

How to generate a widget code

The FLUX Widget Developer Kit (WDK) gives you the ability to create widgets to place on your own site. As a community admin you can navigate to the Widgets page via the Community Manager (through quick menu) -> Flux Widgets. Then, simply select the appropriate widget you would like to put on your own page. As a result you will get to a page where you can customize parameters of the widget with a real time preview of how your widget will look. The screenshot on the next page shows an example of how the Members widget pick-up page looks.

Below the customizable properties there is a text area that is automatically updated each time you change a particular property. This is the actual widget code. To have a widget on your page you should copy/paste this java script code onto your page at the place where you would like widget to appear.

Additionally, there is a Get Original HTML button available which when clicked will display the original HTML markup for the widget. This allows deeper customization of the widget and is outlined later in this document in the section titled "Markup Customization"

Important: as you can see there are two external java script files included in this code – Runtime.js and Context.js. Each of them contains code for the FLUX widget framework which actually implements the widget. **You have to place these includes inside of the <body> tag of your page not in the <head>.** If you want to use more than one FLUX widget on your page you only need include these files once. The actual script that creates the widget is the third script. This passes all parameters to the framework and also indicates the actual place on the page where the widget will appear.

Once you place the widget on your page you can customize its behavior and UI by directly changing the parameters' values within the *createWidget* method. If you are not sure how to change them directly, you can always return back to the widget pick-up page described above and setup its properties once again and generate a new code.

Placing widgets

It is important to note that specific widgets are intended to always be bound to a content item, and will not function correctly unless the permalink/URI of the piece of content to which they are bound is passed to the widget. The fShare, Comments, Content Action and Usage widgets all require this parameter be correctly passed.

For more guidance on placing widgets on your site, see the Widget Integration Guide in the resources section of the Flux Developer site.

Customize a theme to match the look and feel of your site.
When you are happy with the appearance of your Widget, copy the code and add it to your site.

Widget Title

View mode

Item count

Image size

Width

Font

Header Color

Header Font Color

Header font size

Show member login area

Login Color

Login Font Color

Login Link Color

Login font size

Background Color

Font Color

Link Color

Text font size

Footer Color

Border width

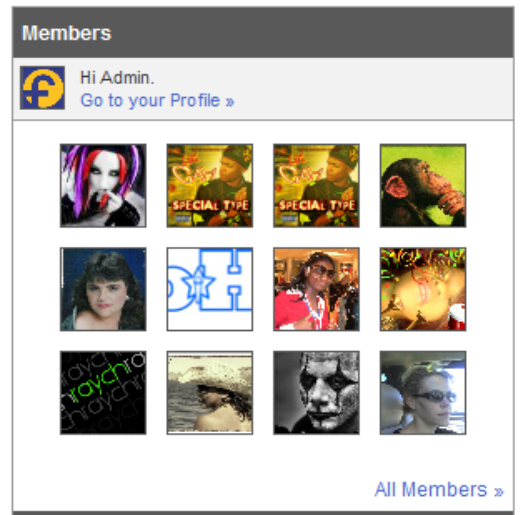
Border color

Border style

```
communityUcid=E2F6FFF0099C774002FFFF6E2&disableCommunityCommon
Css=on" type="text/javascript"></script>
<script type="text/javascript">
    Flux.Util.WidgetsLoader.createWidget(
        'CommunityMembers',
        {accessKey: 'e4cab612-ec98-49be-8527-
53df1be5b49b',title: 'Members',viewMode: 'icons',count: 12,imageSize:
copy to clipboard
```

Original markup:

```
<style type="text/css">
    .Flux .communityMembers {
        border: [STYLE_BORDER_STYLE]
[STYLE_BORDER_WIDTH] [STYLE_BORDER_COLOR];
        overflow: hidden;
```



Advanced Customization

Overview

The WDK allows for advanced customization of all widgets in addition to what is provided via the customizable property editor in the widget pick-up page described above. This functionality can be used to customize the HTML markup of the widget, and also to customize its' underlying logic contained in the widgets' java script. By leveraging this mechanism it's possible to have full control over the UI and functionality of the widget, ranging from some basic changes to the HTML mark-up, to completely changing the widget's behavior if desired.

This customization is achieved by passing an additional parameter to the *createWidget* method of the widget as indicated in the example below for a sample members widget:

```
<script type="text/javascript">
    Flux.Utils.WidgetsLoader.createWidget(
        'CommunityMembers',
        {accessKey: 'e4cab612-ec98-49be-8527-53df1be5b49b',title: 'Members',viewMode:
'icons',count: 10,imageSize: Flux.API.ImageSize.Small,width: '160px',textFont: 'Arial',headerColor:
'#595959',textHeaderColor: '#ffffff',headerFontSize: '12px',loginShowArea: true,loginColor:
'#F2F2F2',loginTextColor: '#333333',loginLinkColor: '#4866C8',loginFontSize:
'11px',backgroundColor: '#FFFFFF',textColor: '#333333',linkColor: '#4866C8',textFontSize:
'12px',footerColor: '#595959',borderWidth: '1px',borderColor: '#999999',borderStyle: 'solid',
customize: {markupUrl: 'http://[FILE_LOCATION]/CustomMarkup.html', scriptUrl:
'http://[FILE_LOCATION]/CustomJSLogic.js'}
        }
    );
</script>
```

The important part is highlighted in green. Notice that the *customize* parameter specifies 2 values – one for the custom HTML mark-up and the other for the custom java script logic. It's possible to pass either *markupUrl* only, *scriptUrl* only, or both depending on which part of the widget is being customized. Techniques for customizing the markup and java script logic of widgets are detailed in the following 2 sections.

Markup Customization

As mentioned above, you can use this type of customization if you find that the existing widget markup doesn't suit all your needs, even after customizing all available parameters. In this case you may redesign the FLUX native markup of the widget to fit your particular needs. You need to have the original markup of the widget before trying to customize it. To grab the original markup, click the "Original markup" button on the pick-up page for the specific widget you want to customize. This will show a text area containing the original markup for the widget. Below is an example of the original widget markup for a Community Menu widget:

```
<style type="text/css">
```

```

.Flux .communityMenu2,
.Flux .communityMenu2 a,
.Flux .communityMenu2 li {
    font-family: [TEXT_FONT];
}

REMAINING CSS STYLES FOR WIDGET...
</style>

<div id="content" class="communityMenu2">
{items}
    <div id="Div1" class="menuBar">
        <span id="menuPlaceHolder"></span>
    </div>
{/items}
{menuExtHeader}
    <ul class="horizontalDropdownMenu">
{/menuExtHeader}
{menuExtItem}
    <li class="menuItem menuItemNum_{ITEM_INDEX}"><a
href="{LINK_URL}" {LINK_ATTRIBUTES}><span
class="menuItemText">{LINK_TEXT}</span></a>{SUB_ITEMS}</li>
{/menuExtItem}
{menuExtFooter}
    </ul>
    <div style="clear: both"></div>
{/menuExtFooter}

{menuExtSubHeader}
    <ul class="horizontalDropdownMenu">
{/menuExtSubHeader}
{menuExtSubItem}
    <li class="submenuItem"><a href="{LINK_URL}" {LINK_ATTRIBUTES}><span
class="menuItemText">{LINK_TEXT}</span></a></li>
{/menuExtSubItem}
{menuExtSubFooter}
    </ul>
{/menuExtSubFooter}

{menuSimpleHeader}
    <table cellpadding="0" cellspacing="0" border="0">
{/menuSimpleHeader}
{menuSimpleItem}
    <td class="{SIMPLE_TD_CLASS}">
        <a href="{LINK_URL}" {LINK_ATTRIBUTES}><span
class="menuItemText">{LINK_TEXT}</span></a>
    </td>
{/menuSimpleItem}
{menuSimpleFooter}
    </table>
{/menuSimpleFooter}
</div>

```

This markup for the widget contains 3 important parts:

1. **CSS Styles and Style Substitutions.** Note – the sample markup here does not display all styles, but just an example. Notice that the CSS styles section contains a number of style substitutions contained in [] brackets (i.e. [TEXT_FONT]). This indicates external parameters which are customized via the widget pick-up page and this value will be dynamically replaced with the parameter from the *createWidget* method. You can customize existing styles as desired, and add new styles as necessary. For a reference of existing CSS Style Substitutions for each widget, refer to appendix B

Also, the style substitutions can be replaced with hard-coded values without adverse effect, although it's advised not to do this. Obviously it would mean that the parameters passed in the *createWidget* method are no longer used if you do this.

2. **Templates.** The markup is built using templates. Templates are denoted by {*TEMPLATE_NAME*}...{/*TEMPLATE_NAME*}, for example {*items*}...{/*items*}. The HTML markup of the widget is contained within each template.

Different templates are used depending upon certain conditions. In the example of the Community Menu widget there are different templates for a 1-level dropdown (simple) vs. a menu containing dropdowns (ext). Also, in some widgets there are different templates depending upon the authentication state of the user (logged-in member, logged-in non-member, logged-out guest).

See Appendix C for a reference guide on templates used for each widget. It is strongly recommended not to remove any of the templates in the widget markup. Doing so may result in unexpected behavior. Use the reference guide in the Appendix to understand which specific templates need modifying when trying to customize the widget. Unused templates can be left unchanged.

3. **Markup Substitutions.** Within the template you will see additional markup substitutions, contained within { }, such as {LINK_URL}. These substitutions are dynamically replaced at runtime with the actual data for the widget. For example, {LINK_URL} will be replaced with the actual URL of the menu item.

Appendix D contains a reference guide of the available markup substitutions for each widget. You can use any of these substitutions in the mark-up as necessary to inject the actual data. Note, it is also possible to create your own markup substitutions for widgets by customizing the java script of the widget. This is described in detail in the "Creating custom markup substitutions" later in this document.

Here is sample HTML that will be generated for a simple community menu (i.e. 1-level without dropdowns) using this default markup:

```
<style type="text/css">
.Flux .communityMenu2,
.Flux .communityMenu2 a,
.Flux .communityMenu2 li {
    font-family: arial;
}
```

REMAINING CSS STYLES FOR WIDGET...

```

</style>

<div class="Flux">
  <div id="CommunityMenu2_0" class="Widget">
    <div id="CommunityMenu2_0_content" class="communityMenu2">
      <div id="Div1" class="menuBar">
        <span id="CommunityMenu2_0_menuPlaceholder">
          <table cellspacing="0" cellpadding="0" border="0">
            <tbody>
              <tr>
                <td class="menuItem menuItemNum_0 menuItemFirst">
                  <a href="http://developer.flux.com/-/Home"><span
class="menuItemText">Home</span> </a>
                </td>
                <td class="menuItem menuItemNum_1">
                  <a href="http://developer.flux.com/-/Documentation"><span
class="menuItemText">Documentation</span>
                  </a>
                </td>
                <td class="menuItem menuItemNum_2">
                  <a href="http://developer.flux.com/-/Examples"><span
class="menuItemText">Examples</span>
                  </a>
                </td>
              </tr>
            </tbody>
          </table>
        </span>
      </div>
    </div>
  </div>
</div>

```

To customize the mark-up of the widget it is necessary to copy and paste the default markup into your own custom.html file. Then, simply modify the mark-up in the appropriate templates to apply all necessary changes (reference Appendix C for details on templates). For example, if you would like your menu widget to call your own specific java script function each time when a user clicks any item the following simple change can be made. Note - In this case, we are only going to modify the {menuSimpleItem} template since we are only concerned with the 1-level menu. If we were creating a dropdown menu then we would need to change the {menuExtItem} and {menuExtSubItem} templates. Notice the part of the template highlighted in green. This is the new code we added to call our custom function.

```

{menuSimpleItem}
    <td class="{SIMPLE_TD_CLASS}">
      <a href="{LINK_URL}" {LINK_ATTRIBUTES} onclick="fnMyMethod(param1, param2,
param3); return true;"><span class="menuItemText">{LINK_TEXT}</span></a>
    </td>
{/menuSimpleItem}

```

Once all customization of the markup is completed, reference the section “Applying custom markup and java script to your widget” to update the widget code to reference this markup.

Java script Logic Customization.

Each widget has an associated java script file that contains the logic for loading data, handling widget actions, rendering widget markup. You might need to customize this logic for your own needs. Note that this file contains the core code for the widget, so exercise caution when changing it to avoid un-expected results.

There are a number of reasons why you may need to change the java script for the widget, but one good example is to customize the rating behavior in a content action widget. By default the content action widget supports only thumbs up so to allow thumbs up and thumbs down it’s necessary to customize the java script logic. Refer to the Widget Examples page on developer.flux.com for some specific code samples.

To customize the java script logic of a widget it is first necessary to get the default java script for that specific widget. You can get this file from the widget pick-up page for the specific widget by clicking the “Original java script” link. This will allow you to download the java script as a reference. When customizing the java script logic of a widget it is necessary to create a new .js file to contain your custom code.

Important – it is only necessary to add your customized java script methods to this .js file. You should not copy and paste the entire default java script. Instead, only copy and paste the methods you wish to change. If you were to copy and paste all methods then any updates made to the core widget logic will not be inherited by your widget.

To modify existing methods simply override them in your custom .js file. You can also add as many custom methods as necessary. Below are instructions for creating a custom java script file to override the default behavior of a widget:

1. Create a separate java script file and copy/paste only the necessary method(s) you want to override from the original java script file
2. It is necessary to make your custom methods override the native methods of the widget. To do this you need to wrap each method you override with the following java script construction:

```
Flux.Widgets.<WIDGET_NAME>.prototype.<METHOD_NAME> = function([parameters]) {  
    <CUSTOMIZED_METHOD_CODE>  
};
```

Where, <WIDGET_NAME> is the Flux WDK widget name (see Appendix A for reference)

<METHOD_NAME> is the name of the existing method being overridden

Note - you can override multiple methods of a single widget in one file. However, do not override methods of multiple widgets in a single java script file. Each widget must use it's own custom java script file.

3. Update the java script code in the methods being overridden
4. Add any other additional java script as necessary in your custom .js file

Below is a simple example of how to override a method in the java script logic for the Content Action widget. In this case, we are just going to change the caption that is dynamically displayed after the user has clicked the "Add to Favorites" button. The following code is the original handler for the event in the default widget code:

```
addContentToUserFeedHandler: function(result, instance, methodName) {
    if (result.StatusCode == Flux.API.StatusCode.Ok) {
        instance._addToMyProfileButton.innerHTML = 'Added!';
    } else {
        Flux.Debug.traceServiceError('UserFeed', methodName, result);
    }
}
```

The following code should be added to the custom java script file for this widget to override this behavior to display "Done" instead of "Added!" when the user clicks the button:

```
Flux.Widgets.ContentAction2.prototype.addContentToUserFeedHandler = function(result, instance,
methodName) {
    if (result.StatusCode == Flux.API.StatusCode.Ok) {
        instance._addToMyProfileButton.innerHTML = 'Done';
    } else {
        Flux.Debug.traceServiceError('UserFeed', methodName, result);
    }
};
```

Once all customization of the java script is completed, reference the section "Applying custom markup and java script to your widget" to update the widget code to reference this custom java script.

Applying custom markup and java script to your widget

After you have finished customizing the markup and/or java script for your widget the following steps should be followed to apply this customization to the widget:

1. Host the .html and/or .js file(s) on a publicly accessible location. You can use your own hosting for this, or you can use your Flux FTP access to upload to the Flux static servers (please refer to the separate document "Using FTP to access Static Servers" on developer.flux.com).
2. **Important - if you host the customized file(s) on your own servers, it is required that there is a crossdomain.xml file in the root path of that domain and it contains following lines. If this crossdomain.xml file is not present or is not configured correctly, the widget will fail to load correctly.**

```
<allow-access-from domain="*" />
<allow-http-request-headers-from domain="*" headers="*" />
```

3. Update the widget code used to place the widget on your site to include the *customize* parameter as described in the overview, using the format below:

```
customize: {markupUrl: 'http://[FILE_LOCATION]/CustomMarkup.html', scriptUrl:
'http://[FILE_LOCATION]/CustomJSLogic.js'}
```

Where [FILE_LOCATION] specifies the absolute path of the custom files on either your own server, or Flux servers.

Note – you can name the .html and .js files whatever you want, but ensure to reference correctly in the *customize* parameter here.

Creating Custom Markup Substitutions

It is possible to use a combination of custom markup and custom java script logic for your widget to introduce your own templates for substitutions. To do this you have to choose a name for your substitution template, for example {MY_SPECIAL_TEMPLATE}. Then you have to create a special piece of code in your custom java script logic around this template, and override the render2() method of the widget (or any other method that best suits your need to use custom substitution template).

Here is an example of what you can do with it:

```
if(<some_condition>)
  text = text.replace(/\{MY_SPECIAL_TEMPLATE\}/g, <some_value>);
else
  text = text.replace(/\{MY_SPECIAL_TEMPLATE\}/g, <another_value>);
```

where, *text* variable contains some text with your substitution template injected inside.

This markup substitution can then be referenced within your custom markup by inserting {MY_SPECIAL_TEMPLATE} wherever you want to dynamically insert this value.

Asynchronous Widget Refresh

The WDK has built-in support for asynchronous refresh of widgets via the “Event Manager”. This allows all widgets on a page to be asynchronously refreshed by calling the following javascript method:

```
Flux.EventManager.update()
```

This is useful on sites which are using Flux API-based authentication. For example, if a user logs in via the Flux API on your site, by calling this method after the user logs in all widgets will be asynchronously refreshed. This means that widgets which have different states dependant on the user’s authentication status will correctly reflect the user’s new state.

Additionally, it is important to note that this asynchronous refresh happens automatically when interacting with widgets, so there is no need for manual refresh except in the use-case above when using the Flux API to login/logout users. So, for example if you have two widgets on a page: Comments and Content Action and they are bound to the same content item. The Content Action widget shows number of comments left for this content and the Comments widget also shows this number. Now, if a comment is left on the content via the Comments widget the number of comments in that widget will be increased. Also, the Event Manager update method is automatically fired in this instance so the Content Action widget now displays the correct count also.

Appendix A. Available Widgets

The WDK Widget Name in the table below denotes the value to be used when referencing this widget in your java script code.

Widget	WDK Widget Name
fShare	FShare2
Members	CommunityMembers

Videos	SimpleVideos
Photos	SimplePhotos
Comments	Comments3
Content Action	ContentAction2
Usage	Usage
Community Menu	CommunityMenu2
Quick Menu	QuickMenu2

Appendix B. CSS Style Substitutions

This section lists the Style Substitutions used by each widget. The values for these styles are passed directly via the widget parameters, which are configured on the widget pick-up page or can be changed directly in the widget code. The CSS in the default markup for each widget uses these pre-defined substitutions.

Widget/ Template	Members	Videos	Photos	Comments	Content Action	Usage	Menu	Quick Menu
BACKGROUND				X				
COMMENT_BACKGROUND				X				
COMMENT_BORDER_STYLE				X				
COMMENT_BORDER_COLOR				X				
COMMENT_BORDER_WIDTH				X				
WIDTH	X	X	X					
STYLE_BORDER_STYLE	X	X	X					
STYLE_BORDER_WIDTH	X	X	X					
STYLE_BORDER_COLOR	X	X	X					
STYLE_WIDTH	X	X	X					
LOGIN_COLOR	X	X	X					
LOGIN_LINK_COLOR	X	X	X					
LOGIN_FONT_SIZE	X	X	X					
LOGIN_TEXT_COLOR	X	X	X					
FOOTER_COLOR	X	X	X					
HEADER_COLOR	X	X	X					

TEXT_HEADER_COLOR	X	X	X					
TEXT_FONT	X	X	X				X	
HEADER_FONT_SIZE	X	X	X					
TITLE	X	X	X					
BACKGROUND_COLOR	X	X	X					
TEXT_FONT_SIZE	X	X	X				X	
TEXT_COLOR	X	X	X	X				X
NORMAL_BACKGROUND_COLOR							X	
BORDER_COLOR							X	X
BORDER_STYLE							X	X
NORMAL_TEXT_COLOR							X	
HOVER_BACKGROUND_COLOR							X	
HOVER_TEXT_COLOR							X	
SIMPLE_TD_CLASS							X	
STYLE_TEXT_FONT				X				X
STYLE_FONT_SIZE				X				X
LINK_COLOR	X			X				X
MENU_BACKGROUND								X
MENU_ITEM_BACKGROUND								X

Appendix C. Markup Templates

This section details the templates used in each widget, and the purpose of each template.

fShare Widget

None

Members Widget

Template Name	Description
LoginNotAuthorize	Only displayed for guests (user is not logged-in)
LoginNotMember	Only displayed for logged-in users who are not a member of the community
LoginMember	Only displayed for logged-in users who are a member of the community
IconModeHeader	Header section of widget displayed for view mode "icons"
IconModeItem	Repeating body section of widget displayed for view mode "icons"

IconModeFooter	Footer section of widget displayed for view mode "icons"
ListModeHeader	Header section of widget displayed for view mode "list"
ListModeItem	Repeating body section of widget displayed for view mode "list"
ListModeFooter	Footer section of widget displayed for view mode "list"
NoItems	Only displayed if there are no members to display in the widget

Videos Widget

Template Name	Description
LoginNotAuthorize	Only displayed for guests (user is not logged-in)
LoginNotMember	Only displayed for logged-in users who are not a member of the community
LoginMember	Only displayed for logged-in users who are a member of the community
IconModeHeader	Header section of widget displayed for view mode "icons"
IconModeItem	Repeating body section of widget displayed for view mode "icons"
IconModeFooter	Footer section of widget displayed for view mode "icons"
ListModeHeader	Header section of widget displayed for view mode "list"
ListModeItem	Repeating body section of widget displayed for view mode "list"
ListModeFooter	Footer section of widget displayed for view mode "list"
NoItems	Only displayed if there are no members to display in the widget
FooterNotLoggedIn	Footer area only displayed for guests (user is not logged-in)
FooterNotMember	Footer area only displayed for logged-in users who are not a member of the community
FooterMember	Footer area only displayed for logged-in users who are a member of the community

Photos Widget

Template Name	Description
LoginNotAuthorize	Only displayed for guests (user is not logged-in)
LoginNotMember	Only displayed for logged-in users who are not a member of the community
LoginMember	Only displayed for logged-in users who are a member of the community
IconModeHeader	Header section of widget displayed for view mode "icons"
IconModeItem	Repeating body section of widget displayed for view mode "icons"
IconModeFooter	Footer section of widget displayed for view mode "icons"
ListModeHeader	Header section of widget displayed for view mode "list"
ListModeItem	Repeating body section of widget displayed for view mode "list"
ListModeFooter	Footer section of widget displayed for view mode "list"
NoItems	Only displayed if there are no members to display in the widget
FooterNotLoggedIn	Footer area only displayed for guests (user is not logged-in)
FooterNotMember	Footer area only displayed for logged-in users who are not a member of the community
FooterMember	Footer area only displayed for logged-in users who are a member of the community

Comments Widget

Template Name	Description
CommentEditor	Only displayed to users who are authorized to post a comment on the content
HeaderMemberView	Header section only displayed for logged-in users who are a member of the community
HeaderNotLoggedInView	Header section only displayed for guests (i.e. not logged-in)
HeaderNotMemberView	Header section only displayed for logged-in users who are not a member of the community
HeaderNotVerifiedView	Header section only displayed for logged-in community members with an un-verified email address
HeaderPageLinkPrevious	Container for "Previous" pagination link in header which is displayed as necessary
HeaderPageLinkNext	Container for "Next" pagination link in header which is displayed as necessary
HeaderPageLinkSeparator	Container for separator for pagination link in header which is displayed as necessary
CommentActionReply	Container for comment "Reply" link which is displayed as necessary
CommentActionEdit	Container for comment "Edit" link which is displayed as necessary
CommentActionDelete	Container for comment "Delete" link which is displayed as necessary
CommentActionFlag	Container for comment "Flag" link
CommentActionSeparator	Container for separator between action links for comment
CommentImageViewSimple	Only displayed if comment creator doesn't have a profile URL
CommentImageViewAsLink	Only displayed if comment creator has a Profile URL
CommentImageViewNone	Only displayed if community has chosen to hide thumbnails of comment creator via community settings
CommentNameViewSimple	Only displayed if comment creator doesn't have a profile URL
CommentNameViewAsLink	Only displayed if comment creator has a Profile URL
CommentItem	Contains the actual comment with creator image, name, date and message
FooterPageLinkPrevious	Container for "Previous" pagination link in footer which is displayed as necessary
FooterPageLinkNext	Container for "Next" pagination link in footer which is displayed as necessary
FooterPageLinkSeparator	Container for separator for pagination link in footer which is displayed as necessary

Content Action Widget

Template Name	Description
AddToMyProfileConfirm	Container for confirmation text displayed when user interacts with content action widget

Usage Widget

None

Community Menu Widget

Template Name	Description
Items	Container for all items in the menu

MenuSimpleHeader	Header section of widget displayed for 1-level menu only
MenuSimpleItem	Repeating body section of widget displayed for 1-level menu only
MenuSimpleFooter	Footer section of widget displayed for 1-level menu only
MenuExtHeader	Header section of widget displayed for top-level items of dropdown menu only
MenuExtItem	Repeating body section of widget displayed for top-level items of dropdown menu only
MenuExtFooter	Footer section of widget displayed for top-level items of dropdown menu only
MenuExtSubHeader	Header section of widget displayed for sub-level items of dropdown menu only
MenuExtSubItem	Repeating body section of widget displayed for sub-level items of dropdown menu only
MenuExtSubFooter	Footer section of widget displayed for sub-level items of dropdown menu only

Quick Menu Widget

Template Name	Description
NotLoggedInView	Only displayed for guests (user is not logged-in)
NotMemberView	Only displayed for logged-in users who are not a member of the community
MemberView	Only displayed for logged-in users who are a member of the community
MenuHeader	Header section of dropdown menu in widget
MenuItem	Repeating body section of dropdown menu in widget which displays menu items
MenuFooter	Footer section of dropdown menu in widget
MenuSeparator	Separator between items in dropdown menu in widget

Appendix D. Pre-defined Markup Substitutions

This section lists the pre-defined markup substitutions available for each widget. These substitutions can be referenced in the markup for the specific widget to dynamically inject data for the widget.

fShare Widget

None

Members Widget

Substitution Name	Description
TITLE	Widget title specified in the widget parameters ("title" param)

USER_PROFILE	Profile URL of logged-in user.
USER_NAME	Display Name of logged-in user. Shows "Guest" for logged-out users
USER_IMAGE	Image tag for Thumbnail of logged-in user
URL_JOIN_COMMUNITY	URL for community login/signup page
URL_LOGIN	URL for community login/signup page
URL_SIGNUP	URL for community login/signup page
URL_LANDING_PAGE	URL for community members page
USER_ITEM_NAME	Display Name of member in list
USER_ITEM_DETAIL_URL	Profile URL of member in list
USER_ITEM_IMAGE_URL	Thumbnail URL of member in list
USER_UCID	Flux user UCID of member in list

Videos Widget

Substitution Name	Description
TITLE	Widget title specified in the widget parameters ("title" param)
USER_PROFILE	Profile URL of logged-in user.
USER_NAME	Display Name of logged-in user. Shows "Guest" for logged-out users
USER_IMAGE	Image tag for Thumbnail of logged-in user
URL_JOIN_COMMUNITY	URL for community login/signup page
URL_LOGIN	URL for community login/signup page
URL_SIGNUP	URL for community login/signup page
URL_LANDING_PAGE	URL for community members page
CONTENT_TITLE	Title of video in list
USER_ITEM_DETAIL_URL	Thumbnail URL of video in list
USER_ITEM_IMAGE_URL	Content Detail URL of video in list

Photos Widget

Substitution Name	Description
TITLE	Widget title specified in the widget parameters ("title" param)
USER_PROFILE	Profile URL of logged-in user.
USER_NAME	Display Name of logged-in user. Shows "Guest" for logged-out users
USER_IMAGE	Image tag for Thumbnail of logged-in user
URL_JOIN_COMMUNITY	URL for community login/signup page
URL_LOGIN	URL for community login/signup page
URL_SIGNUP	URL for community login/signup page
URL_LANDING_PAGE	URL for community members page
CONTENT_TITLE	Title of photo in list
USER_ITEM_DETAIL_URL	Thumbnail URL of photo in list

USER_ITEM_IMAGE_URL	Content Detail URL of photo in list
----------------------------	-------------------------------------

Comments Widget

Substitution Name	Description
CONTENT_TYPE	Content Type name specified in the widget parameters ("contentType" param)
USER_PROFILE_URL	Profile URL of logged-in user.
USER_NAME	Display Name of logged-in user. Shows "Guest" for logged-out users
USER_IMAGE	Image tag for Thumbnail of logged-in user
USER_THUMBNAIL_URL	Thumbnail URL of logged-in user.
COMMENT_ID	Flux ID of comment in list
COMMENT_LEVEL	Level of comment in list. This will always be 0 for flat lists, but is greater than 0 for threaded lists
COMMENT_USER_NAME	Display Name of creator of comment in list
COMMENT_USER_URL	Profile URL of creator of comment in list
COMMENT_USER_IMAGE_URL	Thumbnail URL of creator of comment in list
COMMENT_TEXT	Message body of comment in list
COMMENT_DATE	Creation Date of comment in list
COMMENT_ACTIONS	Displays actions available to current user dependent upon context (reply, edit, delete, flag)
COMMENTS_COUNT	Count of number of comments for current content item
COMMENT_REPLY_ITEMS	Renders all threaded replies for current comment in list if Threaded mode is enabled
COMMENT_IMAGE	Image tag for Thumbnail of creator of comment in list. If comment creator has a profile URL, displays a clickable template, otherwise uses not clickable template. Note – all members have a profile URL but some guests may not specify one.
COMMENT_USER_DISPLAY	Display Name of creator of comment in list. Displays clickable template for Flux members, otherwise uses not clickable template

Content Action Widget

Substitution Name	Description
PERMALINK_URL	Permalink of current content item that Content Action widget is bound to

Usage Widget

Substitution Name	Description
LINK	Permalink of current content item specified in the widget parameters ("link" param)

Community Menu Widget

Substitution Name	Description
-------------------	-------------

LINK_URL	URL of menu item
LINK_TEXT	Display Text of menu item
LINK_ATTRIBUTES	Link attributes of menu item
ITEM_INDEX	Index of menu item
SUB_ITEMS	Displays list of sub-items of menu item

Quick Menu Widget

Substitution Name	Description
USER_NAME	Display Name of logged-in user
USER_IMAGE_URL	Thumbnail URL of logged-in user
USER_PROFILE_URL	Profile URL of logged-in user
JOIN_URL	URL for community login/signup page
SIGN_IN_URL	URL for community login/signup page
COMMUNITY_NAME	Title of current community
USER_MAIL_URL	Inbox URL of logged-in user
UNREAD_MAIL_COUNT	Count of unread mail items for logged-in user
VIEW_PENDING_FRIEND_REQUEST_URL	Friend Management URL of logged-in user
PENDING_FRIEND_REQUEST_COUNT	Count of pending friend requests for logged-in user
URL	URL of quick menu item in list
TITLE	Display Text of quick menu item in list
ATTRIBUTES	Link attributes of quick menu item in list