FATWIRE CORPORATION PROVIDES THIS SOFTWARE AND DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. In no event shall FatWire be liable for any direct, indirect, incidental, special, exemplary, or consequential damages of any kind including loss of profits, loss of business, loss of use of data, interruption of business, however caused and on any theory of liability, whether in contract, strict liability or tort (including negligence or otherwise) arising in any way out of the use of this software or the documentation even if FatWire has been advised of the possibility of such damages arising from this publication. FatWire may revise this publication from time to time without notice. Some states or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

Copyright © 2010 FatWire Corporation. All rights reserved.

The release described in this document may be protected by one or more U.S. patents, foreign patents or pending applications.

FatWire, FatWire Content Server, FatWire Engage, FatWire Satellite Server, CS-Desktop, CS-DocLink, Content Server Explorer, Content Server Direct, Content Server Direct Advantage, FatWire InSite, FatWire Analytics, FatWire TeamUp, FatWire Content Integration Platform, FatWire Community Server and FatWire Gadget Server are trademarks or registered trademarks of FatWire, Inc. in the United States and other countries. Oracle and Java are registered trademarks of Oracle and/or its affiliates. AIX, AIX 5L, WebSphere, IBM, DB2, Tivoli and other IBM products referenced herein are trademarks or registered trademarks of IBM Corporation. Microsoft, Windows, Windows Server, Active Directory, Internet Explorer, SQL Server and other Microsoft products referenced herein are trademarks or registered trademarks of Microsoft Corporation. Red Hat, Red Hat Enterprise Linux, and JBoss are registered trademarks of Red Hat, Inc. in the U.S. and other countries. Linux is a registered trademark of Linus Torvalds. SUSE and openSUSE are registered trademarks of Novell, Inc., in the United States and other countries. XenServer and Xen are trademarks or registered trademarks of Citrix in the United States and/or other countries. VMware is a registered trademark of VMware, Inc. in the United States and/or various jurisdictions. Firefox is a registered trademark of the Mozilla Foundation. UNIX is a registered trademark of The Open Group in the United States and other countries. Any other trademarks and product names used herein may be the trademarks of their respective owners.

This product includes software developed by the Indiana University Extreme! Lab. For further information please visit http://www.extreme.indiana.edu/.

Copyright © 2002 Extreme! Lab, Indiana University. All rights reserved.

This product includes software developed by the OpenSymphony Group (http://www.opensymphony.com/).

The OpenSymphony Group license is derived and fully compatible with the Apache Software License; see http://www.apache.org/LICENSE.txt.

Copyright (c) 2001-2004 The OpenSymphony Group. All rights reserved.

You may not download or otherwise export or reexport this Program, its Documentation, or any underlying information or technology except in full compliance with all United States and other applicable laws and regulations, including without limitations the United States Export Administration Act, the Trading with the Enemy Act, the International Emergency Economic Powers Act and any regulations thereunder. Any transfer of technical data outside the United States by any means, including the Internet, is an export control requirement under U.S. law. In particular, but without limitation, none of the Program, its Documentation, or underlying information of technology may be downloaded or otherwise exported or reexported (i) into (or to a national or resident, wherever located, of) any other country to which the U.S. prohibits exports of goods or technical data; or (ii) to anyone on the U.S. Treasury Department's Specially Designated Nationals List or the Table of Denial Orders issued by the Department of Commerce. By downloading or using the Program or its Documentation, you are agreeing to the foregoing and you are representing and warranting that you are not located in, under the control of, or a national or resident of any such country or on any such list or table. In addition, if the Program or Documentation is identified as Domestic Only or Not-for-Export (for example, on the box, media, in the installation process, during the download process, or in the Documentation), then except for export to Canada for use in Canada by Canadian citizens, the Program, Documentation, and any underlying information or technology may not be exported outside the United States or to any foreign entity or “foreign person” as defined by U.S. Government regulations, including without limitation, anyone who is not a citizen, national, or lawful permanent resident of the United States. By using this Program and Documentation, you are agreeing to the foregoing and you are representing and warranting that you are not a “foreign person” or under the control of a “foreign person.”

FatWire Gadget Server Developer's Guide
Document Revision Date: Jan. 27, 2011
Product Version: Version 1.1

FatWire Technical Support
www.fatwire.com/Support

FatWire Headquarters
FatWire Corporation
330 Old Country Road
Suite 207
Mineola, NY 11501
www.fatwire.com
Table of Contents

1 Introduction .......................................................... 5
   Before You Begin .................................................... 6
   Gadget Specifications ............................................ 6
      Asset Model and Templates ................................... 7
      Sample Assets .................................................. 7
      Auxiliary Files .................................................. 8
      OAuth Protocol .................................................. 9
   Sample Gadgets ................................................... 11
      List Gadget ...................................................... 11
      ThumbList Gadget .............................................. 12
      Slideshow Gadget .............................................. 13
      RSS Feed Gadget ................................................ 14
   Asset Structure ................................................... 14

2 Template Flow ...................................................... 15
   Template Flow for the List Gadget .............................. 16
   Differences in Template Flow ................................... 18
   Why Server Calls are Done Separately .......................... 18

3 Creating Your Own Gadgets ....................................... 19
   Creating Gadgets on Different CM Sites ....................... 20
   Custom Gadgets ................................................... 20
      New Gadget, Content Server Generates Only XML ............ 20
      New Gadget, Content Server Generates XML and Fields Additional Requests .......................... 20
      Same Gadget Logic, Different Content ........................ 21
   Prerequisites for Registering Gadgets ........................... 21
   Sample RSSFeed Gadget .......................................... 24
   Sample List Gadget ............................................... 26
Chapter 1

Introduction

This guide introduces template developers to the process of creating gadgets based on Content Server template code. Sample gadgets, which are included with Gadget Server, are used throughout this guide to illustrate the development task. This set of fully operational gadgets runs on the FirstSite II sample site. Their underlying asset model and template framework provide the tools developers need to get started with creating their own gadgets. One of the sample gadgets supports the OAuth protocol to provide an example of how developers can configure their own gadgets with OAuth support.

This chapter contains the following sections:

- Before You Begin
- Gadget Specifications
- Sample Gadgets
- Asset Structure
Before You Begin

Users of this guide must have:

- A developer’s knowledge of Content Server’s basic and flex asset models and templating.
- The Google Gadget API. Related resources are available at the following URLs:
  - The API Reference is available at: http://code.google.com/apis/gadgets/docs/reference/
- Familiarity with OpenSocial Standards, used to create the environment that enables gadget users to set preferences. OpenSocial documentation is available at: http://wiki.opensocial.org/
- Gadget Server sample gadgets installed on Content Server’s FirstSite II sample site. Installation instructions are available in the Gadget Server Installation Guide. Information about managing gadgets can be found in the Gadget Server User's Guide. The product guides are available on our e-docs site: http://support.fatwire.com
  Accounts with FatWire Technical Support can be opened from the home page.
- An understanding of the OAuth protocol, which is used by the sample List Gadget. Information about OAuth is available at the following URLs:
  - The Beginner’s Guide to OAuth is available at: http://oauth.net/

Gadget Specifications

Four sample gadgets were created by FatWire. They are enabled on the FirstSite II sample site. You can develop your own gadgets, using the processes outlined in this guide. The sample gadgets are:

- **List Gadget**, which presents a listing of headlines and article summaries, linking to their respective full articles. This gadget supports OAuth, which means a visitor can authorize the gadget to retrieve her personalized data (in this case, the visitor’s user name and profile picture) from the gadget’s OAuth Service Provider (which is Gadget Server in this example).
- **ThumbList Gadget**, which presents a list of products, with a thumbnail accompanying each product’s description.
- **Slideshow Gadget**, which renders a series of product images into a slideshow, where the user can click on a thumbnail to view a larger image, then click the larger image to open the page containing full details on the product.
- **RSS Feed**, which presents a list of headlines retrieved from an RSS feed. Each headline links to a full article.
The rest of this chapter provides information about the gadgets’ major components and describes the sample gadgets in detail. This chapter also describes the OAuth protocol, and the functionality OAuth enables when it is integrated with Gadget Server and supported by a gadget (List Gadget in this example).

Asset Model and Templates

Installing the sample gadgets on FirstSite II installs the basic components for creating and rendering gadgets. The components are the data model, templates, and sample assets that provide the gadgets’ content:

• **FW_CSGadget** asset type (its description is **CS-Based Gadget**). All sample gadgets are of type **FW_CSGadget**.
• **FW_RSS** asset type (its description is **RSS Feed**). This asset type is used to specify a URL as the source of content for the RSS Feed gadget.
• **FW_CSGadget/GenerateGadgetXML** template, which is accessed by Gadget Server. This template is used to render the gadget descriptor XML (also referred to as gadget specification XML).
• **FW_CSGadget/ListSiteGadgets** template, which provides a gadget descriptor URL for each gadget on the current content management site.

Sample Assets

The sample assets either provide content for the gadgets or they render the gadgets. The sample assets are referenced by the sample gadgets as described below:

• Gadget content is provided by:
  - Content assets of type **Content_C** (with parent of type **Content_P**), representing sports articles. These assets are used by the List Gadget.
  - Product assets of type **Product_C** (with parent of type **Product_P**), representing sports products. These assets are used by the ThumbList and Slideshow gadgets.
  - Media assets of type **Media_C** (with parent of type **Media_P**), representing images used by the Product assets. These assets are used by the ThumbList and Slideshow gadgets.
  - Recommendation (AdvCols) assets, encapsulating the Content and Product assets. Recommendation assets are used by the List, ThumbList, and Slideshow sample gadgets.
  - Content of type **FW_RSS**, which specifies a URL as the source of content for the RSS Feed gadget.

• Templates render the gadgets:
  - The **GenerateGadgetXML** template is accessed by Gadget Server and calls the templates listed below.
  - An **FW_CSGadget**-typed template exists for each of the sample gadgets. Each of these templates outputs the body of a gadget descriptor XML understandable by Gadget Server. The templates are **G_List**, **G_RSS**, **G_Slideshow**, and **G_ThumbList**.
A typed template named \texttt{G\_JSON} exists for each of the asset types referenced by the sample gadgets: \texttt{Content\_C}, \texttt{Media\_C}, \texttt{Product\_C}, and \texttt{Recommendation (AdvCols)}. These templates provide JSON-formatted output containing data necessary to render the HTML for each asset that is displayed in the gadgets. The templates are invoked via remote requests made in the gadget code.

**Auxiliary Files**

The following image files are used by the sample gadgets:

- Scroller arrow images used by the Slideshow gadget (Figure 1, on page 8). These static images are located in the \texttt{FirstSiteII/gadgets} subdirectory under the Content Server web application.

- Images used as icons, thumbnails, and previews to represent a gadget’s various sections. The default images are located in the \texttt{sample/GadgetImages} directory in the \texttt{gadgetserver.zip} installation package.

**Figure 1:** Images used by sample gadgets
OAuth Protocol

The sample List Gadget supports the OAuth protocol. Gadget Server includes OAuth protocol support to enable visitors to personalize OAuth-enabled gadgets. The OAuth protocol communicates in a secure manner to enable a visitor to authorize a gadget to retrieve certain protected personal data from a third-party website by using Gadget Server as a proxy. This is achieved with the help of visitor redirects, handshakes, and digital signatures, as shown in Figure 2.

Figure 2: Personalized data retrieval as seen in the sample List Gadget
OAuth enables the List Gadget to retrieve a visitor’s user name and profile picture from its OAuth Service Provider (local Gadget Server installation in this example), which is specified in the OAuth section of the gadget’s descriptor XML file. Any gadget that is configured to support OAuth must define OAuth parameters in its descriptor XML file.

Gadgets with OAuth support:

- Can be configured to use either Gadget Server-based visitor authentication or third-party based visitor authentication (such as Google, Twitter, and so on).
- Contain an OAuth section in their descriptor XML files. The OAuth section specifies the gadget’s OAuth Service Provider, which is the website from which the gadget requests visitor authentication.

**Note**

Some gadgets with OAuth support store data on one website, but authenticate visitors through another. In this case, the website that stores visitor credentials provides the log in form to Gadget Server. Once the visitor is authenticated, the website that hosts the data exchanges Gadget Server’s session token for the visitor’s requested data.

- Use secure APIs to display a pop-up authentication window from the website the gadget uses for visitor authentication. Once the visitor is authenticated, the gadget securely retrieves that visitor’s personalized data from the website which hosts the data by using Gadget Server as a proxy, without storing the visitor’s credentials on Gadget Server.

All requests that Gadget Server transfers between the gadget and the gadget’s OAuth Service Provider are secured with the following gadget-specific information:

- **Consumer Key** – Also known as an API key, this is a value used by the gadget to identify itself to the OAuth Service Provider.
- **Consumer Secret** – A secret used by the gadget to establish ownership of the consumer key to the OAuth Service Provider.
- **Consumer Signature Method** – The type of digital signature algorithm used to sign requests secured with OAuth (HMAC-SHA1 or RSA-SHA1). The signature process encodes the consumer key and secret into a verifiable value. This prevents unauthorized parties from using the gadget-specific consumer key and secret to access a visitor’s protected resources.

- Include “oauthpopup” as a required feature in the `<ModulePrefs>` tag of their descriptor XMLs, which enables gadgets to access the Gadget Server OAuth library. This library is provided to the gadget automatically by Gadget Server, and when used together with the `gadgets.io.makeRequest` call, enables the gadget to make visitor redirects, handshakes, and digital signatures to securely retrieve a visitor’s personalized data, without storing any of the visitor’s credentials on the local Gadget Server installation.

Developers can create their own gadgets with OAuth support by using JavaScript APIs. For information about the parameters required to configure a gadget to use the OAuth protocol, see Appendix A, “Analyzing Gadget Descriptor XML Files.”

Administrators have permissions to register CS-Based and/or third-party gadgets with OAuth support to Gadget Server. For more information about registering OAuth-enabled gadgets to Gadget Server, see the Gadget Server User’s Guide.
Sample Gadgets

This section outlines the FatWire sample gadgets and the assets they use. Each asset is listed in the same order in which it is called by Gadget Server when Gadget Server initiates the request. This guide uses the List Gadget in particular to illustrate various concepts.

List Gadget

The List Gadget renders a Recommendation asset containing a list view of Content assets. Each headline links to its full article.

The List Gadget asset references the Recommendation asset to be rendered in the gadget and the template (G_List) that will render the gadget. The relevant assets are:

- The ListGadget asset of type FW_CSGadget, referencing the Recommendation asset to be rendered.
- The G_List template, which produces the body of the gadget descriptor XML used by Gadget Server to render the gadget. (This process is outlined in the Chapter 2, “Template Flow.”)
- G_JSON templates, which render the Recommendation asset and its content:
  - AdvCols/G_JSON and Content_C/G_JSON
- The Recommendation (AdvCols) asset to be rendered in the gadget.
- Content assets included in the Recommendation.
ThumbList Gadget

The ThumbList Gadget is similar to the List Gadget in that it also renders a Recommendation asset. The gadget contains a list view of Product assets. Each headline links to its product page. However, in this gadget, each product is accompanied by a thumbnail image from a Media asset, which is associated with the Product assets via their “Image” attribute.

The relevant assets are:

- The ThumbListGadget asset of type FW_CSGadget, referencing the Recommendation to be rendered and the template (G_ThumbList) that will render the gadget.
- The G_ThumbList template which produces the body of the gadget descriptor XML used by Gadget Server to render the gadget.
- G_JSON templates, which render the Recommendation asset and its content: AdvCols/G_JSON, Product_C/G_JSON, and Media_C/G_JSON
- The Recommendation (AdvCols) asset to be rendered in the gadget.
- Product assets included in the Recommendation asset.
- Media assets referenced by the “Image” attribute of each of the Product assets.
Slideshow Gadget

The Slideshow gadget uses the same content as the ThumbList gadget, but presents the content in an entirely different manner. This gadget displays a thumbnail strip filled with product images. Clicking on an image in the strip displays the image at the maximum size allowed by the gadget area. Clicking the full-size image opens the product’s detail page.

The relevant assets are:

- The SlideshowGadget asset referencing the Recommendation asset to be rendered and the template (G_Slideshow) that will render the gadget.
- The G_Slideshow template which produces the body of the gadget descriptor XML used by Gadget Server to render the gadget.
- G_JSON templates which render the Recommendation asset and its content: AdvCols/G_JSON, Product_C/G_JSON, and Media_C/G_JSON
- The Recommendation (AdvCols) asset to be rendered in the gadget.
- Product assets included in the Recommendation asset.
- Media assets referenced by the “Image” attribute of each Product asset.
RSS Feed Gadget

The RSS Feed gadget displays the most recent items from an RSS feed, utilizing the Google Gadget API for its built-in feed-reading functionality.

The relevant assets are:

- The RSSGadget asset of type FW_CSGadget referencing the feed (FW_RSS) to be requested and the template (G_RSS) that will render the gadget.
- The G_RSS template which produces the body of the gadget descriptor XML used by Gadget Server to render the gadget.
- An FW_RSS asset containing the feed URL to be requested in the gadget.

Asset Structure

The sample gadgets are based on a new asset type, FW_CSGadget, which is used to specify information required for rendering the gadgets. It is a basic asset type, containing the following attributes:

- Name of descriptor template is used to specify the name of the FW_CSGadget template that will be invoked to generate the gadget’s descriptor XML. For the List Gadget, the template is FW_CSGadget/G_List (hence, G_List is specified in this field and later resolved as a typed template).
- The DataAsset association is a single, any-type asset association. This association references the asset that provides the main content of the gadget. For example, the DataAsset association for the List Gadget references a Recommendation asset containing a static list of Content assets to be rendered by the gadget.

The descriptor template attribute is read by the FW_CSGadget/GenerateGadgetXML template. The DataAsset association is read by the descriptor template itself (for example, G_List).
Chapter 2  
Template Flow

This chapter describes the flow of template execution, using the List Gadget as an example.

- Template Flow for the List Gadget
- Differences in Template Flow
- Why Server Calls are Done Separately
Template Flow for the List Gadget

By design, the only way to request a gadget’s descriptor XML is by invoking the GenerateGadgetXML template on an FW_CSGadget asset. When the template is invoked, the rendering process begins, as shown in Figure 3 and outlined below:

Figure 3: Gadget XML output

1. GenerateGadgetXML first outputs the XML declaration, a recommended feature for all XML documents. Outputting the XML declaration at the beginning of the process saves gadget developers from having to consider it in every gadget they write.

2. GenerateGadgetXML then loads the asset that corresponds to the passed c (FW_CSGadget) and cid (id of ListGadget) and looks up its descriptor template value.

Note
Together, the c and cid point to a specific asset in the Content Server system. Parameter c is the type of asset and cid is the identifier of the asset. This information is required whenever a template is invoked.

3. GenerateGadgetXML invokes the template in the descriptor template field (FW_CSTemplate/G_List in the case of the List Gadget), passing it the same c and cid as in step 2.

Notice that the G_List template is in itself not externally callable (its SiteCatalog entry’s pageletonly field is set to T). This setting prevents these templates from being called directly (for example, in cases where they are not designed to be called).
4. The called template, `G List`, outputs the entire body of the gadget descriptor XML. At this point, the gadget descriptor XML is complete.

**Note**

Appendix A, “Analyzing Gadget Descriptor XML Files” provides information about the features of the descriptor XML. The template code itself contains comments illuminating important concepts. See “Before You Begin,” on page 6 for links to additional resources regarding the authoring of gadget descriptor XMLs.

Once the gadget is rendered based on its descriptor XML, the gadget retrieves its content. Figure 4 illustrates how the List Gadget retrieves content. This process varies, depending on the gadget. The List Gadget retrieves articles from Content Server via the following JavaScript output by the `G_List` template (this line of code calls a Google Gadgets API function):

```javascript
  gadgets.io.makeRequest(url, handleJson, params);
```

The `G_List` template crafts an additional Content Server URL, pointing to this gadget's `DataAsset` (in this case a Recommendation), via the `G_JSON` template for that asset type. That URL is eventually fed to the `gadgets.io.makeRequest` function, which can be used to make additional remote server calls expecting various forms of data such as XML, JSON, or even Atom/RSS. This request initiates the process of retrieving content.

**Figure 4:** Retrieving content

![Diagram](image)

1. Gadget code in the XML body invokes the `AdvCols/G_JSON` template and prepares the beginning of a JSON response. Since Recommendations contain lists of other assets, the template simply begins the output of a JSON array, then loops through the list of assets, expecting to call a respective `G_JSON` template on each.

2. In the case of the List Gadget, all of the children are Content assets, so `Content_C/G_JSON` is invoked for each asset in the Recommendation. For each invocation, the `Content_C/G_JSON` template outputs a complete JSON object representation containing relevant fields of the asset.
3. In between individual Content_C/G_JSON calls, AdvCols/G_JSON outputs a comma to properly delimit objects in the array it is constructing. After the loop is completed, the AdvCols/G_JSON template closes the array.

4. This array is received by the gadget code (gadgets.io.makeRequest, page 17), which then executes the handleJson callback function as prescribed by the original makeRequest call. The handleJson function wraps the content in HTML, which renders that gadget’s view (a list view in this example).

Differences in Template Flow

The previous section illustrated template flow for the List Gadget. While template flows for the other sample gadgets are very similar, differences in their Content Server template logic are worth noting.

ThumbList and Slideshow

Gadget rendering follows a path in Content Server that is very similar to the path of the List Gadget. The main difference is what is invoked by the AdvCols/G_JSON template. While the List Gadget references a Recommendation filled with Content assets, the ThumbList and Slideshow gadgets reference a Recommendation containing Product assets. These gadgets also inspect attributes specific to the response of the Product_C/G_JSON template (which includes data produced by the Media_C/G_JSON template).

RSS

The RSS Gadget’s second request is fired straight to the URL of the RSS feed, which may or may not be on Content Server. The URL is read from the associated FW_RSS asset, which is loaded within the G_RSS template.

Why Server Calls are Done Separately

You may be asking why an additional server call is always involved. For instance, why not simply retrieve all of the articles for the List Gadget from the G_List template itself? The answer to this is twofold:

1. Gadget descriptor XML may be expected to be cacheable by the gadget container. This means that developers should avoid embedding volatile data in the XML.

2. An additional server call helps to separate presentation logic (for example, in G_List) from the underlying model (in the asset types’ G_JSON templates).
Chapter 3

Creating Your Own Gadgets

- Creating Gadgets on Different CM Sites
- Custom Gadgets
- Prerequisites for Registering Gadgets
Creating Gadgets on Different CM Sites

The sample gadgets must be enabled on FirstSite II. Developers will be interested in creating gadgets on different CM sites. The following steps outline the steps for enabling gadget support on additional CM sites:

1. Enable the FW_CSGadget asset type and its start menu items on the CM site. (By default, the start menu items are accessible to users with the Designer, Site Admin, or General Admin role.)

2. Share the following Template assets to the site:
   FW_CSGadget/GenerateGadgetXML and FW_CSGadget/ListSiteGadgets
   (If you are reusing the sample gadgets, share their templates from FirstSite II, enable the relevant asset types, and if necessary share the assets. For example, if you are reusing the RSS Feed Gadget, share the FW_CSGadget/G_RSS Template asset from FirstSite II and enable the FW_RSS asset type.)

Custom Gadgets

In addition to having a knowledge of sample gadget architecture, it is helpful to see what types of Content Server based gadgets can be created. This section outlines the requirements for creating different types of gadgets.

New Gadget, Content Server Generates Only XML

You may want to create a new gadget where the only additional content is from another website. The RSS Feed Gadget is an example of a gadget that functions in this way. See “RSS Feed Gadget,” on page 14.

Requirements in this scenario are:

- A gadget of type FW_CSGadget referencing the template and asset listed below.
- FW_CSGadget template that generates the body of the gadget descriptor XML (for example, FW_CSGadget/G_RSS).
- An asset that specifies the URL to be requested in the gadget.

New Gadget, Content Server Generates XML and Fields

In this scenario, you will populate a gadget with content directly from one of Content Server’s CM sites. The List, ThumbList, and Slideshow gadgets are examples of gadgets that function in this way. See Figures 3 and 4.

Requirements in this scenario are:

- A gadget of type FW_CSGadget referencing the template and asset listed below.
- FW_CSGadget template (such as G_List) for rendering the body of the gadget descriptor XML.
- Template(s) for rendering data to be returned in response to the additional requests made by the gadget code (for example, the G_JSON templates included with the sample gadgets).
- Assets to be referenced by the gadget (or its additional requests).
Same Gadget Logic, Different Content

You can create gadgets that have the same functionality but pull different sets of content. Requirements in this scenario are:

- An FW_CSGadget asset that references a pre-existing FW_CSGadget template.
- Assets that provide the gadget with content.

Prerequisites for Registering Gadgets

For gadgets to be recognized by Gadget Server, they must be registered (typically by administrators of the Gadget Server application). A gadget is accessed via the Content Server URL that generates the gadget descriptor XML. This URL can be quickly obtained for every FW_CSGadget asset on the content management site by querying the FW_CSGadget/ListSiteGadgets template. The template can be easily accessed in one of the following ways:

- In the Content Server Advanced interface, preview any FW_CSGadget on the site. In the InSite window, select ListSiteGadgets from the “Template” drop-down menu.
- Or -
- Open a browser and navigate directly to http://<host>:<port>/<application context>/wem/<sitename>/FW_CSGadget/ListSiteGadgets (where host, port, and application context correspond to the Content Server installation, and sitename is the name of the content management site where the gadget exists).

The list of sample gadgets and their corresponding URLs is shown in the following figure.

| ListGadget | http://10.120.19.73:8080/cs/ContentServer?c=FW_CSGadget&pgename=FirstSiteI%2FFW_CSGadget%2FGenerateGadgetXML&cid=1269873534992 |
| RSSGadget  | http://10.120.19.73:8080/cs/ContentServer?c=FW_CSGadget&pgename=FirstSiteI%2FFW_CSGadget%2FGenerateGadgetXML&cid=1269873535165 |
| Slideshow  | http://10.120.19.73:8080/cs/ContentServer?c=FW_CSGadget&pgename=FirstSiteI%2FFW_CSGadget%2FGenerateGadgetXML&cid=1269873535725 |
| ThumbListGadget | http://10.120.19.73:8080/cs/ContentServer?c=FW_CSGadget&pgename=FirstSiteI%2FFW_CSGadget%2FGenerateGadgetXML&cid=1269873553620 |
Appendix A

Analyzing Gadget Descriptor XML Files

Each gadget is defined by a descriptor XML. This appendix provides information about the parameters defined in the descriptor XMLs of the sample CS-Based gadgets.

This appendix contains the following:

- Sample RSSFeed Gadget
- Sample List Gadget
Sample RSSFeed Gadget

A gadget’s descriptor XML contains all the data defined for the gadget within the Module tag. This tag holds information about the gadget’s dependencies, user preferences (if any), appearance settings, functionality, and so on.

This section analyzes CS-Based gadget asset descriptor XML files, using code snippets from the sample RSSFeed gadget as an example.

Analyzing the code of the RSSFeed Gadget’s Descriptor XML File

These lines specify the properties and dependencies of the gadget:

```xml
<ModulePrefs title="FatWire RSS" height="350">
    <screenshot="http://localhost:8100/cs/FirstSiteII/gadgets/RSS/screenshot.png"/>
    <thumbnail="http://localhost:8100/cs/FirstSiteII/gadgets/RSS/thumbnail.png"/>
    <Require feature="dynamic-height"/>
</ModulePrefs>
```

The following lines define the gadget’s user preferences, which are values for the gadget that visitor’s can modify. In this code snippet, visitors will be able to select the amount of news feeds that the RSSFeed gadget displays at one time:

```xml
<UserPref name="max" display_name="Number of Items" datatype="enum" default_value="5">
    <EnumValue value="1" display_value="1"/>
    <EnumValue value="3" display_value="3"/>
    <EnumValue value="5" display_value="5"/>
    <EnumValue value="10" display_value="10"/>
</UserPref>
```

The Content tag contains all of the content for the RSSFeed gadget’s descriptor XML file, including the gadget’s CSS file, which defines the gadget’s appearance, and the gadget’s JavaScript (contained within the <script> tag), which specifies the interactive components and initialization functionality of the gadget.

For example, the CSS file defined within the RSSFeed gadget’s Content tag looks as follows:

```css
/* === generic styles === */
body #container {
    padding: 15px;
    padding-bottom: 5px; /* 5 plus 10 from bottom-most entry */
    margin: 0;
}
body #container * {
    padding: 0;
    margin: 0;
    font-size: 11px;
    font-family: Tahoma,Arial,Helvetica,sans-serif;
    color: #666;
}
body #container a, body #container * a {
    text-decoration: none;
```
color: #555;
}
body #container a:hover, body #container * a:hover {
    color: #3b9cce;
}
.error {
    background-color: #fcc;
    color: #c00;
}
/* === gadget-specific styles === */
#container .list {
    padding-left: 1em; /* provide proper indentation space for bullets */
}
#container .list li {
    padding-bottom: 10px;
}
#container .headline {
    font-weight: bold;
}
</style>
Sample List Gadget

The sample List Gadget supports the OAuth protocol. A gadget with OAuth support contains an OAuth section within the ModulePrefs tag of its descriptor XML. This tag specifies the following three endpoints related to OAuth operations:

Table 1: OAuth URL endpoints

<table>
<thead>
<tr>
<th>URL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Token URL</td>
<td>Initiates the authentication process by the gadget and issues the request token. The request token is a temporary token which is active only during the authentication phase. As soon as the visitor’s credentials are validated and she is logged in, the request token is exchanged for an access token.</td>
</tr>
<tr>
<td>Access Token URL</td>
<td>Retrieves the access token in exchange for the request token.</td>
</tr>
<tr>
<td>Authorization URL</td>
<td>The location that will be opened in a pop-up window to display the authentication form.</td>
</tr>
</tbody>
</table>

Table 2: Pre-approved request tokens

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gs_request_token</td>
<td>The pre-approved token that is passed to the gadget. The gadget can use this to start an OAuth session. It is a short-lived, single-use token with a 30 second lifespan.</td>
</tr>
<tr>
<td>gs_request_token_secret</td>
<td>The secret for the pre-approved request token, which is also needed to start the OAuth session.</td>
</tr>
</tbody>
</table>

The rest of this section analyzes code snippets from the sample List Gadget’s descriptor XML that pertain to OAuth functionality.

Analyzing OAuth parameters within the sample List Gadget’s code snippets

The <ModulePrefs> tag includes "oauthpopup" as a required feature to demonstrate OAuth support for this gadget:

```xml
<ModulePrefs title="Latest News" height="350">
  <Require feature="dynamic-height"/>
  <Require feature="oauthpopup"/>
</ModulePrefs>
```

These lines define the OAuth section, which is contained in the ModulePrefs tag:

```xml
<OAuth>
  <Service name="gs">
    <Request url="http://10.120.19.25:8480/user-service/request_token"/>
    <Access url="http://10.120.19.25:8480/user-service/access_token"/>
  </Service>
</OAuth>
```
This line, located within the fetchData JavaScript function, specifies the URL to which OAuth requests fire:

```javascript
var url = userServiceUrl + '/echo?siteId=' + prefs.getString('gs_site_id');
```

These lines specify information about the nature of the request. Gadget Server behaves accordingly based on these values. The OAUTH_SERVICE_NAME parameter specifies the service, defined in the <OAuth> section, from which data is requested:

```javascript
var params = {};
params[gadgets.io.RequestParameters.CONTENT_TYPE] = gadgets.io.ContentType.JSON;
params[gadgets.io.RequestParameters.AUTHORIZATION] = gadgets.io.AuthorizationType.OAUTH;
params[gadgets.io.RequestParameters.METHOD] = gadgets.io.MethodType.GET;
params[gadgets.io.RequestParameters.OAUTH_SERVICE_NAME] = "gs";
```

This line loads the pre-approved request token. Pre-approved request tokens are valid only once:

```javascript
var requestToken = prefs.getString('gs_request_token');
```

These lines specify the pre-approved request token. For more information about the request tokens, see Table 2:

```javascript
params[gadgets.io.RequestParameters.OAUTH_REQUEST_TOKEN] = requestToken;
params[gadgets.io.RequestParameters.OAUTH_REQUEST_TOKEN_SECRET] = prefs.getString('gs_request_token_secret');
params[gadgets.io.RequestParameters.OAUTH_USE_TOKEN] = "always";
```

The remaining code in the fetchData function makes a request to the OAuth Service Provider, and defines the callback function which will handle the response once it is received:

```javascript
gadgets.io.makeRequest(url, function (response){...},params)
```

The if conditions within the callback function define the three possible outcomes when a request is made and the OAuth signature is enabled:

- If authentication is needed, the response object contains an oauthApprovalUrl property, which can be used to open the authentication pop-up window containing the login form.

```javascript
if (response.oauthApprovalUrl) {
    var onOpen = function () {
        showOneSection('waiting');
    };
    var onClose = function () {
        fetchData();
    };
```
Appendix A. Analyzing Gadget Descriptor XML Files

Sample List Gadget

```javascript
var popup = new gadgets.oauth.Popup(response.oauthApprovalUrl,
    null, onOpen, onClose);
$('personalize').onclick = popup.createOpenerOnClick();
$('approvaldone').onclick = popup.createApprovedOnClick();
showOneSection('approval');
}

• If the authentication is successful or the visitor has already been authenticated, the
  response object contains the data property. The data property contains the server’s
  response for the requested URL.
else if (response.data) {
  var res = response.data;
  $('main').innerHTML = '<div><div style="display:inline;">' +
    userServiceUrl + res.userpicurl + '"></div><div style="margin-left:20px;display:inline;font-
  size:14pt;font-weight:bold;">Hello, " + res.displayname +
    "</div></div>
showOneSection('main');

This line clears the actual gadget’s content and (re-)issues the request for the gadget’s
  data, now that the user is authorized:
$('container').innerHTML = "";
makeRequest();
}

• These lines specify the oauthError property which is populated if there is an OAuth
  protocol error:
else {
  var errmsg = document.createTextNode('OAuth error: ' +
    response.oauthError );
  $('main').appendChild(errmsg);
showOneSection('main');
}
```