1/ and GridX at https://github.com/oria/gridx/releases.

These instructions assume that the files are in a subdirectory of `<install-dir>`/csp/broker named dojo-release-1-9-1 (see "Adding Plugins to the Page Class" for details). The following files must be available:

- /dojo/dojo.js
- /app/dojo_2DChart.js
- /app/dojo_191Dijit.js
- /dijit/themes/claro/claro.css (or another theming style sheet)
- /gridx/resources/claro/Gridx.css (required only if you are using the GridX plugin)

Add the following filename strings to the *JSINCLUDES* and *CSSINCLUDES* (or *CSS3INCLUDES*) parameters of your page class. Caché assumes that /csp/broker is the root directory for all relative paths:

- *CSSINCLUDES* filenames:

```
"dojo-release-1-9-1/dijit/themes/claro/claro.css,
dojo-release-1-9-1/gridx/resources/claro/Gridx.css"
```

- *JSINCLUDES* filenames:

```
"dojo-release-1-9-1/dojo/dojo.js, dojo-release-1-9-1/app/dojo_2DChart.js,
dojo-release-1-9-1/app/dojo_191Dijit.js"
```

Registering the Dojo Plugins

The following example demonstrates how the Dojo page manager and one or more Dojo helpers can be registered in your **pageContents** XData block (as described in "Adding Plugins to the Page Class"). Each Dojo helper plugin is independent of the others, so register only the Dojo helpers you intend to use:

```
<mojo:dojo-1.9.1-PageManager>
  <mojo:dojo-1.9.1-DijitHelper/>
  <mojo:dojo-1.9.1-2DChartHelper/>
  <mojo:dojoGridX-1.3.0-Helper/>
  <mojo:mojoDefaultHelper/>
</mojo:dojo-1.9.1-PageManager>
```

The Dojo page manager is required if you use Dojo helper plugins, and can also be used with most other helper plugins. The Default helper plugin is required only by some layout objects in the Dojo Dijit helper.

The Dojo Dijit helper provides extended versions of HTML5 elements. If it is used together with the HTML5 helper, the Dojo Dijit helper should be listed first. Plugin conflicts are also likely when the Dojo Dijit helper is used with other helpers that extend HTML5 elements (see "Detecting and Resolving Plugin Conflicts").

Note: A page class that uses the Dojo page manager can include more than one documentView in its **pageContents** XData block. For a working example, see the HTML5 sample program ("Using HTML5 Helper Sample Code").

Other Options and Requirements

- Your page class must implement page method **adjustContentSize()**, event handler **onlayoutHandler()**, and an XData **Style** block, as described in "Additional Dojo Page Manager Requirements" later in this chapter.
- Dojo Dijit helper layout objects $AccordionContainer, $BorderContainer, $ContentPane, $DojoxContentPane, and $LayoutContainer require the $loop layout object provided by the Default helper plugin. These layout objects will display the string `"($loop)"` in place of the expected output if the Default helper is not included.

The page manager plugin has several methods and properties not discussed here. See %ZEN.Mojo.Plugin.dojo191PageManager in the Caché class reference for details.

For more information on the helper plugins, see %ZEN.Mojo.Plugin.dojo1912DChartHelper, %ZEN.Mojo.Plugin.dojo191DijitHelper, and %ZEN.Mojo.Plugin.dojoGridX130Helper in the class reference.

Note: **Deprecated Dojo Plugin Classes**

The %ZEN.Mojo.Plugin.dojoPageManager, %ZEN.Mojo.Plugin.dojo2DChartHelper, %ZEN.Mojo.Plugin.dojoDijitHelper, and %ZEN.Mojo.Plugin.dojoGridXHelper plugin classes are currently available for backward compatibility with earlier versions of Zen Mojo. These classes also use Dojo version 1.9.1, and usage is identical to the current plugins except that the older classes are registered using the following elements:

```
<mojo:dojoPageManager>
  <mojo:dojoDijitHelper/>
  <mojo:dojo2DChartHelper/>
  <mojo:dojoGridXHelper/>
</mojo:dojoPageManager>
```

These older classes are not being updated and will be removed in a later release.

9.1.1 Additional Dojo Page Manager Requirements

In order to use the Dojo page manager plugin, your page class must implement overrides for page method **adjustContentSize()** and event handler **onlayoutHandler()**, and add an XData **Style** block. If you omit any of these items, the web page will not display your contents.

- Override for basePage.**adjustContentSize()**

This method must specify the width, height, and position of each documentView. For a details and examples, see "Overriding the adjustContentSize() Method" in the "Working with Plugins" chapter.

- Override for basePage.**onlayoutHandler()**

  This method must first invoke the inherited **onlayoutHandler()**, and then resize the top-level layout object. In the following example, `'layoutContainer-1` is the top-level layout object within the layout graph used by the page:

```
ClientMethod onlayoutHandler() [ Language = javascript ]
{
  this.invokeSuper('onlayoutHandler',arguments);
  var topContainer = zen('leftView').getItemByKey('layoutContainer-1');
  if (topContainer) {
    topContainer.$dojoItem.resize();
  }
}
```

  (For details on `$dojoItem`, see the section on "Direct Access to the Dojo Libraries" in "Using Dojo in a Template Class").

- Add an XData **Style** block

  This style block must define the `dvDocument` class with height as 100%. For example:

```
XData Style {
<style type="text/css">
  .dvDocument { height: 100%;}
</style>
}
```

9.2 Using Dojo in a Template Class

The following statements apply to all three Dojo helper plugins.

Dojo Layout Objects

See "Dojo Helper Layout Objects" later in this chapter for a complete list of available layout objects.

Custom Dojo Layout Object Methods

The following methods are available within all appropriate layout objects:

- **$hide()** — Hides the layout object.
- **$refresh()** — Re-renders the HTML for the layout object.
- **$show()** — Shows the layout object.

Dojo Helper Plugin Methods

Each of these plugins has several utility methods. See %ZEN.Mojo.Plugin.dojo1912DChartHelper, %ZEN.Mojo.Plugin.dojo191DijitHelper, and %ZEN.Mojo.Plugin.dojoGridX130Helper in the class reference for details.

Direct Access to the Dojo Libraries

The Dojo helper plugins provide direct access to the Dojo Toolkit libraries via the `$dojoItem` property of each layout object. You can use the documentView.**getItemByKey()** method to return a layout object, and then use the `$dojoItem` property of that layout object to call a library function. For example:

```
myDocView.getItemByKey('myKey').$dojoItem.show();
```

(For an example using **getItemByKey()**, see "Tutorial 3: Event Handling" in *Using Zen Mojo*).

Event Handling

Dojo plugins have no special event handling requirements.

9.2.1 Dojo Helper Layout Objects

The following lists are intended as a quick reference to the layout options offered by the Dojo helper plugins. For complete reference information, use the Plugin Documentation app (see "Using the Plugin Documentation and Widget Reference Apps"):

dojo2DChart

- $Chart — represents a 2D chart.
- $ChartLegend — represents a chart legend.

dojoGridX

- $Gridx — represents a Dojo GridX. A **$refresh** call will update the body of the table with the current data of the associated store.

dojoDijit

Since the Dojo Dijit helper provides extended versions of HTML5 elements, plugin conflicts are likely when the it is used with other helpers that extend HTML5 elements (see "Detecting and Resolving Plugin Conflicts" earlier in this book).

$AccordionContainer	$DropDownMenu	$MenuSeparator	$TextBox
$BorderContainer	$Editor	$MultiSelect	$Textarea
$Button	$Fieldset	$NumberSpinner	$TimeTextBox
$CheckBox	$FilteringSelect	$NumberTextBox	$TitlePane
$CheckedMenuItem	$Form	$PopupMenuBarItem	$ToggleButton
$ColorPalette	$HorizontalRule	$PopupMenuItem	$Toolbar
$ComboBox	$HorizontalRuleLabels	$ProgressBar	$ToolbarSeparator
$ComboButton	$HorizontalSlider	$RadioButton	$Tooltip
$ContentPane	$InlineEditBox	$RadioMenuItem	$TooltipDialog
$CurrencyTextBox	$LayoutContainer	$Select	$Tree
$DataList	$LinkPane	$SimpleTextarea	$TreeObjectStoreModel
$DateTextBox	$Menu	$SplitContainer	$ValidationTextBox
$Dialog	$MenuBar	$StackContainer	$VerticalRule
$DojoxContentPane	$MenuBarItem	$StoreMemory	$VerticalRuleLabels
$DropDownButton	$MenuItem	$TabContainer	$VerticalSlider

9.3 Using Dojo Plugin Sample Code

A sample application is available for the Dojo page manager and helper plugins. In Studio, go to ZMdemo.dojo in the SAMPLES namespace, and compile the classes in that package if necessary. To run the application, open ZMdemo.dojo.HomePage and select `View > Web Page`. When running in debug mode, remember to set the debug target URL to csp/samples/ZMdemo.dojo.HomePage.cls.

In addition to the Dojo page manager and helper plugins, the sample application also uses the HTML5 and Default helper plugins.

The Dojo page manager and helper plugins are also used in the HTML5 sample program (see "Using HTML5 Helper Sample Code"), which demonstrates how to implement two different instances of documentView.

10

Page Manager and Helper Plugins for jQuery Mobile

The jQuery Mobile page manager plugin and jQuery Mobile helper plugin enable you to use the jQuery Mobile framework. The jQuery Mobile framework is designed for mobile devices, but can also be used on desktop computers. For detailed information about jQuery Mobile, see http://jquerymobile.com/.

10.1 Adding jQuery Mobile to the Page Class

Adding Required jQuery Mobile Files

The files listed in this section are contained in jQuery Mobile version 1.4.5 and jQuery (core) version 1.10.2. If they are not already installed on your system, you can get jQuery Mobile at https://jquerymobile.com/download/all/ and jQuery at https://code.jquery.com/jquery/.

These instructions assume that the jQuery and jQuery Mobile files are in `<install-dir>`/csp/broker (see "Adding Plugins to the Page Class" for details). The following files must be available:

- jquery.mobile-1.4.5.min.js
- jquery.mobile-1.4.5.min.css
- jquery-1.10.2.min.js

Note: Plugins are also available for versions 1.4.3 and 1.3.2 of jQuery Mobile. See "Older jQuery Mobile Classes" for details.

Add the following filename strings to the *JSINCLUDES* and *CSSINCLUDES* (or *CSS3INCLUDES*) parameters of your page class. Caché assumes that /csp/broker is the root directory for all relative paths:

- *CSSINCLUDES* filenames:

  ```
  "jquery.mobile-1.4.5.min.css"
  ```

- *JSINCLUDES* filenames:

  ```
  "jquery-1.10.2.min.js, jquery.mobile-1.4.5.min.js"
  ```

Bear in mind that files are loaded in the order that they are listed in the parameters. In this case, the jQuery library must be listed first because the jQuery Mobile library depends on it.

Registering the jQuery Mobile Plugins

The following example demonstrates how the jQuery Mobile page manager and helper can be registered in your **pageContents** XData block (as described in "Adding Plugins to the Page Class"):

```
<mojo:jQM-1.4.5-PageManager>
  <mojo:jQM-1.4.5-Helper/>
  <mojo:mojoDefaultHelper/>
</mojo:jQM-1.4.5-PageManager>
```

Page manager and helper elements can both take optional attributes (see "Support for jQuery Mobile Animation and Themes").

Note: A page class that uses the jQuery Mobile page manager can include only one documentView in its **pageContents** XData block.

The jQuery Mobile page manager is required when you use jQuery Mobile helper plugin, and can also be used with most other helper plugins. The Default helper plugin is required by several jQuery Mobile layout objects.

The jQuery Mobile helper provides extended versions of HTML5 elements. If it is used together with the HTML5 helper, the jQuery Mobile helper should be listed first. Plugin conflicts are also likely when the jQuery Mobile helper is used with other helpers that extend HTML5 elements (see "Detecting and Resolving Plugin Conflicts").

Other Options and Requirements

- jQuery Mobile helper layout objects $collapsible, $collapsibleset, $navbar, and $listview require the $loop layout object provided by the Default helper plugin. These layout objects will display the string `"($loop)"` in place of the expected output if the Default helper is not included.
- In the **pageContents** XData block, both `<mojo:jQM-1.4.5-PageManager>` and `<mojo:jQM-1.4.5-Helper/>` elements can take optional attributes. For details, see "Support for jQuery Mobile Animation and Themes" later in this chapter.

For more information on these plugins, see %ZEN.Mojo.Plugin.jQM145PageManager and %ZEN.Mojo.Plugin.jQM145Helper in the class reference.

10.1.1 Older jQuery Mobile Classes

Support for jQuery Mobile version 1.4.3 is provided by the %ZEN.Mojo.Plugin.jQM143PageManager and %ZEN.Mojo.Plugin.jQM143Helper plugin classes . Usage is identical to version 1.4.5 except that jquery.mobile-1.4.3.min.js and jquery.mobile-1.4.3.min.css are used, and the elements shown in the following example are used in the **pageContents** XData block:

```
<mojo:jQM-1.4.3-PageManager>
  <mojo:jQM-1.4.3-Helper/>
  <mojo:mojoDefaultHelper/>
</mojo:jQM-1.4.3-PageManager>
```

Support for jQuery Mobile version 1.3.2 is provided by the %ZEN.Mojo.Plugin.jQM132PageManager and %ZEN.Mojo.Plugin.jQM132Helper plugin classes. Usage is identical to version 1.4.5 except that jquery.mobile-1.3.2.min.js and jquery.mobile-1.3.2.min.css are used, and the elements shown in the following example are used in the **pageContents** XData block:

```
<mojo:jQM-1.3.2-PageManager>
  <mojo:jQM-1.3.2-Helper/>
  <mojo:mojoDefaultHelper/>
</mojo:jQM-1.3.2-PageManager>
```

Note: **Deprecated jQuery Mobile Plugins**

The jQuery Mobile %ZEN.Mojo.Plugin.jQMPageManager and %ZEN.Mojo.Plugin.jQMHelper plugin classes are currently available for backward compatibility with earlier versions of Zen Mojo. These classes also use jQuery Mobile version 1.3.2, and usage is identical to the current 1.3.2 plugins except that the older classes are registered using the following elements:

```
<mojo:jQMPageManager>
  <mojo:jQMHelper/>
  <mojo:mojoDefaultHelper/>
</mojo:jQMPageManager>
```

These older classes are not being updated and will be removed in a later release.

10.1.2 Support for jQuery Mobile Animation and Themes

In the **pageContents** XData block, both page manager and helper elements support optional attributes related to themes and animation. The following example uses all five attributes:

```
<mojo:jQM-1.4.5-PageManager jQueryTheme="a" jQueryAnimation="flip">
  <mojo:jQM-1.4.5-Helper jQueryHeaderTheme="a" jQueryBarTheme="d" jQueryCollapsibleTheme="c"/>
</mojo:jQM-1.4.5-PageManager>
```

Optional jQuery Mobile Page Manager Attributes

You can specify the following attributes for the page manager element:

- `jQueryTheme`

  Specifies the jQuery theme for the documentView. For example:

  ```
  <mojo:jQM-1.4.5-PageManager jQueryTheme="a">
  ```

- `jQueryAnimation`

  Specifies the jQuery Mobile animation to use when modifying the document stack (pushing new documents onto the stack or popping documents off the stack). For example:

  ```
  <mojo:jQM-1.4.5-PageManager jQueryAnimation="flip">
  ```

Optional jQuery Mobile Helper Attributes

You can specify the following attributes of for the helper element:

- `jQueryHeaderTheme`

  Specifies the jQuery theme for the header and footer.

- `jQueryBarTheme`

  Specifies the jQuery theme for grids and bars.

- `jQueryCollapsibleTheme`

  Specifies the jQuery theme for collapsible sets.

Any or all of the helper attributes can be used in the same element. For example:

```
<mojo:jQM-1.4.5-Helper jQueryHeaderTheme="a" jQueryBarTheme="d" jQueryCollapsibleTheme="c"/>
```

If you need to change these attributes after initial setup, use the **changeTheme()** helper plugin method (described later in "Custom jQuery Mobile Helper Plugin Methods").

10.2 Using jQuery Mobile in a Template Class

jQuery Mobile Layout Objects

See "jQuery Mobile Helper Layout Objects" later in this chapter for a complete list of available layout objects.

Custom jQuery Mobile Layout Object Methods

The following methods are available within all appropriate layout objects:

- **$hide()** — Hides the layout object.
- **$refresh()** — Re-renders the HTML for the layout object.
- **$show()** — Shows the layout object.

jQuery Mobile Helper Plugin Methods

See "Custom jQuery Mobile Helper Plugin Methods" later in this chapter for a list of jQuery Mobile utility methods.

Direct Access to the jQuery Mobile Framework

The jQuery Mobile plugins provide direct access to the jQuery Mobile framework via the `$.mobile` object, which your client methods can use to refer to jQuery Mobile methods. For example:

```
$.mobile.loading('show') ;
```

Event Handling

jQuery Mobile plugins have no special event handling requirements.

10.2.1 jQuery Mobile Helper Layout Objects

The following list is intended as a quick reference to the layout options offered by the jQuery Mobile helper plugin. For complete reference information, use the Plugin Documentation app (see "Using the Plugin Documentation and Widget Reference Apps"):

$button	$controlgroup	$input	$navbaritem	$radio-button
$checkbox	$fieldset	$link	$pagecontent	$select
$collapsible	$footer	$listview	$panel	$string
$collapsibleset	$grid	$listviewitem	$password	$text
$control	$header	$navbar	$popup	$textarea

Since the jQuery Mobile helper provides extended versions of HTML5 elements, plugin conflicts are likely when it is used with other helpers that extend HTML5 elements (see "Detecting and Resolving Plugin Conflicts" earlier in this book).

10.2.2 Custom jQuery Mobile Helper Plugin Methods

In addition to methods inherited from %ZEN.Mojo.Plugin.baseHelperPlugin, the jQuery Mobile helper plugin provides the following methods:

- **changeTheme()**

```
ClientMethod changeTheme(newTheme = "c", headerTheme = "a", barTheme = "a")
[ Language = javascript ]
```

Given a theme, a header theme, and a bar theme, this method updates the themes used in this page. For example:

```
var mainView = zen('mainView');
mainView.getPluginByName('jQM-1.4.5-Helper').changeTheme('d','b','b');
```

The themes are originally set as attributes of the helper element in the **pageContents** XData block (see "Support for jQuery Mobile Animation and Themes" earlier in this chapter).

- **openPanel()**

```
ClientMethod openPanel(id) [ Language = javascript ]
```

Given the id of a $panel object, this method displays that panel. For example:

```
var mainView = zen('mainView');
getPluginByLayoutObjectType($panel).openPanel('leftPanel');
```

- **closePanel()**

```
ClientMethod closePanel(id) [ Language = javascript ]
```

Given the id of a $panel object, this method closes that panel.

- **togglePanel()**

```
ClientMethod togglePanel(id) [ Language = javascript ]
```

Given the id of a $panel object, this method toggles the panel open or closed.

- **showPopup()**

```
ClientMethod showPopup(key, options) [ Language = javascript ]
```

Given the key of a $popup object, this method displays that object. The *options* argument is an object that contains any options for the popup. For example:

```
var view = zen('mainView');
view.getPluginByLayoutObjectType($popup').showPopup('form-popup',{x:150,y:200});
```

- **closePopup()**

```
ClientMethod closePopup(key) [ Language = javascript ]
```

Given the key of a $popup object, this method closes that object.

10.3 Using jQuery Mobile Plugin Sample Code

The Widget Reference offers interactive widget samples for the jQuery Mobile helper plugin. The reference includes working examples, detailed descriptions, and source code that you can cut and paste into your own applications.

A complete sample application is also available for each version the jQuery Mobile page manager and helper plugins. For the jQuery Mobile 1.4.5 version, open Studio and go to ZMdemo.JQM145 in the SAMPLES namespace. Compile the classes in that package if necessary. To run the application, open ZMdemo.JQM145.HomePage and select `View > Web Page`. When running in debug mode, remember to set the debug target URL to csp/samples/ZMdemo.JQM145.HomePage.cls.

For the other jQuery Mobile versions, follow the same directions but specify ZMdemo.JQM143 or ZMdemo.JQM (for version 1.3.2).

In addition to the jQuery Mobile page manager and helper plugins, these sample applications also use the Google Maps, HTML5, and Default helper plugins.

11

Creating Plugins

This chapter describes the basic requirements to follow if you create your own plugin classes. It discusses the following topics:

- Creating a Page Manager Plugin
- Creating a Helper Plugin
- Creating Plugin Documentation

11.1 Creating a Page Manager Plugin

To create a page manager plugin, your page class must meet the following requirements:

- It must extend %ZEN.Mojo.Plugin.basePageManager.
- It must specify a logical name. To do this, include the pluginName property and specify its InitialExpression. By convention, specify the same name as used in the XData block (see the next item).
- It can specify the *XMLNAME* parameter. This is useful if you want to include the version number (with dashes and period characters) in the name of the plugin, as used in the XData block. If *XMLNAME* is not specified, the name of the plugin defaults to the short class name.
- It can specify a version. To do this, include the version property and specify its InitialExpression.
- It must implement the **onCheckLibraries()**, **afterRenderDocument()**, and **afterPopDocument()** methods (see the following section for details).

11.1.1 Required Page Manager Methods

The following methods of the page class are required by any page manager plugin.

onCheckLibraries()

```
ClientMethod onCheckLibraries() [ Language = javascript ]
```

Alerts the user if the needed libraries are not available. Zen Mojo invokes this method when it displays a Zen Mojo page that uses this page manager class. This method should perform a suitable check on whether the libraries needed by this plugin class are available to the page class. If the libraries are not available, the method should use **alert()** to display a warning to the user.

For example:

```
ClientMethod onCheckLibraries() [ Language = javascript ]
{
  if (typeof $ === 'undefined') {
    alert('jQuery library is not loaded correctly. Check your includes.');
    return false;
  } else if (typeof $.UIGoToArticle === 'undefined') {
    alert('Chocolate Chip UI library is not loaded correctly. Check your includes.');
    return false;
  }
  return true;
}
```

If no libraries are needed, the method can simply return true.

afterRenderDocument()

```
ClientMethod afterRenderDocument(docView, displayMode, html) [ Language = javascript ]
```

where *docView* is the id of the documentView instance, *displayMode* is the current display mode of the instance, and *html* is the HTML generated for this instance. *displayMode* can be `'layout'`, `'data'`, `'html'`, or `'iframe'`

This method controls how Zen Mojo defines the DOM (Document Object Model) used by the browser. The sequence of events is as follows:

1. The page is requested.
2. Zen Mojo generates the HTML needed for each documentView.
3. If the property suppressRender is false, Zen Mojo inserts the HTML into the DOM.

   Otherwise, Zen Mojo skips this step.
4. Zen Mojo calls **afterRenderDocument()**.

If this method does not need to do anything, it can simply return null.

afterPopDocument()

```
ClientMethod afterPopDocument(docView, render) [ Language = javascript ]
```

Controls how the page manager handles transitions when **popDocument()** is called. The sequence of events is as follows:

1. A client method calls **popDocument()**.
2. Zen Mojo renders the new document as described in the **afterRenderDocument()** entry.
3. Zen Mojo calls **afterPopDocument()**.

If this method does not need to do anything, it can simply return null.

11.2 Creating a Helper Plugin

To create a helper plugin, your helper class must meet the following requirements:

- It must extend %ZEN.Mojo.Plugin.baseHelperPlugin.
- It must specify a logical name. To do this, include the pluginName property and specify its InitialExpression. By convention, specify the same name as used in the XData block (see the next item).

- It can specify the *XMLNAME* parameter. This is useful if you want to include the version number (with dashes and period characters) in the name of the plugin, as used in the XData block. If *XMLNAME* is not specified, the name of the plugin defaults to the short class name.
- (Recommended) It can specify a version. To do this, include the version property and specify its InitialExpression.
- It must implement the **onCheckLibraries()**, **getFeatures()**, and **createLayoutObjects()** methods (see the following section for details).

11.2.1 Required Helper Plugin Methods

The following methods of the helper class are required by any helper plugin.

onCheckLibraries()

```
ClientMethod onCheckLibraries() [ Language = javascript ]
```

Alerts the user if the needed libraries are not available. Zen Mojo invokes this method when it displays a Zen Mojo page that uses this helper plugin class. This method should perform a suitable check on whether the libraries needed by this plugin class have been loaded (see the example for the page class version of **onCheckLibraries()**, shown previously in "Required Page Manager Methods"). If the libraries are not available, the method should use **alert()** to display a warning to the user..

If no libraries are needed, the method can simply return true.

getFeatures()

```
ClientMethod getFeatures() [ Language = javascript ]
```

Returns an array of objects, with one object for each layout element supported by this helper plugin. Each object in this array must include the `identifier` property, whose value must be the logical name of the layout element. Zen Mojo uses this information internally to determine how to dispatch rendering requests.

createLayoutObjects()

```
ClientMethod createLayoutObjects(type, instance) [ Language = javascript ]
```

Generates the HTML for each layout object supported by this helper plugin. Zen Mojo calls this method once for each layout object in the layout graph. The arguments are *type* (the type of layout object) and *instance* (the layout object itself).

This method should define *instance*`.$render` for each possible type, and *instance*`.$render` should return the HTML that is appropriate for that type.

If you want to expose a way to retrieve specific generated HTML elements, create a unique id for those element. To generate a unique id, use the Zen Mojo **$makeId()** function, which generates an unique id in all scenarios, even if you are using the document stack. The following shows an example:

```
html.push('<h1 id="'+this.$makeId('caption')+'" class="'+captionClass+'"
style="'+zenGet(this.captionStyle)+'">');
```

Within the logic that defines the HTML for a given layout object, be sure to pass a different string value to this function for each HTML element that must have an id.

11.3 Creating Plugin Documentation

To create plugin documentation for a page manager, specify comments for that class (with three slashes at the start of each line). The Plugin Documentation app automatically displays these comments (see "Using the Plugin Documentation and Widget Reference Apps").

To create plugin documentation for a helper plugin, define an associated class as follows:

- It must extend %ZEN.Mojo.Plugin.baseHelperDocumentation
- Its name must be *fullPluginClassName*`Documentation`
- It must implement the **getDocumentation()** method as follows:

  ```
  ClientMethod getDocumentation(identifier) [ Language = javascript ]
  ```

  Where *identifier* is the type of a layout object provided by the helper plugin. For each possible *identifier*, this method should return an object with the following properties:

  – description — Short description of the layout object.

  – attributes — Array of objects that describe the attributes supported for this layout object. Each object in this array should specify the following properties:

    - name — Name of the attribute
    - type — Type of the attribute (valid types are `string`, `number`, `boolean`, `date`, `object`, `array`, or `function`)
    - description — Description of the attribute

  For example:

  ```
  { description: 'Description of the identifier e.g. $loop',
    attributes: [ {
      name:'value',
      type:'string',
      description:'Holds the value of the html element'
    } ]
  }
  ```

12

Using the Plugin Documentation and Widget Reference Apps

Zen Mojo includes two applications that provide documentation resources:

- Plugin Documentation — provides reference information for the plugins, including details on the layout objects provided by each helper plugin.
- Widget Reference — provides an online reference system for widgets from Bootstrap, jQuery Mobile, Chart.js, and HighCharts. Each widget sample includes a description, the rendered widget or widgets, and a button to display source code.

12.1 Plugin Documentation

The Plugin Documentation application provides reference information for all plugins, including detailed information on all layout objects provided by each helper plugin. It uses the Dojo plugin and requires the Dojo library files. This application consists of a set of classes in the %ZEN.Mojo.PluginDocumentation package.

12.1.1 Using the Plugin Documentation App

To access the Plugin Documentation, use a URL of the following form:

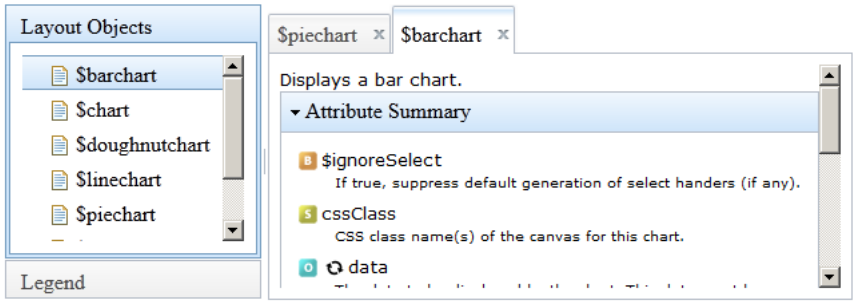
`http://`*host*`:`*port*`/csp/sys/%25ZEN.Mojo.PluginDocumentation.HomePage.cls`

where *host* is the name of the host on which Caché is running and *port* is the port that Caché is using.

For example, the following URL would be used by for instance of Caché running on the local machine and using the Caché default port number:

```
http://localhost:57772/csp/sys/%25ZEN.Mojo.PluginDocumentation.HomePage.cls
```

On startup, the app displays lists of helper plugins and page manager plugins. When you select a plugin, a new list displays layout objects for that plugin. When you select a layout object, a tab is created on the right, containing a list of all available attributes for that layout object:



If you click on the bar labeled `Legend` (lower left), it expands to display an explanation for the colored icons in front of each attribute (for example, is a Boolean, is a String, and so forth).

12.1.2 Installing Plugin Documentation from a Kit

If you are using Caché or Ensemble 2015.1 or higher, this application is installed automatically. If you are using the Zen Mojo kit, install this application as follows:

1. Make the CACHELIB database read/write.
2. The Plugin Documentation classes are in the %ZEN.Mojo.PluginDocumentation package. Load the class files into the %SYS namespace and compile them.
3. Copy the Dojo library files into the directory *install-dir*/csp/broker, so that you have a directory named *install-dir*/csp/broker/dojo-release-1-9-1.
4. Copy the folder zenmojo-images into *install-dir*/csp/broker, so that you have a directory named *install-dir*/csp/broker/zenmojo-images.

12.2 Widget Reference

Zen Mojo also provides an online reference system for widgets, which demonstrates each widget with different configurations. The widget reference currently includes samples for jQuery Mobile, Chart.js, HighCharts, and Bootstrap.

The purpose of the widget reference is to provide code samples in a copy-and-paste format and to reduce the learning curve for Zen Mojo. Each sample includes a description, a display that renders the widget, and a button to display source code for the layout object, sample data, and CSS styles. This application consists of a set of classes in the %ZEN.Mojo.WidgetReference package.

12.2.1 Using the Widget Reference App

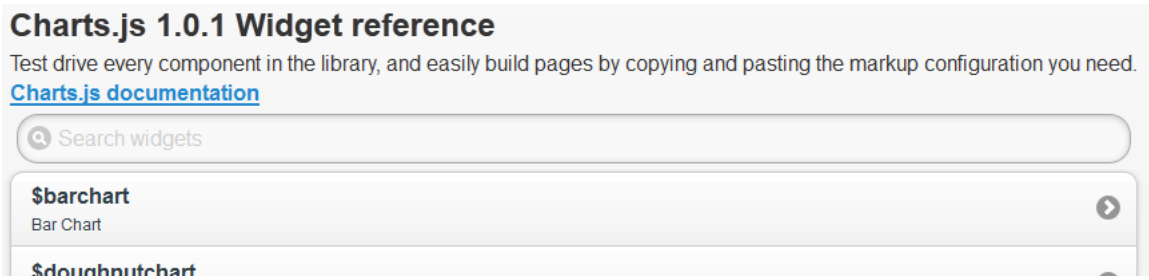
To access the Widget Reference, use a URL of the following form:

`http://`*host*`:`*port*`/csp/sys/%25ZEN.Mojo.WidgetReference.Home.cls`

where *host* is the name of the host on which Caché is running and *port* is the port that Caché is using. For example, the following URL would be used for an instance of Caché running on the local machine and using the Caché default port number:

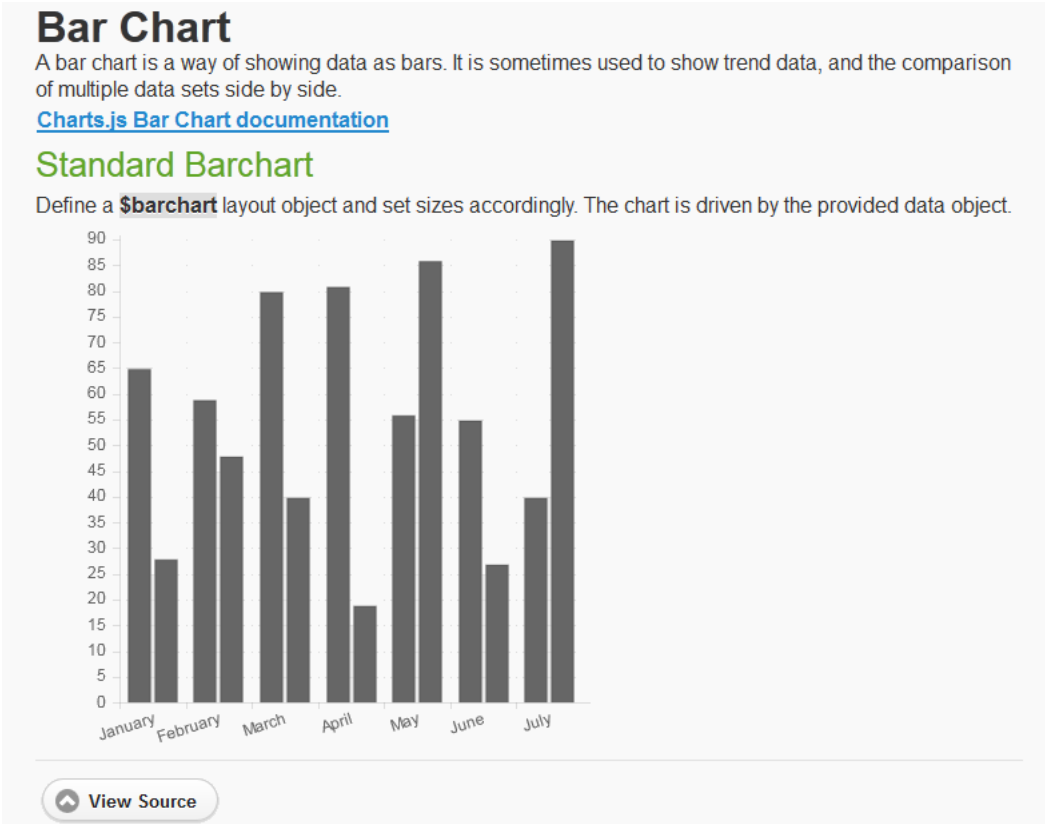
```
http://localhost:57772/csp/sys/%25ZEN.Mojo.WidgetReference.Home.cls
```

On startup, the app displays a list of plugins for which a widget reference is available. When you select a plugin, a list of layout objects is displayed for that plugin:



Each widget reference also displays a link to the official documentation for that library (for example, the `Charts.js documentation` link shown above would take you to http://www.chartjs.org/docs/).

When a layout object is selected, one or more working examples will be displayed:



When you click the `View Source` button, a popup window displays code for the layout and data, which can be copied and pasted into your own applications: