Widget SDK v1.0

Introduction

This document describes the functionality available for use in building widgets as of CM1 version 2.4. It is only intended for use by Percussion widget developers and partners with which we will work closely, and is not intended for public use. This document is a work in progress and the capabilities of the SDK and what is described in this document will evolve over time.

Current widgets shipping with the system may be using functionality that is not currently considered safe to use in developing widgets that will not be part of the shipping product – only functionality listed as available in this doc is safe for use, so be careful when starting by copying existing widgets or using them as a guide. This restriction includes (but is not limited to) JEXL functions, JavaScript libraries, Velocity macros, images, CSS, content types, content editor controls, and shared fields.

When viewing this document in MS Word, use Ctrl-Click to follow links, and the "Web" toolbar's "Back" button to navigate back to the original location in the document.

Widget Overview

A widget can be considered a 'black box' that has inputs and outputs and operates in different contexts: editing, configuring and rendering (in different contexts as well.)

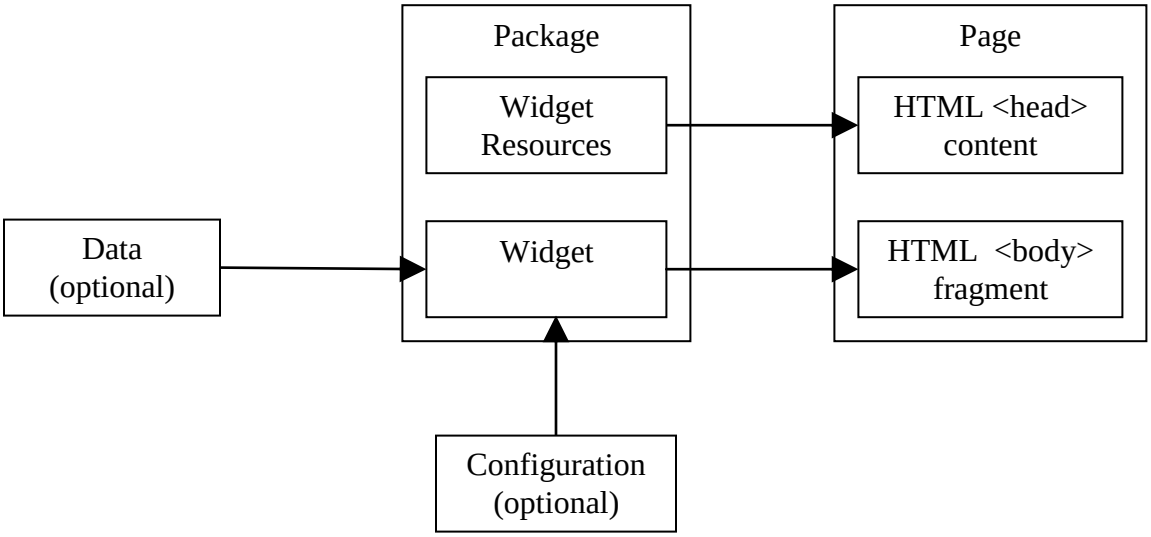


Figure 1
Representation of widget with its inputs and outputs.

The **Data** input may come from an asset associated with the widget, content retrieved from the CM1 repository (as an asset) or external source (e.g. DTS) based on some query specified in the asset or from a hard-coded external source. For a CM1 asset, the widget provides the type definition and the system creates the editor for the content.
Configuration may be associated with each instance, or with the widget itself. The latter could be, for example, a configuration file that is changeable by the admin. For instance configuration, the

widget specifies the properties and their type and the framework renders an editor. The values of these properties are available at assembly time.

Finally, the output of the widget is usually a fragment of HTML that is rendered within the <body> element. The widget resource definition controls the inclusion of css and js links in the head (or possibly other parts of the document.) These files may link to other resources as well such as images within the css.

The widget itself is composed of private data definition, configuration definition, and code. The code is composed of JEXL, velocity, JS and CSS. The is also implicit code included thru the resource definition for JS and CSS.

Widget files

Widgets are comprised of several files. This section defines each of the files and their use. Here is a depiction of the file structure:

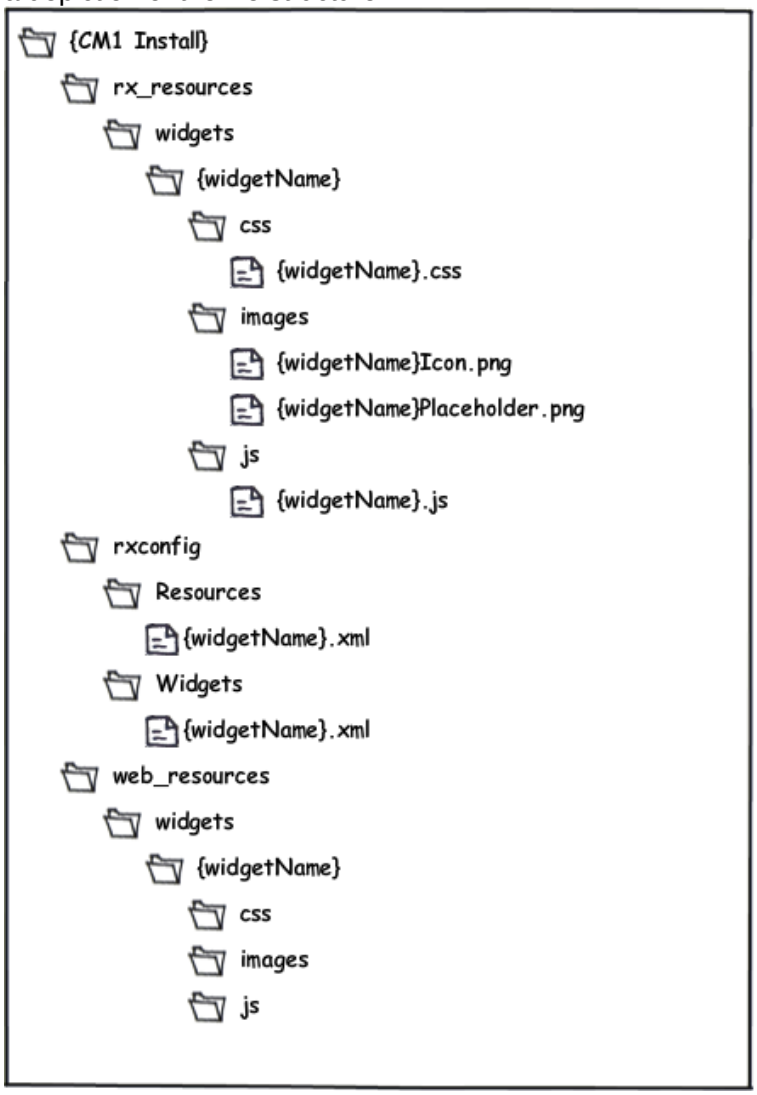


Figure 2 – Widget File Structure

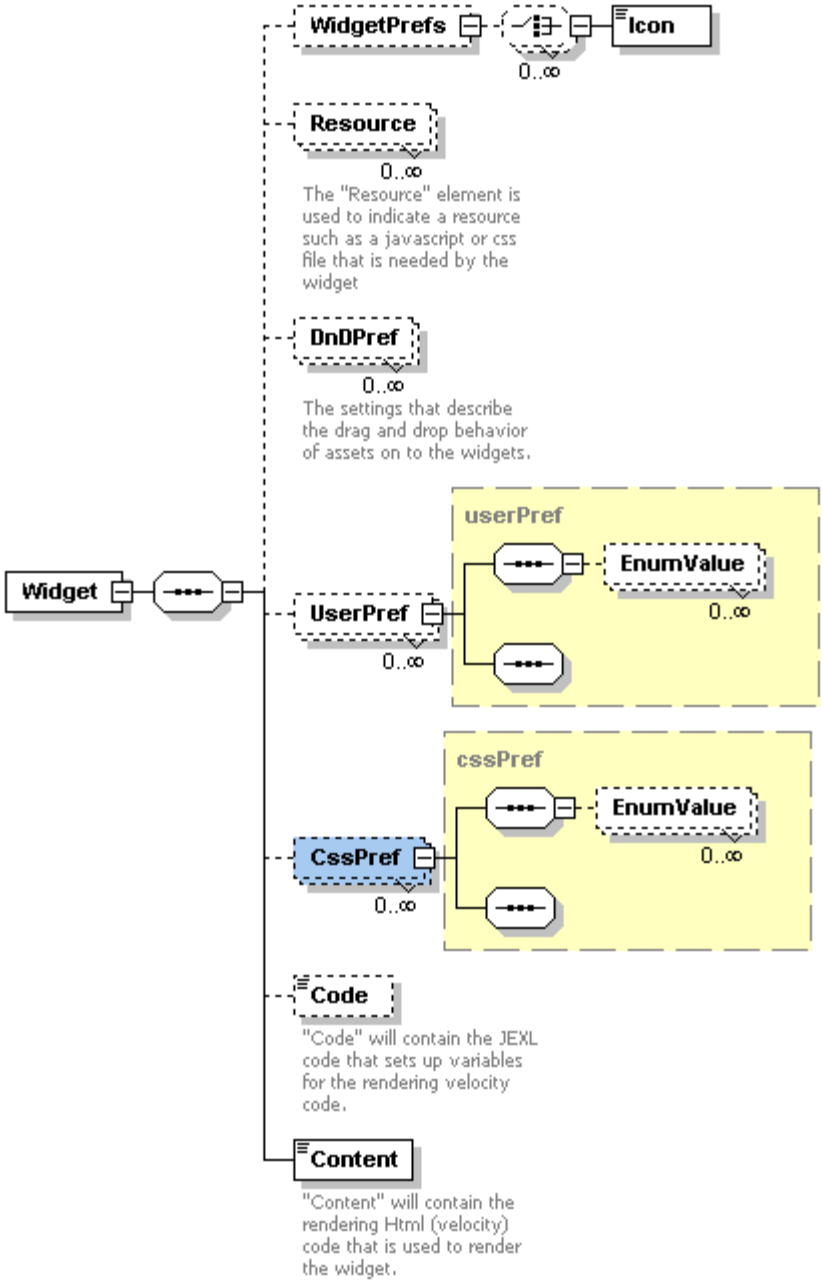
Here {widgetName} specifies the name of the widget. Let's examine each type of file.

Widget Definition

/rxconfig/Widgets/{widgetName}.xml

This file defines the widget. Its format is defined by PSWidgetDefinition.xsd. The widget name should be camel case and start with a prefix uniquely identifying the author's company, for example a calendar widget created by Percussion would be called "percCalendar".

Here is the schema for the top level XML structure, followed by a description of each section:



WidgetPrefs

Defines attributes of the widget:

```
<WidgetPrefs title="Hello World"
    contenttype_name="percHelloWorld"
    description="Widget for saying hello to the world"
    author="Percussion Software Inc"
    thumbnail=
         "/rx_resources/widgets/images/widgetIconHelloWorld.png"
    preferred_editor_width="550"
    preferred_editor_height="525"
    create_shared_asset ="true"
    is_editable_on_tempate ="true"
    />
```

- **title** – The name displayed in the widget tray
- **contenttype_name** – if the widget is backed by an asset, this is the internal name of the content type to use to edit and store the asset
- **author** – Identifies the author, not currently used by the product
- **thumbnail** – the image displayed in the widget tray
- **preferred_editor_width** – the width in pixels used to display the widget's asset editor. If not specified, the default value of 800 is used.
- **preferred_editor_height** - the height in pixels used to display the widget's asset editor. If not specified, the default value of 400 is used.
- **create_shared_asset** – Used to force a widget to use a local asset only. If "false", then only a local asset can be created, if "true", then a shared or local asset may be created. If the property is not specified the default value of "true" is used. See the Title widget for an example of this behavior. Widgets using assets that are resources must specify "true".
- **is_editable_on_tempate** – When the widget is placed on a template, this is used to prevent including content in the widget on the template. If "false", then an asset may only be created/placed in the widget when editing a page. If "true", then an asset may be created/placed in the widget on either a template or a page. If the property is not specified the default value of "true" is used. See the Title widget for an example of this behavior.

Resource – not supported, planned for future use

DnDPref – not supported, planned for future use.

UserPref

User preferences define the widget properties that can be configured from the Layout tab of a Template or Page. This is the configuration used to define how the widget output is rendered. Any configuration used to define the content that is rendered should be defined in the Content Type that is used to edit the Widget's Asset (see the section on "Content Types"). For example, in a Google Map widget, a UserPref could be used define the size of the map rendered on the page, while a Google Map Asset would include a field to specify the coordinates to center the map on.

The built-in widget configuration editor supports 4 types of data: bool, enum, number and string. Three types of controls are used with them: checkbox, combobox and editbox for number and string. The only validation provided is to check that a number field is in fact a number. Here are examples of each:

- **string** (displays an editbox):

```
<UserPref name="width"
  display_name="Width in pixels or percent"
  required="false"
```

```
  default_value=""
  datatype="string"/>
```

- **bool** (displays a checkbox):

```
<UserPref name="summary"
  display_name="Show page summary"
  default_value="false"
  datatype="bool"/>
```

- **number** (displays an editbox):

```
<UserPref name="maxlength"
  display_name="Max List Length"
  datatype="number" />
```

- **enum** (displays a droplist):

```
<UserPref name="wrapper"
    display_name="Choose format"
    required="true"
    datatype="enum" default_value="h1">
    <EnumValue value="h1" display_value="Heading 1" />
    <EnumValue value="h2" display_value="Heading 2" />
    <EnumValue value="h3" display_value="Heading 3" />
    <EnumValue value="h4" display_value="Heading 4" />
    <EnumValue value="h5" display_value="Heading 5" />
    <EnumValue value="h6" display_value="Heading 6" />
    <EnumValue value="paragraph" display_value="Paragraph" />
    <EnumValue value="div" display_value="Div" />
</UserPref>
```

CSSPref

The CSS Preferences define the widget properties that can be configured from the Style tab of a Template or Page. Best practice is to at least define a CSS Root Class name, but any CSS class names that should not be hard-coded in the output should be editable in this section.

```
<CssPref name="rootclass"
  display_name="CSS root class"
  datatype="string" />
<CssPref name="summaryclass"
  display_name="CSS page summary class"
  datatype="string" />
```

Code

The Code element is used to pre-process the user preferences, CSS preferences, and any asset data to set up variables that will be used by the Velocity code in the Content section to render the widget output. Jexl is the only supported code type. Below is an example and brief explanation. Note that the JEXL code is wrapped in a CDATA section. This is best practice to eliminate the need to escape commonly occurring characters.

```
<Code type="jexl">
<![CDATA[
$wrapper = $perc.widget.item.properties.get('wrapper');
$rootclass=$perc.widget.item.cssProperties.get('rootclass');
```

```
    $assets = $rx.pageutils.widgetContents($sys.assemblyItem, $perc.widget.item.id,
        null, null);
    if (!empty($rootclass))
        $classAttribute = 'class="' + $rootclass + '"';
    ]]>
    </Code>
```

- `$wrapper = $perc.widget.item.properties.get('wrapper')` - Get the value of the widget pref "wrapper" which defines the element to use and assign it to the "$wrapper" variable.
- `$rootclass==$perc.widget.item.cssProperties.get('rootclass');` - Get the value of the CSS pref "rootclass" and assign it to the "$rootclass" variable
- `$assets = $rx.pageutils.widgetContents($sys.assemblyItem, $perc.widget.item.id, null, null)` – Used to load any asset(s) contained in the widget if asset values are needed in the code section.
- `$classAttribute = 'class="' + $rootclass + '"'` – Set up the class attribute to use as inline HTML

Jexl functions and their use will be covered in detail in "JEXL Functions".

Content

The Content element contains the rendering Html and Velocity code that is used to render the widget. Velocity is the only supported content type. Below is an example and brief explanation. Note that the velocity is wrapped in a CDATA section. This is best practice to eliminate the need to escape commonly occurring characters.

```
    <Content type="velocity">
    <![CDATA[
   <div $!classAttribute>
      #loadRelatedWidgetContents()
      #if( ! $perc.widgetContents.isEmpty() )
         #set($node = $perc.widgetContents.get(0).node)
         $rx.pageutils.html($node,'rx:text')
      #elseif ($perc.isEditMode())
      <$wrapper class="hello-world-sample-content" title="This hello world widget is
showing sample content" ></$wrapper>
      #end
   </div>    ]]>
    </Content>
```

- `#loadRelatedWidgetContents()` – Load the asset and calls `$perc.setWidgetContents()` ,which is required to edit the asset w/in the page.
- `<div $!classAttribute>` – Outputs div element with class attribute value that was prepared in the Code section – note – the ! tells the processor to output nothing if the variable is not defined (the default is to output the string "$variablename".
- `#set($node = $perc.widgetContents.get(0).node)` – checks if there is an asset, and if so, gets its JCR Node and assigns it to $node
- `$rx.pageutils.html($node,'rx:text')` – if there is an asset, gets the value from the "text" field of the asset as HTML content.

Velocity macros will be covered in more detail in "Velocity Macros Functions".

Editing Resources

Resource files used only during page editing should be placed under /rx_resources/ in a subdirectory with the same name as the widget:

- /rx_resources/widgets/{widgetName}/css/<widgeName>.css – any css used only during page editing.
- /rx_resources/widgets/{widgetName}/images/<widgeName>Icon.png – the widget icon, displayed in the Widget Tray.
- /rx_resources/widgets/{widgetName}/images/<widgeName>Placeholder.png – icon to use when a widget that uses an asset is empty (does not yet have an asset), or for widgets that do not render while editing a page (for example, the Flash widget only renders during preview and on a published page).
- /rx_resources/widgets/{widgetName}/js/<widgeName>.js – any javascript functions used only during page editing.
- /rx_resources/images/ContentTypeIcons/filetypeicons<assetType>.png – icon used for the asset type in the Finder, must be specified in the content type properties.

Runtime Resources and File Dependencies

Additional files used by a widget at runtime should place them under the "/web_resources/widgets/" directory in a subdirectory with the same name as the widget (e.g. "/web_resources/widgets/helloWord/"). The types of files are further grouped in subdirectories by type:

- /web_resources/widgets/{widgetName}/css
- /web_resources/widgets/{widgetName}/images
- /web_resources/widgets/{widgetName}/js

Resource Definition File

Each of the files that need to be included in the header of the generated page that are required at runtime must be listed a Resource Definition file in the "/rxconfig/Resources" directory . The file naming convention is {widgetName}.xml (e.g. "/rxconfig/Resources/percHelloWorld.xml").

Here is an example resource file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Resources>
    <asset primary="true" id="flashBinary" contentType="percFlashAsset"
        legacyTemplate="perc.fileBinary">
        <linkAndLocations type="jexl">
        <![CDATA[
        $rx.resourceHelper.createDefaultLinkAndLocations($perc);
        ]]>
        </linkAndLocations>
    </asset>
    <file path="/web_resources/widgets/category/js/category.js" type="javascript"
        id="category_js">
        <dependency refid="percSystem.jquery"/>
    </file>
    <file path="/web_resources/cm/css/dynatree/skin/ui.dynatree.css" type="css"
        id="jquery-dynatree_css"/>
    <file id="jquery-dynatree_js" path="/web_resources/cm/jslib/jquery.dynatree.js"
        type="javascript">
```

```
        <dependency refid="percSystem.jquery" />
    </file>
</Resources>
```

<asset> Elements

Certain asset types represent resources that are published individually as files in addition to being referenced as widget content on a page. Currently this includes the File, Image, and Flash asset types. An "asset" entry in the resource definition file informs CM1 how to publish the asset file. Here is an example:

```
<asset primary="true" id="myWidgetBinary" contentType="percMyWidgetAsset"
       legacyTemplate="perc.fileBinary">
       <linkAndLocations type="jexl">
       <![CDATA[
         $rx.resourceHelper.createDefaultLinkAndLocations($perc);
       ]]>
       </linkAndLocations>
</asset>
```

To configure your own asset type to publish as files, use the above example verbatim, changing only the "id" and "contentType" attribute values:

- id – A unique identifier to use to generate file links in the widget's Code or Velocity sections (see below)
- contentType – The name of the asset's content type, used to match the resource definition in order to determine the target location when publishing the asset. There are additional settings required on the content type as well, see the "Content Types" section of this document for details.

The jexl is boiler plate and uses the default location scheme used by CM1. In CM1 the file path and name of a published resource should match the path and name represented in the Finder for the asset. For example, a File in CM1 with the Finder path of "/Assets/uploads/files/file1.pdf" will be published to "/uploads/files/file1.pdf".

To generate a proper link to the file in the widget output, include the following Jexl where `<resource_file_name>` is the name of the resource definition xml file, and `<asset.id>` is the id attribute value specified for the "asset" element in the resource definition file.

```
$link=$rx.pageutils.itemLink($linkContext, $assetItem,
"<resource_file_name>.<asset.id>");
```

Here is a more complete example:

```
$assetItems = $rx.pageutils.widgetContents($sys.assemblyItem, $perc.widget, null,
     null);
$perc.setWidgetContents($assetItems);
if ( ! $assetItems.isEmpty() ) {
     $assetItem = $assetItems.get(0);
     $link=$rx.pageutils.itemLink($linkContext, $assetItem,
          "perc_widgets_myWidget.myWidgetBinary");
}
```

See `$rx.pageutils.itemLink()` for more information.

<file> Elements

References for the “file” element entries in the resource definition file will be automatically added to the head tag of every pages CM1 generates, and path differences between preview and published pages are automatically handled. File dependencies can be used to ensure correct ordering. Any other resource that a file requires to be loaded first should be listed as a dependency of the file and referenced by ID.

The following system dependencies are currently defined, and can be identified as dependencies using the refid “percSystem.<id>”, e.g. “percSystem.jquery” for jquery.js:

```
<file id="jquery" path="/web_resources/cm/jslib/jquery.js" type="javascript"/>
<file id="perc_decoration_style" path="/web_resources/cm/css/perc_decoration.css"
   type="css"/>
<file id="jquery-ui" path="/web_resources/cm/jslib/jquery-ui.js" type="javascript">
   <dependency refid="percSystem.jquery" />
</file>
<file id="perc-jquery-ui-style" path="/web_resources/cm/themes/smoothness/jquery-ui-
   1.8.9.custom.css" type="css">
   <dependency refid="percSystem.perc_decoration_style" />
</file>
```

CM1 currently ships with jQuery version 1.5, and jQuery UI v1.8.9. To avoid conflicts, use js libraries that are compatible with these versions and confirm the version when you create your widget. If your widget depends on them, be sure to include them as dependencies in the widget’s resource file.

Content Types

Content Types are used to define the asset type used by a widget. Content Types are defined using the Workbench. This document does not cover how to use the workbench or create content types, but outlines the required configuration(s) needed to develop widgets. When creating a content type, the following values should be supplied:

- Name: The name should be camel case and begin with a lowercase prefix uniquely identifying the author. For widgets built by Percussion, use “perc” as the prefix, for example “percHelloWorld”. This name should be specified in the widget definition as the “contenttype_name” attribute value.
- Label: Displays in the UI as the asset type.
- Community: Ensure it is visible to the “Default” community.
- Default Workflow: “Default Workflow”
- Allowed Workflows: “Default Workflow”, “Local Content”
- Content Type Icon: specifies the icon that will appear in the Finder. Should be defined with this path/name: /rx_resources/images/ContentTypeIcons/filetypeicons<assetType>.png
- The “sys_title” field label should be renamed to “Name” – this is the name that appears in the Finder for the asset.

Otherwise all default settings and system fields should be acceptable without modification.

In general, all non-system fields should be local fields. If several widgets were being developed and packaged together, then they could create their own shared group if that made sense and include it

with the package. A common package can also be created with the shared group, and then widgets that use that package can specify it as a dependent package. Child Fields are not supported.

For fields that should appear in the Meta-Data section of the asset editor, check off the “Treat field as meta-data” property in the “All Properties” dialog of the field editor.

For fields that you want to exclude from export via the Content API, uncheck the “Allow export of this field”. This feature was added in v2.4 of CM1, and will be available in the SDK Workbench installation planned for v2.5.

Field Validation

Field level validation is supported by the CM1 asset editor. You may use any of the built in validations. The error message is supported. The simplest validation is to check off the “Required” field property.

Supported System Controls

The following system controls are supported by the CM1 UI. For now, use existing asset content types as a guide for configuration options:

- sys_CalendarSimple
- sys_DropDownSingle
- sys_EditBox
- sys_File
- sys_HiddenInput
- sys_RadioButtons
- sys_SingleCheckBox
- sys_TextArea
- sys_tinymce

Choices

For controls that support choices (sys_DropDownSingle), there are two supported options for defining the choices:

- Retrieve from table
- Define choices for this control only

The “Use a keyword” and “Retrieve from xml application” options are not supported.

Retrieve choices from table

This option is only intended to be used for customer specific custom widgets that need to access data available in the customer environment. The choices must be retrieved using a datasource that references a database/schema other than the CM1 repository. To enable the datasource, add a “-ds.xml” file to the “AppServer/server/r/\deploy” directory to enable a datasource. Before the widget is installed, the customer will need to do create/configure this file to use the same datasource name and restart the CM1 server. You cannot include this file in your widget package as that will cause it to be overwritten each time the package is installed/updated.

Custom TinyMCE configuration

Note that the percSimpleTextAsset uses sys_tinymce control with a custom configuration. This is done by specifying the parameter “tiny_config” with the value:
../rx_resources/tinymce/simple_config.js

This configuration file is part of the perc.baseWidgets package and is not public. Your widget should create and use its own configuration

Unsupported System Controls

The following system controls are not supported by CM1

- sys_CheckBoxTree
- sys_EditLive
- sys_FileWord
- sys_HtmlEditor
- sys_Table – Child fields are not supported
- sys_VariantDropDown
- sys_webImageFX

Non-Public Controls

These controls are part of individual widget packages:

- categoryListWidgetControl
- commentsFormWidgetControl
- formWidgetControl
- imageAutoListWidgetControl
- imageWidgetControl
- pageAutoListWidgetControl
- percQueryControl
- percTagListControl
- tagListWidgetControl

Custom Controls:

Custom controls can be created and packaged. Custom control definitions should be placed in "rx_resources\stylesheets\controls". It is permissible to copy an existing non-public control in order to make a private copy of it for use with your widget's asset type.

Shared Fields:

Currently there are no shared fields available for public use.

Resources

A resource is an asset that is published as an independent file. The following settings are required for Content types that are used to create resources:

- Select "Produces Resource" on the Content Type's "Properties" tab.
- The sys_title field must specify the file name to use when publishing.
- The field to hold the binary data should be marked as "Treat field as binary" in the field editor, and must be named "item_file_attachment". Additionally there must be a field named "item_file_attachment_size" which holds the size of the file.
- You may want to additionally use the sys_FileInfo() and perc_AssetRename() Input Transforms see the percFileAsset content type as an example.

Additional requirements:

- The Widget Resource file must specify an <asset> element for your asset type (see the "Resource Defintion File" section).
- You also need to specify the following the widget prefs in the Widget Definition XML file:
  `create_shared_asset ="true"`

Custom Controls:

Custom controls can be developed for use in Asset Editing. See ("Writing Custom Controls" in the Rhythmyx Technical Reference Manual for details). Custom controls should use the following file naming convention and location:

rx_resources/stylesheets/controls/{widgetName}Control.xsl

Be sure to follow the instructions for controls that can be included in a package.

If you write a custom control, you may need to perform some work before the form is actually submitted. To hook into the framework's save handling, you create a function and register it by calling:

```
$.topFrameJQuery.PercContentPreSubmitHandlers.addHandler(myHandler)
```

After the user presses the Save button, the framework will call each registered handler in the order it was registered. If any handler returns false, the save is canceled and the editor remains open (the handler that returned false is responsible for notifying the user of the problem.) The signature for myHandler is:

```
function myHandler()
{
	//Some code
	return abool; //abool is a boolean
}
```

The framework will remove the handler after the dialog is successfully dismissed (either saved or cancelled.) Be sure to include the return statement; without that the form doesn't get submitted.

If writing a custom control, the framework guarantees one library: jquery (jquery-1.5 at the time of this writing.) Confirm the version when you create your control.

JavaScript

Javascript used by the widget at runtime should be defined in files and specified as resources and placed under the web_resources directory as already mentioned. Externally referenced Javascript files should be defined inline in the widget velocity output. For example:

```
<Content type="velocity">
<![CDATA[
<div>

#if($assets.isEmpty())
    <script type="text/javascript"
      src="http://maps.google.com/maps/api/js?sensor=true"></script>
#else
    ...
```

```
 #end

</div>
 ]]>
 </Content>
```

CM1 currently ships with jQuery version 1.5, and jQuery UI v1.8.9. To avoid conflicts, use js libraries that are compatible with these versions and confirm the version when you create your widget. If your widget depends on them, be sure to include them as file elements in the widget's resource definition file.

NOTE: No other JavaScript shipped with CM1 should be considered public or available for use in your widget, so take care when copying an existing widget.

CSS

All css for preview/published page should be in the theme, grouped together. Use the widget name to namespace class ids to avoid collisions.

During assembly, if a widget doesn't have any content and it is being rendered in edit mode, it should output sample content that is representative of the widget. To determine if the widget is being assembled in the edit context, check the $perc.isEditMode( ) flag in velocity.
The widget should have a css file that contains the css properties to render this background. An example for the page auto list looks like this:

```
.pagelist-sample-content
{
    height: 100px;
    background: transparent
url(/Rhythmyx/rx_resources/widgets/pageAutoList/images/widgetLinkAutoListSampleBackground.png)
no-repeat scroll center center;
    filter: alpha(opacity=50);
    opacity: .50;
}
```

The "`perc-widget`" class is added by the system when it is rendering the widgets and can be used for styling all the widgets. Each widget can provide a unique css class as described above to configure all the instances of that type of widget, and then use the rootclass CSS preference to provide CSS for an individual instance.

JEXL Functions

JEXL functions are used within the Code section and to some extent in the Content section of the Widget Definition file. . Most JEXL should be limited to use in the Code section as much as possible, where full JEXL logic syntax is supported, and used to set up variables for use in the Content section in order to keep the Content section clean and readable. The Content section should contain only the JEXL needed to impact rendering, or that is needed to be mixed inline with rendering Velocity output.

System JEXL Functions

The following JEXL functions are public and available for use. If a method is not listed here, it should not be considered public and is not safe to use, even if it is used in one of the widgets shipped with CM1. Properties should be considered read-only.

$perc.isEditMode()

Determine if the widget is rendering in the page editor as opposed to during preview or publishing, typically used in the Content section to display empty content, used in the Velocity code in the Content section:

```
#if ($perc.isEditMode())
    <div class="file-sample-content" title="This file widget is showing sample
     content"></div>
#elseif ($perc.isPreviewMode())
    ...
#else
    ...
#end
```

$perc.isPreviewMode()

Determine if the widget is rendering during assembly of a page in preview mode, typically used in the Content section for widgets that require delivery tier services that are not available during preview.

```
#if (($perc.isPreviewMode())
    <div class="file-sample-content" title="This file widget is showing sample
     content"></div>
#else
    ...
#end
```

$perc.linkContext

Gets the link context, to be passed into the `$rx.pageutils.itemLink()` function.

$perc.page.id

Gets the id of the current page being assembled, typically to be passed into `$rx.pageutils.getItemPath()`

$perc.page.linkTitle

Gets the title of the current page being assembled. The Title widget uses this to display the title as a heading on the page:

```
<$wrapper $!classAttribute >$tools.esc.html($!perc.page.linkTitle)</$wrapper>
```

$perc.setWidgetContents()

If a widget uses an asset, this method must be called to enable the page editor to properly edit the asset. The asset must first be found and loaded using `$rx.pageutils.widgetContents()` , and then the value returned is passed into the `$perc.setWidgetContents()` method. For example:

```
$assetItems = $rx.pageutils.widgetContents($sys.assemblyItem, $perc.widget, null,
null);
$perc.setWidgetContents($assetItems);
```

If the asset is not needed for further processing in the Code section, then the convenience Velocity macro `#loadRelatedWidgetContents()` may be used instead to perform the same actions. The asset can then be accessed by calling `$perc.widgetContents.get(0).`

$perc.widget

The widget being rendered, automatically bound to the current assembly context, passed into `$rx.pageutils.widgetContents().`

$perc.widget.item.cssProperties.get()

Used to get the value of a css preference defined in the Widget Defintion. For example, to get the "rootclass" css preference:

```
$rootclass=$perc.widget.item.cssProperties.get('rootclass');
```

$perc.widget.item.id

Get the id of the current widget, passed into a call to `$perc.widgetContents()`.

$perc.widget.item.name

Get the name of the widget.

$perc.widget.item.properties

Get the value of a widget preference defined in the Widget Defintion. For example, to get the "wrapper" widget preference:

```
$perc.widget.item.properties.get('wrapper');
```

$perc.widgetContents

Get the asset item that has been set by a previous call to `$perc.setWidgetContents()`. The value returned is a list of assets (java.util.List). If there is no asset set on the widget, the list will be empty; otherwise the asset will be the first entry in the list:

```
$perc.widgetContents.get(0);
```

To get field values from the asset, call `$perc.widgetContents.get(0).getNode()` to get the JCR Node, for example:

```
$start = $assetItem.getNode().getProperty('start_time').Date;
```

One other method is supported, `$perc.widgetContents.get(0).getFolderPath()`, used to get the folder path of the asset as a string.

$rx.pageutils.getDeliveryServer()

Get the url of the delivery server as defined in "config/delivery-servers.xml". May be empty if no entry is found.

```
$sUrl = $rx.pageutils.getDeliveryServer() + "/feeds/rss/" + $siteName + "/" +
$feedname +"/";
```

$rx.pageutils.getItemPath()

Get the finder path to a page.

```
$pageId = $perc.page.id;
if(!empty($pageId))
    $finderPath = $rx.pageutils.getItemPath($pageId);
```

This should not be used to generate links to pages or assets. Instead use $rx.pageutils.itemLink().

$rx.pageutils.html()

Gets html content from a field and escapes the field's contents to html if needed. If the field used the "sys_tinymce" control, it is assumed to be an html field and the contents will not be escaped. If the field is not found an exception will be thrown.

```
title=$rx.pageutils.html($assetItem,'displaytitle');
```

$rx.pageutils.itemLink()

Creates the appropriate link to the specified file or page. The link will automatically be adjusted based on whether the page is being previewed or published. The item must be a resource asset or a page.

To create a link to a resource, you need to supply a "resource definition id" as the 3rd parameter ("percSystem.fileBinary" in the example below). This ID is defined by the "<asset>" entry in the widget's Resource Defintion File. See that section for details.

```
$linkContext = $perc.linkContext;
$assetItems = $rx.pageutils.widgetContents($sys.assemblyItem, $perc.widget,
     null, null);
$perc.setWidgetContents($assetItems);
if ( ! $assetItems.isEmpty() ) {
    $assetItem = $assetItems.get(0);
    $link=$rx.pageutils.itemLink($linkContext, $assetItem,
     "percSystem.fileBinary");
    $title=$rx.pageutils.html($assetItem,'displaytitle');
    $link = $tools.esc.html($link);
}
```

To create a link to a page:

```
$linkContext = $perc.linkContext;
$assetItems = $rx.pageutils.widgetContents($sys.assemblyItem, $perc.widget,
     null, null);
$perc.setWidgetContents($assetItems);
if ( ! $assetItems.isEmpty() ) {
    $assetItem = $assetItems.get(0);
    $link=$rx.pageutils.itemLink($linkContext, $assetItem);
    $title=$rx.pageutils.html($assetItem,'displaytitle');
    $link = $tools.esc.html($link);
}
```

Then in the Content (Velocity) section:

```
#if( ! $perc.widgetContents.isEmpty() )
    <a href="$link" target=$target>$title</a>
#elseif ($perc.isEditMode())
    <div class="file-sample-content" title="This file widget is showing
     sample content"></div>
#end
```

$rx.pageutils.widgetContents()

Loads the assets of a widget that are related to the specified page and/or template. The value returned is a list of assets (intended for future support of widgets that can contain multiple assets). If there is no asset set on the widget, the list will be empty; otherwise the asset associated with a widget will be the first entry in the list. If a widget uses an asset, this must be called followed by a call to `$perc.setWidgetContents()`. For example:

```
$assetItems = $rx.pageutils.widgetContents($sys.assemblyItem, $perc.widget, null,
null);
$perc.setWidgetContents($assetItems);
 if ( ! $assetItems.isEmpty() ) {
    $assetItem = $assetItems.get(0);
    $link=$rx.pageutils.itemLink($linkContext, $assetItem);
    $title=$rx.pageutils.html($assetItem,'displaytitle');
    $link = $tools.esc.html($link);
 }
```

If the `#loadRelatedWidgetContents()` Velocity macro is used, then the call to `$perc.setWidgetContents()` is not required.

This function may also be used to execute a JCR query:

- The optional 3rd argument of the $rx.pageutils.widgetContents() method can be used to specify the name of a content finder. The "`perc_AutoWidgetContentFinder`" can be used to execute a JCR query and return results.

- The 4th argument is a map of parameters to pass to the content finder. The two parameters used by the perc_AutoWidgetContentFinder are:
  - query - The JSR-170 query to be performed
  - max_results - Optional parameter. It is the maximum number of the returned result from the find method if specified, zero or negative indicates no limit. It defaults to zero if not specified.

Example usage:

```
$params=$rx.string.stringToMap(null);
$params.put('query', ($assetItem.getNode().getProperty('query').String));
$params.put('max_results', $maxlength);

$finderName="Java/global/percussion/widgetcontentfinder/perc_AutoWidgetContentFinder";
$relresults=  $rx.pageutils.widgetContents($sys.assemblyItem, $perc.widget.item.id,
   $finderName, $params);
```

$rx.string.*

Utility methods for string manipulation. All methods are available for use.

$rx.codec.*

Utility methods for dealing with base64 and XML encodings. All methods are available for use.

$rx.date.*

Utility methods for date manipulation. All methods are available for use.

$rx.db.get()

Utility method to execute a sql query against a given datasource, e.g.:

```
$rx.db.get('mydatasourcename', 'select column from table where...');
```

$sys.assemblyItem

The assembly item for the current page being assembled, automatically bound to the current assembly context, passed into `$rx.pageutils.widgetContents()`.

None of the methods on this object are public.

$tools

All velocity tools exposed by $tools are available for use:
Velocity tools are available using the $tool prefix. For details, see the Velocity tools documentation: http://velocity.apache.org/tools/devel/.

Tool	Class
$tools.alternator	org.apache.velocity.tools.generic.AlternatorTool
$tools.date	org.apache.velocity.tools.generic.DateTool
$tools.esc	org.apache.velocity.tools.generic.EscapeTool
$tools.mill	org.apache.velocity.tools.generic.IteratorTool
$tools.list	org.apache.velocity.tools.generic.ListTool
$tools.math	org.apache.velocity.tools.generic.MathTool
$tools.number	org.apache.velocity.tools.generic.NumberTool
$tools.render	org.apache.velocity.tools.generic.RenderTool
$tools.sorter	org.apache.velocity.tools.generic.SortTool
$tools.parser	org.apache.velocity.tools.generic.ValueParser

Custom Jexl Functions

If you need additional functionality, you can implement custom JEXL function extensions. See the "JEXL Extensions" section of the Rhythmyx Technical Reference Manual for details on how to create a custom JEXL function. Things to keep in mind when registering the JEXL extension:

- Extensions are registered in the Workbench in the System Design Tab.

- The name of your extension will be used to make calls from Jexl. To reference your function, use the $user prefix in your code, followed by the extension name, and then the Jexl function name in the extension, dot-delimited, as in `$user.myextension.mymethod.`
- Be sure to specify "`com.percussion.extension.IPSJexlExpression`" in the "Supported interfaces" list.
- Include your java code in a jar file, specify that jar file and jars for any other required libraries as "Required files". This will automatically package your jars with the extension, and it will isolate your extension at runtime in it's own class loader.

Things to keep in mind when writing a Jexl extension:

- Be sure to properly annotate your methods
- Extensions should be thread-safe and stateless
- Extensions may only call public APIs.

Velocity Macros Functions

loadRelatedWidgetContents()

Loads the asset and sets it on the $perc context. Calling this macro is the equivalent of calling:

```
$assetItems = $rx.pageutils.widgetContents($sys.assemblyItem, $perc.widget, null,
null);
$perc.setWidgetContents($assetItems);
```

If those calls have been made in the Code section, it is redundant to call this macro in the Content section. Typical usage:

```
#loadRelatedWidgetContents()
#if( ! $perc.widgetContents.isEmpty() )
  #set($node = $perc.widgetContents.get(0).node)
  $node.getProperty('rx:html').getString()
#end
```

Packaging

Widgets are delivered using the packaging capabilities of the platform. The Package Builder is used to package a widget, the Package Installer is used to install them, and the Package Manager is used to view, installed packages, check their status, and reinstall or uninstall a package.

The Package Builder creates "Descriptors", which are a package definition. Descriptors are then used by the Package Builder to create the package file (*.ppkg). The Package Installer is used to install a package using the package file. Objects on the system can only be part of one package at a time, either a Descriptor created with the Package Builder, or as part of a package installed by the Package Installer. The package file is only used by the system during installation.

Package Builder

The package builder is launched by running PercussionPackageBuilder.exe located in the root of your devtools installation directory.

When building a package for a widget, follow these guidelines:

- Generally create a single package for each Widget. Be sure to follow the naming convention for widget packages. It may make more sense to package two or more widgets together that need to be used together. The Percussion Blog Post and Blog Index widgets are an example of this.
- Select the Widget's Content Type under "Design Objects" on the "Selection" tab if the widget includes an Asset type. All required dependencies of the content type including custom controls, table schema, extensions etc. will be automatically included for you.
- If you have created any custom JEXL functions, select the extension under "Design Objects" on the "Selection" tab. Any required files/jars for the extension that were specified as "Required Files" when the extension was registered through the Workbench will be automatically included in the package along with the extension.
- Add all files created for and used by your widget as "File Resources" on the "Selection" tab. Note that in addition to selecting a single file, you can select a folder and click the "Add >" button to add all files within that folder and its subfolders as File Resources. You will need:
    - /rxconfig/Widgets/{widgetName}.xml
    - /rxconfig/Resources/{widgetName}.xml
    - rx_resources/widgets/{widgetName}
    - web_resources/widgets/{widgetName}
- You should not need to specify a "Configuration Definition" or a "Default Configuration".
- Check the "Dependencies" tab to ensure your package does not inadvertently depend on any other packages. None of the packages installed with the system are public, and your package should not require or add any of them as a dependent package.
- If your widget uses shared resources that are required by more than one widget, and you are not packaging those widgets together in a single package, put those shared resources in their own package first and then specify that package as a dependent package in the "Dependencies" tab by checking it off under "Additional Required Packages" (unless it is automatically included as a "Dependent Package".

Package Installer

This tool is used to install a widget. You should test your package by installing it onto a fresh instance of CM1. The package builder is launched by running PercussionPackageInstaller.exe located in the root of your devtools installation directory.

You may also need to use the installer to install a widget onto a new system in order to fix a bug or add new functionality. After installing the package, you need to convert it to a "source package" or Descriptor so that you can modify its contents and then re-package it. See "Convert to Source" in the "Package Manager" section for details.

Package Manager

The Package Manager is used to view installed packages on a system, verify if anything installed by the package has been modified since installation, set visibility for the package, or uninstall a package. The manager is a web app with this url: http://<server>:<port>/cm/packages

Visibility

After installing a new package, it is automatically associated with the single "Default" community used by CM1. There should be no need to adjust visibility settings.

Convert to Source

Objects installed via the Package Installer are "owned" by that package on that system and cannot be added to another package on that system. The same is true for objects included in a package descriptor created by the Package Builder – they are owned by that "source" package. installed

package can be converted to a descriptor or “source” so that its objects can be modified and repackage.

THIS SHOULD ONLY BE USED ON PACKAGES FOR WHICH YOU ARE THE LEGITIMATE AUTHOR.

Select the package and then click on the “Convert to Source” entry under the “Actions” menu. This converts the installed package into a Descriptor, leaving it in the same state as if its objects had been created on that system and then added to a descriptor using the Package Builder. The new Descriptor can be seen in the Package Builder (if open it must be restarted).

Convention & File Locations

Percussion widgets will follow the following conventions:

Object	Naming guidelines and file locations
Widget	Camel case and start with a namespace prefix uniquely identifying the author’s company, e.g. perc for widgets built by Percussion. Pattern: {namespace}{WidgetBaseName} (= {widgetName}) Example: percHelloWorld
Widget definition File	This file must conform to PSWidgetDefinition.xsd schema. Pattern: {widgetName}.xml Location: <rxroot>/rxconfig/Widgets/ Example: percHelloWorld.xml
Widget Content Type	Pattern: {widgetName}Asset
Widget resource files (e.g. css, js, images)	Resources used during editing only are stored under rx_resources. <rx_root>/rx_resources/widget/{widgetName}/ Resources used for rendering the widget are stored under web_resources. <rx_root>/web_resources/widget/{widgetName}/ Any sub-folder structure can be used , but typically, you may have images, css and js sub-folders. All resources that need to be included as links in a published file must be registered with the system by including them in a Resource Description file. (See **Widget resource description file** entry for naming and location details.)
Widget icon	Pattern: {widgetName}Icon.png Location: <rx_root>/rx_resources/widget/{widgetName}/images/
Widget sample background	Pattern: {widgetName}Placeholder.png Location: <rx_root>/rx_resources/widget/{widgetName}/images/ This location is referenced in the corresponding css file. This background image should follow the pattern of other widgets – place the widget icon in the middle of a panel. [tbd – add more info about opacity and such.]
Widget resource description file	This file must conform to PSResourceDefinitionGroup.xsd schema Pattern: {widgetName}.xml Location: <rxroot>/rxconfig/Resources/
Custom control for content type that is specific to that type	Starts with lower-case, then camel cased. The name should describe the purpose/type of the control, prepended with your namespace. Pattern: {namespace}{ControlType}.xsl Location: <rxroot>/rx_resources/stylesheets/controls Any resource files used by these controls should be stored under <rxroot>/rx_resources/widget/{widgetName}/[css,js,images…]/

Object	Naming guidelines and file locations
css class names, html element ids	Names used by widgets for editing or rendering. Dash (-) separated, lower-cased nouns, prefixed with the namespace prefix (i.e. *perc-)*. Thought should be given as to the re-usability of the class and the name should be built accordingly. For example, if you happen to be building a list and you want to have classes for odd and even entries, then you might create names such as perc-list-odd-entry, perc-list-even-entry. Third party tools keep their css class names – don't add a namespace prefix. Pattern: {namespace}-[unique-name-or-id] Example: perc-list-odd-entry
css class name for sample content	Pattern: {namespace}-[widget-name]-sample-content
Package name	Pattern: {namespace}.widget.{widgetName}.ppkg Example: perc.widget.helloWorld.ppkg If packaging just the widget, then use the above pattern. If the package is broader than this, choose a name for the package that describes it's contents and use the the following pattern: Pattern: {namespace}.widget.{packageName}.ppkg If packaging a gadget, use the following pattern: perc.gadget.{gadgetname}.ppkg
JEXL Extension	Pattern: Use names that describe the purpose of the class, using standard Java naming conventions and prepending your namespace prefix. Example: percStringUtils Example reference: $user.percHelloWorld
JEXL Method	Use names that describe the purpose of the method using standard Java naming conventions. (The methods are namespaced by the class.)

Cookbook

This section provides the steps needed to create various types of widgets, with links to the relevant section of this document.

Create a widget

1. Create a Widget Definition. Include any JavaScript, CSS, JEXL code or Velocity output.
2. Create an icon for the widget as well as any other files needed during editing.
3. Create or copy any runtime resources needed.
4. Create a Resource Defintion File for any javascript dependencies.
5. Package the widget.

Create a widget that uses an asset

1. Create a widget.
2. Create a Content Type.
3. Set the `contenttype_name` Widget Preference in the Widget Definition.
4. Call the `$rx.pageutils.widgetContents()` and the `$perc.setWidgetContents()` methods from the widget's JEXL code, or else call the `#loadRelatedWidgetContents()` from the widget's Velocity code
5. Be sure to include the Content Type when you package the widget.

Create an asset that is published as a file

1. Follow the steps to create a widget that uses an asset.
2. Follow the additional steps to create a Resource.
3. Call `$rx.pageutils.itemLink()` from the widget's JEXL code to generate the correct url to use to link to the file.

Create a widget that displays a list of pages

1. Create a widget.
2. Generate a JCR query to locate the pages. The page auto list widget is an example of using an asset to specify the criteria. Keep in mind that the `pageAutoListWidgetControl` is not public and cannot be used as is (make a copy or implement your own custom control).
3. In the widget's JEXL, call the `$perc.setWidgetContents()` method using the optional 3rd and 4th arguments to specify the "`perc_AutoWidgetContentFinder`" and JCR Query.
4. Then use `$rx.pageutils.itemLink()`to generate the correct url to use to link to each page.

Create a widget that displays a list of Files

1. Create a widget.
2. Generate a JCR query to locate the file assets. The image auto list widget is an example of using an asset to specify the criteria. Keep in mind that the `imageAutoListWidgetControl` is not public and cannot be used as is (make a copy or implement your own custom control).
3. In the widget's JEXL, call the `$perc.setWidgetContents()` method using the optional 3rd and 4th arguments to specify the "`perc_AutoWidgetContentFinder`" and JCR Query.
4. Then use `$rx.pageutils.itemLink()`to generate the correct url to use to link to each page.

Create a widget that doesn't render content in Edit Mode

1. Create a widget.
2. In the widget's Velocity code, use `$perc.isEditMode()` to conditionally suppress rendering.

Create a widget with an asset that is always local content

1. Follow the steps to create a widget that uses an asset.
2. In the widget's preferences, set `create_shared_asset ="false"`.

Create a widget with an asset that is not editable on a template

1. Follow the steps to create a widget that uses an asset.
2. In the widget's preferences, set `is_editable_on_tempate ="false"`.

FAQ

How do I give a widget to someone else?

1. Follow the instructions for creating a package.

How do I install a Widget onto a new system?

1. If the widget has not been packaged, follow the instructions for creating a package.
2. Follow the instructions for installing a package.

How do I see what widgets are installed?

1. Open the Package Manager.

How do I uninstall a widget?

1. Open the Package Manager.
2. Select the package and click the “Uninstall” option in the “Actions” menu.

How do I create a widget by copying an existing widget?

1. See these restrictions regarding copying existing widgets.