Developing OpenSocial gadgets for IBM Connections 4.0

David McMullin (dmcmulli@ie.ibm.com), IBM
Bernadette Carter (bacarter@us.ibm.com), IBM

September 2012

Summary: OpenSocial gadgets are small components written in XML, HTML, and JavaScript that display custom content. This white paper explains how to develop gadgets for IBM Connections 4.0, primarily focusing on using the developer bootstrap page for quickly testing gadgets.

Table of Contents

1 Overview

The process for adding a gadget to IBM® Connections once it has finished development is reasonably straightforward; however, it can be cumbersome while a gadget is still in development and updates are being made.

For this reason, it can be useful to enable developer mode and skip many of the whitelisting or wsadmin command steps needed when adding a finalized gadget.

This white paper focuses on how to use developer mode with the bootstrap page and other best practices for developing gadgets in IBM Connections 4.0. We address the three distinct ways to render OpenSocial gadgets in IBM Connections but if, for example, you want only to develop a Share dialog gadget, you should read the Guidelines for all Gadgets and then the Share dialog section.

2 Guidelines for all gadgets

Let's begin by discussing the guidelines for all gadgets.

2.1 Enabling developer mode

The server Administrator can enable a development page and a number of hosts who have the authority to deliver "developer gadgets". To do this:

1. Edit the opensocial-config.xml file in the LotusConnections-config directory (on a production server you should use the admin commands to check out the file first; refer to the Connections 4.0 product documentation topic, Editing configuration files, for more details).
2. Set the enabled attribute of developer to "true" and include the base URL(s) of any servers from which you plan to publish developer gadgets (see listing 1). You can then use the developer bootstrap page to test gadgets.
3. If you want developer-mode gadgets to have access to these features, set allowSSOFeature and allowIntranetProxyAccess to true (these features are discussed below in more detail in Section 2.4, “Gadget options”).
4. If you are not using a production server, it may be convenient to set allServers to true, allowing any server to publish developer-mode gadgets; however, note that this is not secure.

Listing 1. opensocial-config.xml

```
<gadget-settings>
        . . .
        <developer enabled="true" allowSSOFeature="true" 
allowIntranetProxyAccess="true">
            <developer-hosts-whitelist allServers="false">
                <!--
                List of base URLs that are allowed to publish 'developer-
mode' gadgets
                <host url="http://{host.com}/base/url/1" />
                ...
                <host url="http://{another.host.com}/base/url/N" />
                 -->
            </developer-hosts-whitelist>
        </developer>
        . . .
</gadget-settings>
```

With developer mode enabled, gadget URLs from the specified hosts will be granted the permissions specified as a query string appended to the URL, for example, http://{host.com}/base/url/1/gadget.xml?__dev_proxyPolicy__=external_only.

It is not necessary to construct these URLs manually; the bootstrap page allows you to enter the gadget URL, select access options (proxy policy, feature access, and render mode) and copy the formatted URL. The formatted URL can then be used anywhere in IBM Connections (including an Activity entry) to render the gadget with those access options.

2.2 Using the bootstrap page

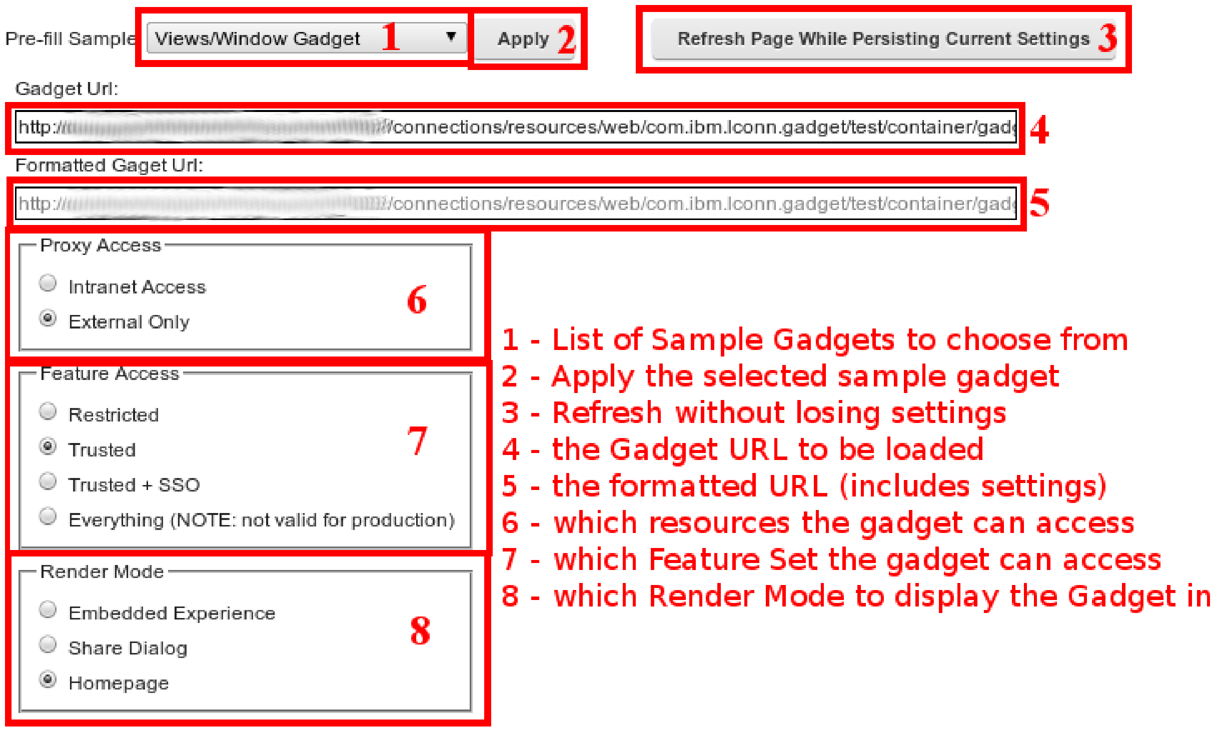
The bootstrap page is intended for testing gadgets quickly, without excessive setup or administration, allowing more rapid gadget development (see figure 1). It is located at {server_root}/connections/resources/web/com.ibm.lconn.gadget/test/bootstrap.html.

2.3 Using the examples

The simplest way to learn about the page is to try a few of the sample gadgets it provides. To render a sample gadget, select it from the drop-down list (1) and click Apply (2). This populates the gadget URL and options but does not render the gadget.

To render, click the appropriate button on the right-hand side of the page, "[Re]Load Gadget" for Home page and Embedded Experience gadgets, "[Pre]Load Gadget" and then "Open Global Sharebox" for Share dialog gadgets.

Figure 1. Options for rendering gadgets on the bootstrap page



Gadget Url: http://[redacted]/connections/resources/web/com.ibm.lconn.gadget/test/container/gad

Formatted Gaget Url: http://[redacted]/connections/resources/web/com.ibm.lconn.gadget/test/container/gad

Proxy Access: Intranet Access [✓ External Only]

Feature Access: Restricted [✓ Trusted] Trusted + SSO Everything (NOTE: not valid for production)

Render Mode: Embedded Experience Share Dialog [✓ Homepage]

1 - List of Sample Gadgets to choose from
2 - Apply the selected sample gadget
3 - Refresh without losing settings
4 - the Gadget URL to be loaded
5 - the formatted URL (includes settings)
6 - which resources the gadget can access
7 - which Feature Set the gadget can access
8 - which Render Mode to display the Gadget in

2.4 Gadget options

A list of gadget options is available on the left-hand of figure 1. Note the two URL input boxes: the **Gadget URL (4),** which should be entered by the user; and the **Formatted Gadget URL (5)**, which is produced by the page and encodes the options selected below. This URL can be used to register the gadget in the Home page Administration page. You must be a Home page administrator to do this, and the formatted URL should only be used during development, after which the corresponding options in the admin UI should be used.

Proxy Access (6). This setting dictates whether the gadget will have access to resources within the server's intranet.

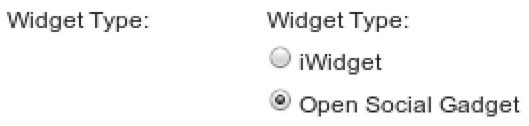
Feature Access (7). This setting dictates which set of features the gadget will have permission to use. For more details, see the Connections product documentation topic, "Developing OpenSocial gadgets".

Render Modes (8). This setting defines how the gadget will render; each mode is discussed in a separate section of this paper.

3 Example of deploying the sample User Preferences gadget

Let's try adding one of the sample gadgets to connections:

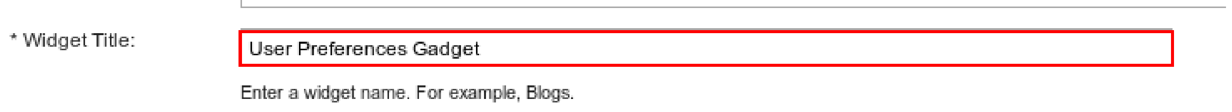
1. Select "UserPrefs Gadget" from the Examples drop-down list and click Apply.
2. If you haven't already, load the gadget, and try changing the userPrefs and then reLoading.
3. Optionally, host your own gadget and render with the same settings; all you need to do is change the gadget URL.
4. Go to the Home Page in Connections (you must be an administrator) and click Administration.
5. Click "Add another widget" and select "Open Social Gadget":



Widget Type: Widget Type:
iWidget [✓ Open Social Gadget]

6. You can leave most of the settings as default for now, except these three:

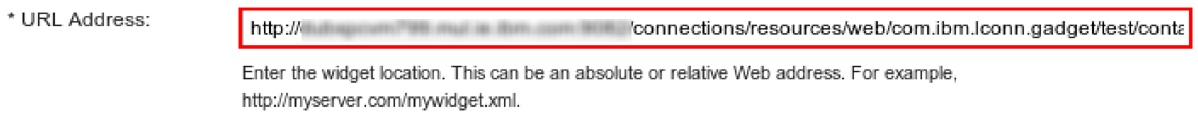
   a) Enter a gadget title:



* Widget Title: User Preferences Gadget

Enter a widget name. For example, Blogs.

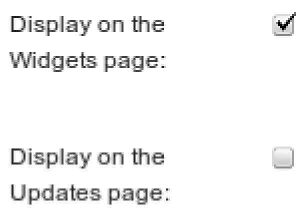
   b) Enter your gadget URL:



* URL Address: http://[redacted]/connections/resources/web/com.ibm.lconn.gadget/test/conta

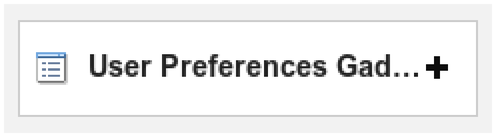
Enter the widget location. This can be an absolute or relative Web address. For example, http://myserver.com/mywidget.xml.

c) Select a render point; we chose the Widgets page ("My Page" in the menu on the left-hand side):



Display on the Widgets page:	✓
Display on the Updates page:	☐

7. Click Save, select your newly added gadget, and click Enable.
8. Navigate to My Page and add your new gadget via the Customize menu:



9. Optionally, try changing user preferences using the Edit menu (see figure 2); notice that changes are now persisted (see figure 3).

Figure 2. Edit menu

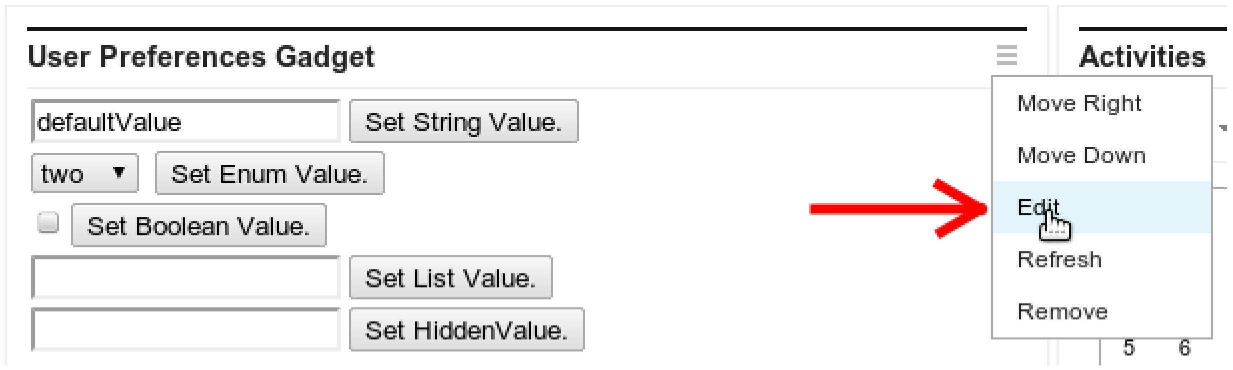
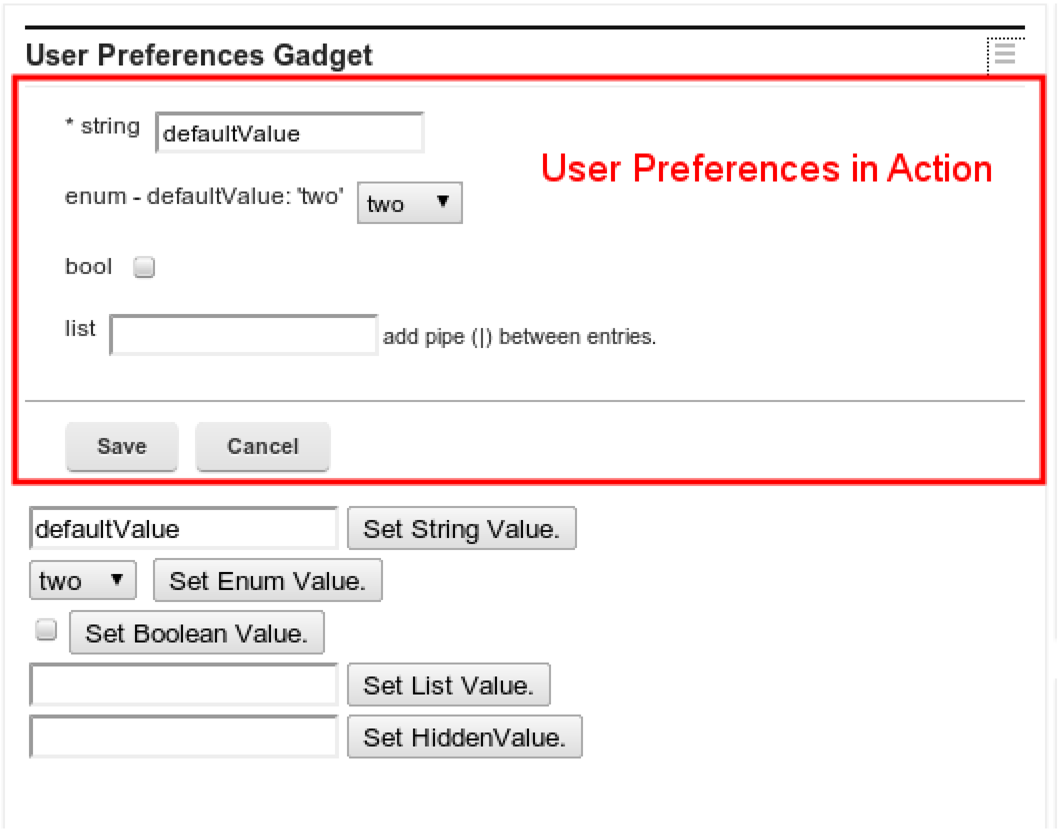


Figure 3. Using the sample Connections UserPrefs gadget



* string: defaultValue
enum - defaultValue: 'two': two
bool
list: add pipe (|) between entries.

defaultValue
two

4 General best practices for developing gadgets

Use `gadgets.util.registerOnLoadHandler`

If you are executing JavaScript™ inside your gadget and you rely on gadget features being loaded, always wrap your code in `gadgets.util.registerOnLoadHandler:`

```
gadgets.util.registerOnLoadHandler(
  function() {
    // do some work with gadget features.
});
```

Note, however, that there is no gadget "onLoad" event; this function simply implements a function queue, but you can be guaranteed that by the time your code reaches the top of the queue, all gadget features are loaded.

Be careful when resizing

By requiring the "dynamic-height" and "dynamic-width" OpenSocial features, a gadget is provided access to gadgets.window.adjustHeight and gadgets.window.adjustWidth. Whenever content is added to or removed from the gadget, the gadget should make an adjustHeight call (with no parameters) so that the height of the gadget's iFrame will shrink or grow as needed, based on the content displayed.

It's important to note, however, that there are limits to the size of your gadget in many contexts. One way to check on the success of your resize attempt is to query gadgets.window.getViewPortDimensions.

Use content-rewrite exclusion to improve performance

Use the content-rewrite exclude feature and serve cascading style sheets (CSS) and JavaScript directly from your server.

Be careful with `gadgets.views.openGadget`, `openEmbeddedExperience` and `openUrl`

IBM Connections supports only the modal dialog viewTarget. Attempts to render your gadget (or an Embedded Experience or URL) in another viewTarget such as tab or float will simply render in the modal dialog instead.

Use data Pipelining where possible

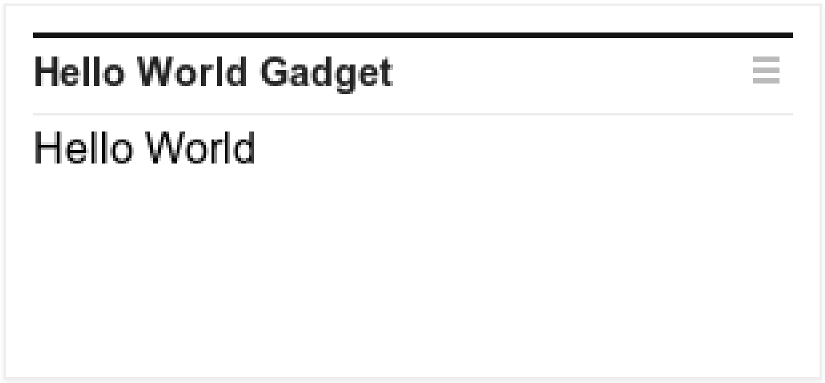
The data pipelining feature is great way to improve performance as it reduces the number of requests needed after your gadget has loaded. With pipelining, you can instruct the server to load resources (from http endpoints or gadget data APIs) with your gadget, as part of the same initial response. The "People/Pipeline Gadget" sample gadget from the bootstrap page is an excellent place to learn more about pipelining.

5 Developing Home page gadgets

What are Home page gadgets?

This render mode does not involve any major changes or features for the developer to be aware of. Gadgets rendered this way can be seen on My Page and in the sidebar on the right-hand side of Activity Stream pages (Discover, Saved, Action Required, My Notifications, Status Updates, I'm Following). Figure 4 shows a sample Home page gadget.

Figure 4. Sample Home page gadget

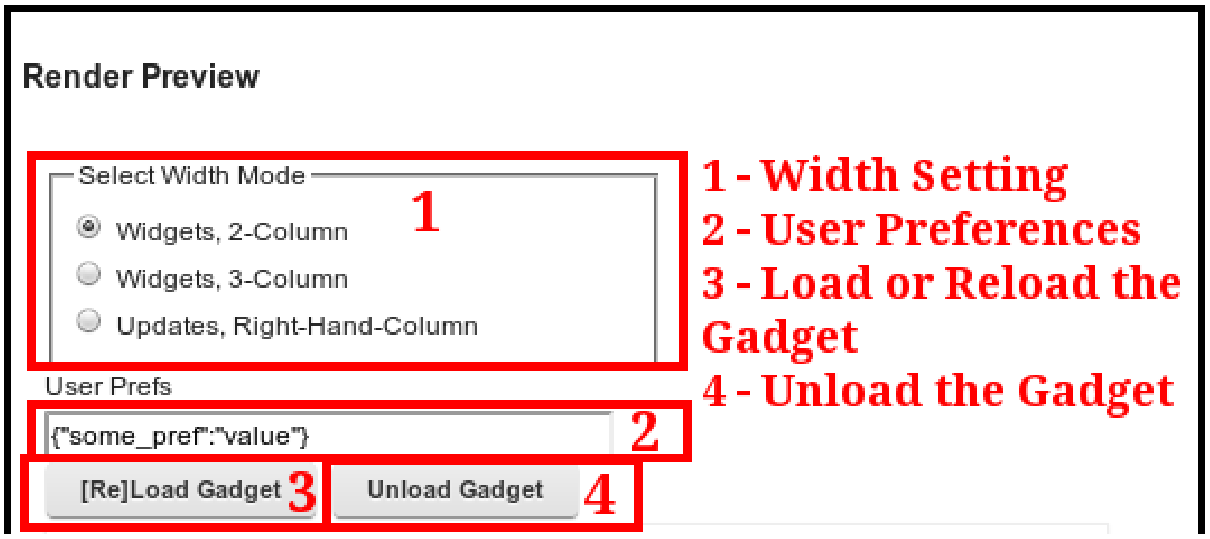

Hello World Gadget

Hello World

Bootstrapping Home page gadgets

Figure 5 shows the options for rendering Home page gadgets on the bootstrap page.

Figure 5. Options for rendering Home page gadgets



It is possible to select any one of three widths (1), corresponding to different render contexts. User Preferences (2) can be passed to the gadget in JavaScript Object Notation (JSON) format. The bootstrap page does not provide any User Preference persistence, and trying to set them from the gadget will fail; however, the feature is fully functional in IBM Connections.

Note that the behavior of the title bar on the bootstrap page is different than the behavior in the Home page. On the bootstrap page, "[WIDGET TITLE]" is always displayed, whereas in the Home page, the initial value is the title string configured in the Admin UI (not the title in the gadget's ModulePrefs section), and calls to `gadgets.window.setTitle` will successfully set the title.

Moving Home page gadgets from bootstrap page to IBM Connections

The example given above in Section 2, “Guidelines for all gadgets,” demonstrates the process for adding a Home page gadget to IBM Connections.

5.1 Best practices for Home page gadgets

Adding a Help Link

Home page gadgets support surfacing a Help menu action for users. To specify the Help link, the gadget adds a `<Link ...>` tag to the ModulePrefs section of the gadget XML definition, as shown in listing 2.

Listing 2. Adding a Help link to a gadget's ModulePrefs section

```
<Module>
  <ModulePrefs title="my title" description=my description">
    ...
    <Link rel="ibm.connections.help" href="endpoint://help/..."/>
    ...
  </ModulePrefs>
</Module>
```

When Home page detects the `<Link ...>` tag, a Help action is added to the action menu for the gadget. When a user selects the Help menu, the link opens in a new window, and the endpoint://help/ portion of the href is replaced by the Help context from the Connections config file.

User Preferences

When gadgets are shown on the Connections Home page, `<UserPref ...>` attributes are introspected and, if present, an Edit action is added to the gadget's context menu. Clicking the Edit choice displays a UI for editing the gadget's user preferences. Clicking the Save button persists the changes back to the server for the gadget.

In addition, `gadgets.Prefs.set()` is supported. Note that setting a Preference via the API causes a request to the server to persist the value. This is done for every API call.

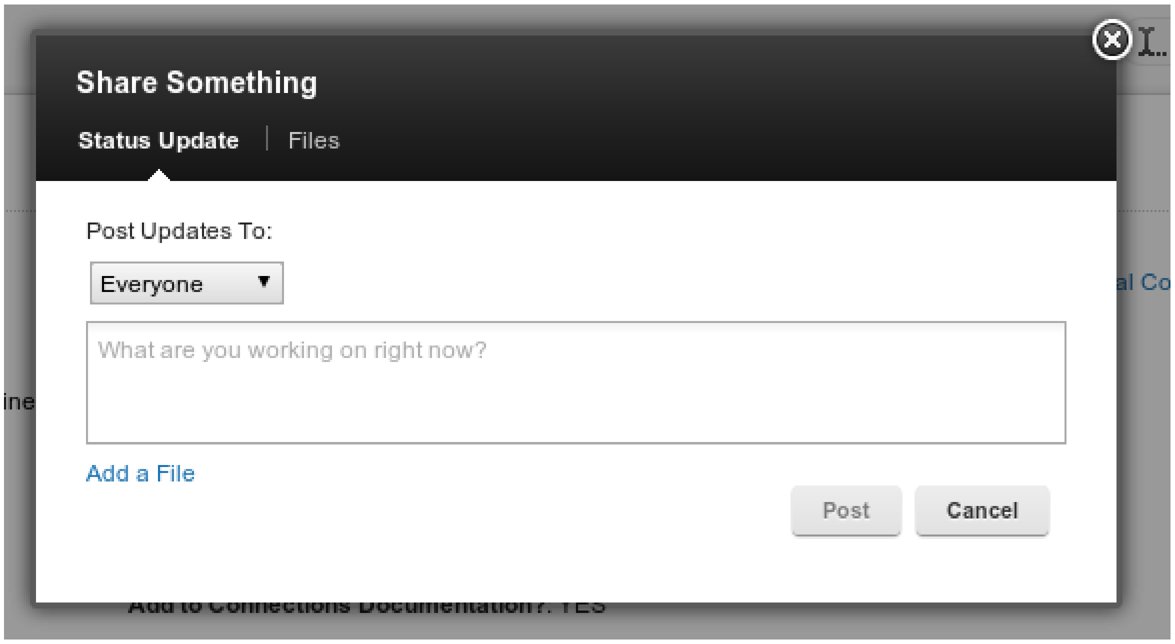
6 Developing Share dialog gadgets

What are Share dialog gadgets?

Gadgets rendered in the Share dialog should provide a way for users to share information or content (see figure 6). The gadget must contribute a tab by defining an action that has the path "container/sharebox" in order to be visible in the Share Something dialog. (See the product documentation topic, “Adding new ways to share content”, for additional details.)

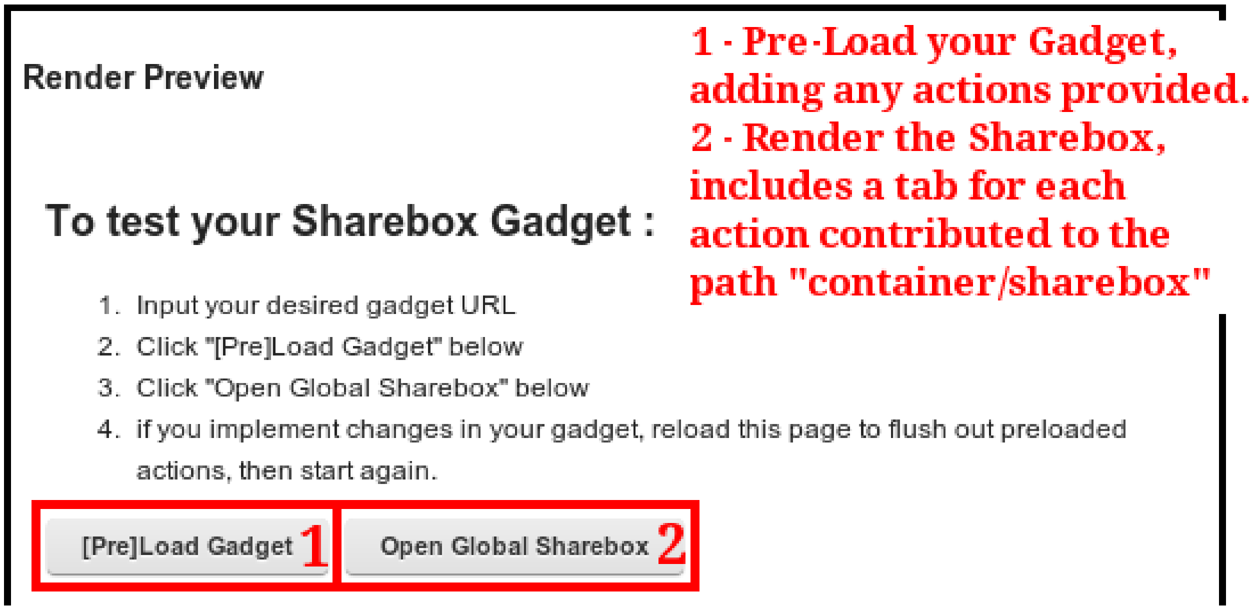
The gadget must also include the "ibm.connections.sharedialog" feature in order to interact with the Share dialog container. It is safest to include this feature by using an `<Optional>` tag, as opposed to a `<Require>` tag.

Figure 6. Share dialog with Status Update and Files gadgets



Bootstrapping Share dialog gadgets
Figure 7 shows the options of rendering Share dialog gadgets, which are referred to as Sharebox gadgets on the bootstrap page.

Figure 7. Options for rendering Share dialog or Sharebox gadgets



Render Preview

To test your Sharebox Gadget :

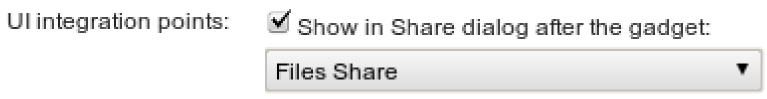
1. Input your desired gadget URL
2. Click "[Pre]Load Gadget" below
3. Click "Open Global Sharebox" below
4. if you implement changes in your gadget, reload this page to flush out preloaded actions, then start again.

1 - Pre-Load your Gadget, adding any actions provided.
2 - Render the Sharebox, includes a tab for each action contributed to the path "container/sharebox"

Refer to the sample Sharebox gadget on the bootstrap page to get started. It may be helpful when developing a Share dialog or Sharebox gadget to use the "Refresh Page While Persisting Current Settings" button, to allow you to Re- or Pre-Load your Share dialog gadget quickly, without having to set up the options on the left-hand side again.

Moving Share dialog gadgets from bootstrap page to IBM Connections
The process for moving a Share dialog gadget is quite similar to that for a Home page gadget, which can be seen in the example from Section 2 above (see figure 8). For the Share dialog, you must choose where in the tab order your gadget will be displayed. Note that you may not place your gadget before the Share dialog gadgets provided by IBM Connections.

Figure 8. Adding a Share dialog gadget to Connections



UI integration points: ✓ Show in Share dialog after the gadget: Files Share

6.1 Best practices for Share dialog gadgets

Gadget resizing (dynamic-height and dynamic-width)
Functions to use:

- `gadgets.window.adjustHeight`
- `gadgets.window.adjustWidth`

The Share dialog allows for containing gadgets to have a *width* of up to 500 px before horizontal scroll bars will appear. However, the gadget is allowed to take up as much or as little *height* as needed. As the content rendered in the gadget changes, the gadget must inform the container of the size change so that the Share dialog will resize to provide the height needed. The use of these functions is covered above in Section 4, "General best practices for developing gadgets."

Adding tabs to the Share dialog (action-contributions)

Function to use:

- `gadgets.actions.updateAction`

You may add tabs to the dialog by defining actions in the gadget definition (see the product documentation topic, "Adding new ways to share content", for additional details.) Each action must have an ID that is unique from all other actions loaded in the Share dialog. Your gadget can define the function to be called when a user clicks on your gadget's tab by calling gadgets.actions.updateAction and passing in the action.

The action must consist of an "id" and a "callback". The callback is called each time a user clicks on the tab that was created in the Share dialog for the action; therefore, the gadget should perform rendering tasks inside the callback function.

However, since the callback is invoked each time the tab is selected, your gadget will want to keep track of if it has been initialized previously. If the gadget has already been initialized, it should perform only updates to the UI as needed.

The gadget should associate a callback function with each of the gadget actions after the gadget has been loaded, using logic similar to that shown in listing 3.

Listing 3. Registering a Share dialog action

```
gadgets.util.registerOnLoadHandler(function() {
  if (gadgets.actions) {
    var customAction = {
      id: "customAction",
/* Note: The id matches the id of the action from the ModulePrefs section -
Example:
<action id="customAction" path="container/sharebox" label="Custom Action" />
*/
      callback: updateContext
/* Note: The callback function will get called each time the action is invoked
      (Share dialog Example: Every time the tab is selected.*/
    };
    gadgets.actions.updateAction(customAction);
  }
});
```

Gadget initialization

Function to use:

- `ibm.connections.sharedialog.registerCloseListener`

The `ibm.connections.sharedialog.registerCloseListener` method provides a way for the gadget to know that the Share dialog has closed but has not been destroyed. When the dialog has been closed, the function that has been passed into `ibm.connections.sharedialog.registerCloseListener` will be called, and the function passed into `registerCloseListener` should revert the gadget to the default state.

After the Share dialog has been closed, the contents of the gadget will remain until a page change or page refresh has occurred. Therefore, it is imperative for the gadget to return itself to the default state when it has been informed that the Share dialog has been closed, so it's in the default state if the user re-opens the Share dialog without performing a page change or page refresh.

The function passed into `registerCloseListener` should not invoke `ibm.connections.sharedialog.close` because the Share dialog is closed but not destroyed when the function registered has been called:

```
ibm.connections.sharedialog.registerCloseListener(function() {
  //Reset gadget contents to default state
});
```

User has entered content to share

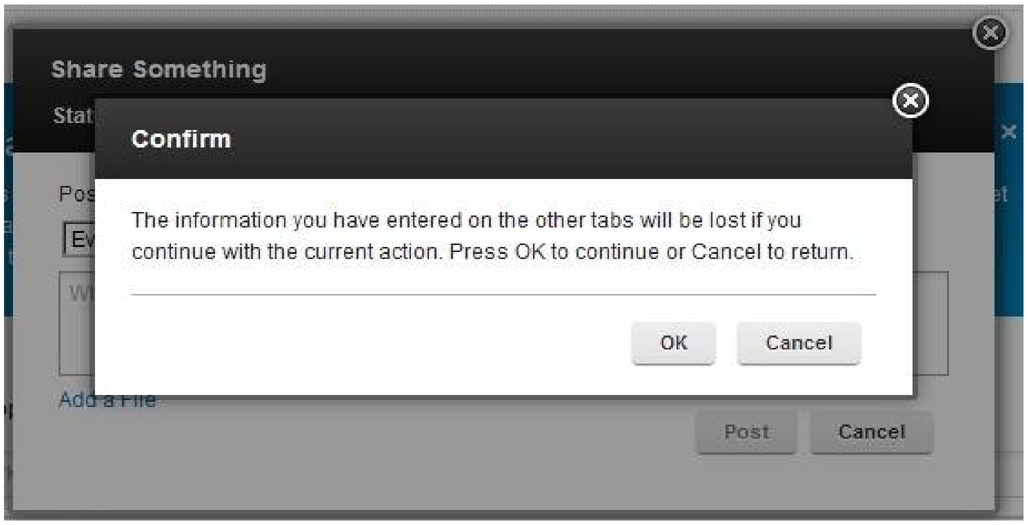
Function to use:

- `ibm.connections.sharedialog.setDirty`

The gadget should call `ibm.connections.sharedialog.setDirty(true)` when one of the fields has been changed from the default state. On the other hand, if all fields have been returned to the default state, then the gadget should call `ibm.connections.sharedialog.setDirty(false)` to make the Share dialog aware that there are not any fields with user input changes.

If “true” was passed into setDirty the last time it was called and a user attempts to close the Share dialog on a different tab from your gadget---by using the "X" on the Share dialog or by pressing the Escape key---then a confirm prompt will be displayed so that the user must confirm the desire to close the Share dialog (see figure 9).

Figure 9. Confirmation prompt when exiting the Share dialog with unsaved changes



Share Something

Confirm

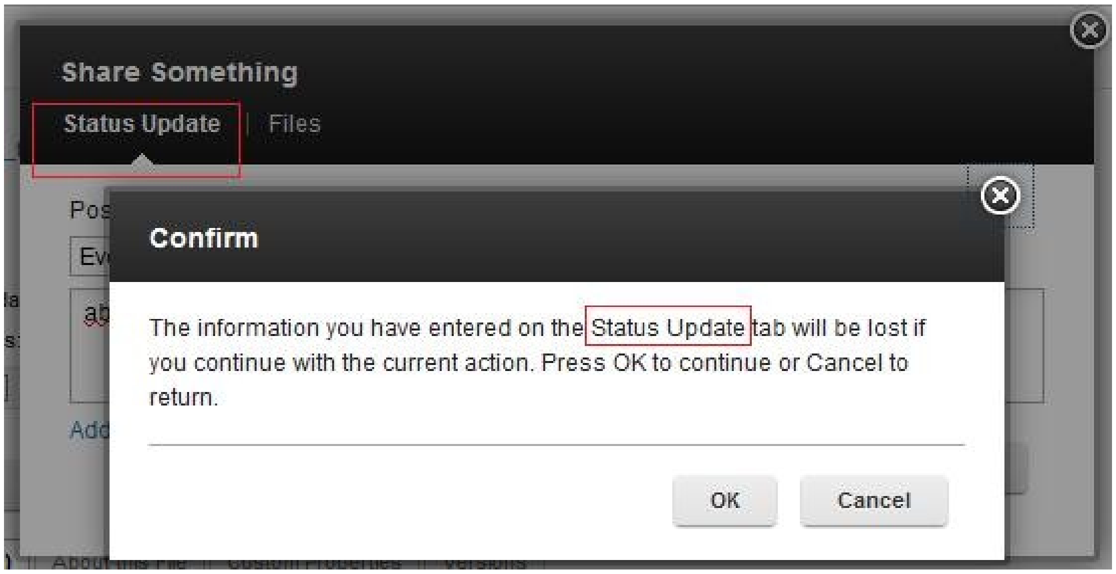
The information you have entered on the other tabs will be lost if you continue with the current action. Press OK to continue or Cancel to return.

For consistency with other gadgets in the Share dialog, gadgets should have a Cancel button that resets the gadget to the default state and then makes a call to close the dialog.

If you click Cancel in your gadget and none of the other gadgets in the Share dialog have unsaved changes, then when your gadget invokes `ibm.connections.sharedialog.close`, the Share dialog should close without a confirm prompt.

On the other hand, if the user is on the tab of your gadget, the Share dialog knows that your gadget has changes, and the user attempts to close the dialog by clicking the "X" or pressing the Escape key, then the message will reference the name of your gadget. This is highlighted by the red box around the gadget action name in figure 10.

Figure 10. Confirmation prompt when exiting the Share dialog with unsaved changes



The information you have entered on the Status Update tab will be lost if you continue with the current action. Press OK to continue or Cancel to return.

Sharing In Progress

Function to use:

- `ibm.connections.sharedialog.setSubmitState`

The call to `ibm.connections.sharedialog.setSubmitState(true)` should only be done if the gadget needs to prevent the user from switching to a different tab in the Share dialog. If this function is used, it should only be called immediately before doing a content share submission.

When the submission is over, `ibm.connections.sharedialog.setSubmitState(false)` must be called, to allow the user to be able to change tabs in the Share dialog. Therefore, if this function is used, then `ibm.connections.sharedialog.setSubmitState(false)` should be called before closing the dialog.

Sharing Success

Functions to use:

- `ibm.connections.sharedialog.onActivityEntryAddition`
- `ibm.connections.sharedialog.displayMainPageMessage`
- `ibm.connections.sharedialog.close`

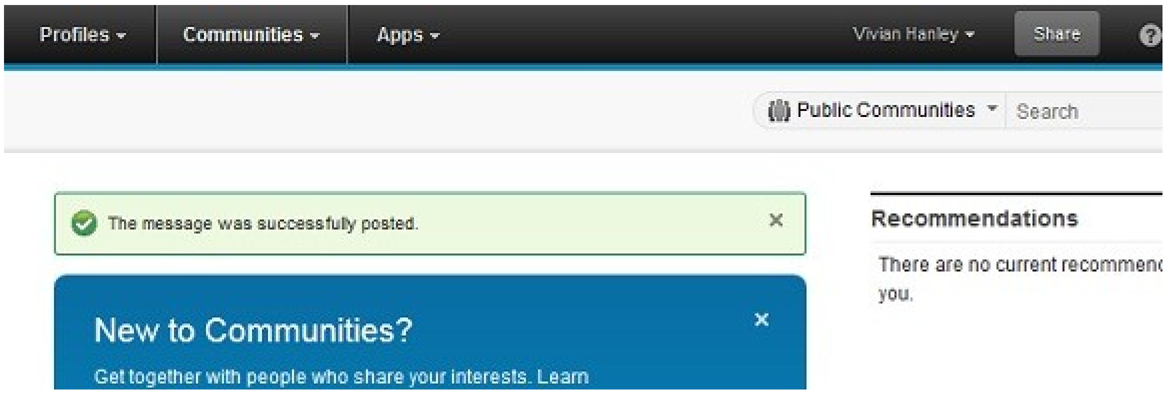
If the sharing action from your gadget will result in an Activity Stream entry in the Connections UI, it is recommended to call `ibm.connections.sharedialog.onActivityEntryAddition()` and pass in the "id" of the item that was shared.

This alerts the Connections Activity Stream that a new event has occurred so that it can refresh the stream to display the new event if it is available. If the result of sharing content in your gadget does not generate an Activity Stream event, then the gadget should not use this method.

It is recommended to display a Success message on the Connections UI and close the Share dialog after a successful content share (see figure 11). The gadget can display a success message by making the following call:

```
ibm.connections.sharedialog.displayMainPageMessage(
                ibm.connections.sharedialog.MessageTypeParam.SUCCESS,
                "The message was successfully posted."
);
```

Figure 11. Success message after closing Share dialog



The message was successfully posted.

There are no current recommend you.

After calling the method to display the Success message, the gadget should reset itself to the default state and then call `ibm.connections.sharedialog.close()` to close the Share dialog.

NOTE: Your gadget should not call any other methods on the feature or attempt to perform any additional clean-up or actions after calling `ibm.connections.sharedialog.close()`.

Sharing Error

Function to use:

- `ibm.connections.sharedialog.displayMainPageMessage`

It is recommended to keep the Share dialog open and display a message in your gadget if an error occurred while sharing. However, if you want the Share dialog to close after a sharing error, it is recommended to notify the user that the share action was not successful by calling the following to display an error message in the Connections UI:

```
    ibm.connections.sharedialog.displayMainPageMessage(
                      ibm.connections.sharedialog.MessageTypeParam.ERROR,
                      "Unable to share content."
    );
```

After displaying the message, then you can call `ibm.connections.sharedialog.close()` to close the Share dialog if desired.

NOTE: You should not call any other methods on the feature or need to perform any additional clean-up or actions after calling `ibm.connections.sharedialog.close()`.

Cancel button or Close Share dialog

Function to use:

- `ibm.connections.sharedialog.close`

Before calling `ibm.connections.sharedialog.close` to close the Share dialog, your gadget should reset itself to the default state. If a user performs a page refresh or changes pages, then the gadget will be reloaded.

However, if the user does not refresh the page or change pages, then subsequent open actions on the Share dialog will result in the gadget displaying the state in which it was closed, which is an inconsistent experience with the gadgets provided by IBM Connections.

NOTE: You should not call any other methods on the feature or need to perform any additional clean-up or actions after calling `ibm.connections.sharedialog.close()`.

Adding a Loading Indicator

Inside of your gadget div, create a div for the "waiting" experience while a separate div for the "content" is being populated. The Share dialog passes in as a gadget user preference a loading gif url that is used by out of the box IBM Connections Share dialog OpenSocial gadgets (see listing 4).

Listing 4. Example CSS that uses the loadingGifUrl

```
<style type="text/css">
         .customGadget div.loading {
            background-image:url(__UP_loadingGifUrl__);
            background-repeat: no-repeat;
            min-height: 60px;
            width: 100%;
            background-position: center center;
         }
</style>
```

You can have access to this user preference by including the following in your gadget definition:

```
    <UserPref name="loadingGifUrl" required="false" type="hidden"
               default_value="" datatype="string"/>
```

Here's an example DOM node setup that includes a gadgetWaiting div that is displayed immediately and a gadgetBody div that the gadget should populate during initialization:

```
       <div id="gadgetDiv" class="customGadget">
            <div id="gadgetWaiting" class="loading"></div>
            <div id="gadgetBody" style="display:none"></div>
        </div>
```

Hide the gadgetWaiting div and display the gadgetBody content div at the end of the initialization process as shown in listing 5. The initialization process should begin when the gadget action's callback function has been invoked and the selection type matches `com.ibm.social.sharebox.context`.

Listing 5. Hiding the loading image and displaying gadget body

```
<script type="text/javascript">

gadgets.util.registerOnLoadHandler(function() {
  if (gadgets.actions) {
    var customAction = { id: "customAction", callback: updateContext};
    gadgets.actions.updateAction(customAction);
  }
});
function initCustomContent(context) {
  var waitingDiv = document.getElementById("gadgetWaiting"),
  gadgetBodyDiv = document.getElementById("gadgetBody");

  //Populate gadget content div
  gadgetBodyDiv.innerHTML="Content for display in Share dialog for custom
gadget";

  //Hide waiting div, display content
  waitingDiv.style.display = "none";
  gadgetBodyDiv.style.display = "";
}

function updateContext(selection) {
  if(selection.type == "com.ibm.social.sharebox.context")
    initCustomContent(selection.dataObject);
}

</script>
```

Action callback caveat

Remember that updateContext is called each time a user visits the gadget's tab or re-opens the Share dialog, when it was previously on your gadget and the page has not been changed or refreshed.

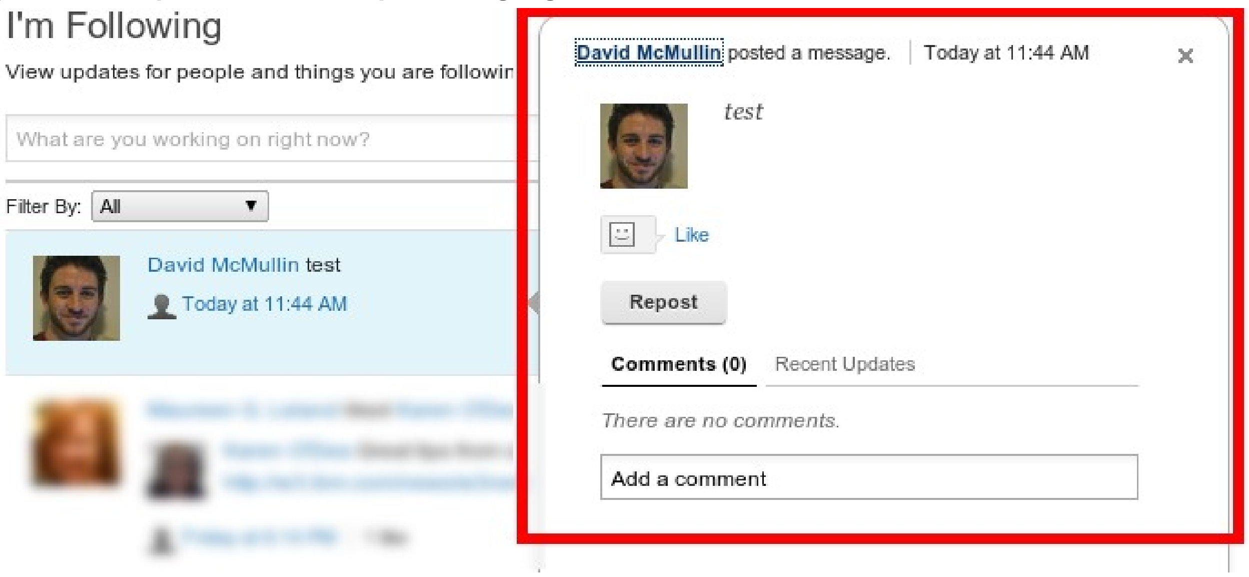
Therefore, you will likely only want to perform the initialization the first time your action callback function is called; each additional time you will likely want to only do an update as needed.

7 Developing Embedded Experience gadgets

What are Embedded Experience gadgets?

This render mode is intended for a deeper interaction with an activity entry, and you trigger it by clicking an entry in the user's Activity Stream. A common example is allowing the user to comment or repost the post of another user, without navigating away from the Activity Stream. Figure 12 shows an example of this gadget.

Figure 12. Sample Embedded Experience gadget



Bootstrapping Embedded Experience gadgets

Figure 13 shows the options for rendering Embedded Experience gadgets on the bootstrap page; figure 14 shows the options for POSTing Embedded Experience gadgets on the bootstrap page.

Figure 13. Options for rendering Embedded Experience gadgets

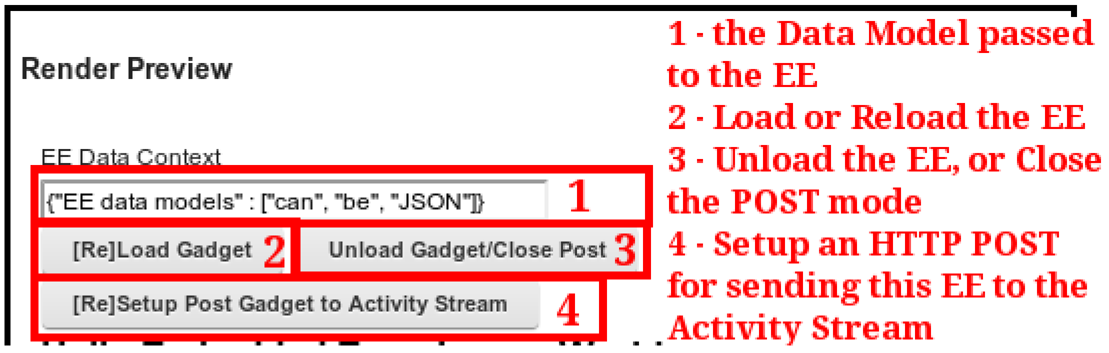
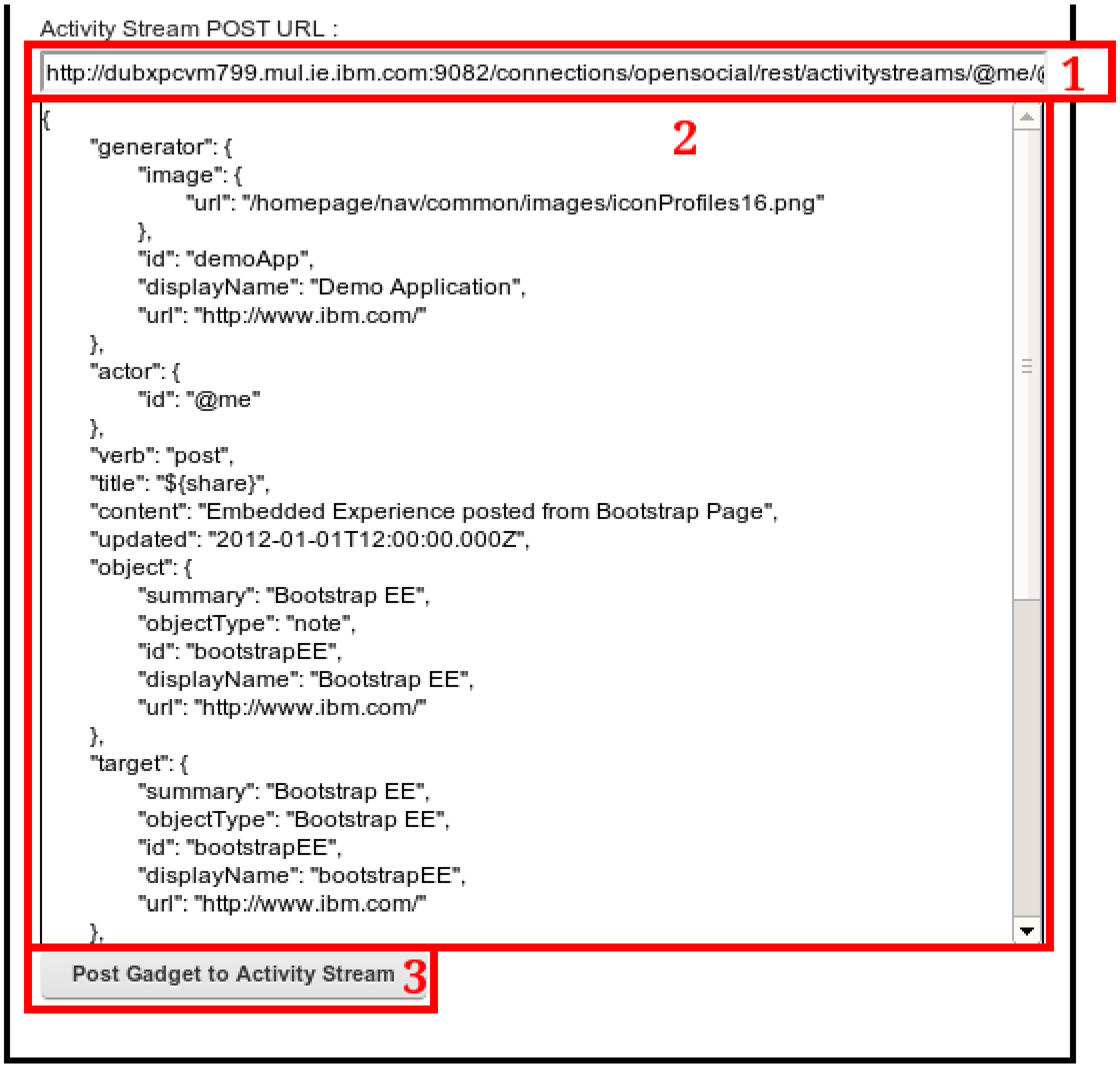

Render Preview
EE Data Context
{"EE data models" : ["can", "be", "JSON"]}
[Re]Load Gadget
Unload Gadget/Close Post
[Re]Setup Post Gadget to Activity Stream
1
2
3
4
1 - the Data Model passed to the EE
2 - Load or Reload the EE
3 - Unload the EE, or Close the POST mode
4 - Setup an HTTP POST for sending this EE to the Activity Stream

Figure 14. Options for POSTing Embedded Experience gadgets



```
http://dubxpcvm799.mul.ie.ibm.com:9082/connections/opensocial/rest/activitystreams/@me/(
```

```
{
        "generator": {
                "image": {
                        "url": "/homepage/nav/common/images/iconProfiles16.png"
                },
                "id": "demoApp",
                "displayName": "Demo Application",
                "url": "http://www.ibm.com/"
        },
        "actor": {
                "id": "@me"
        },
        "verb": "post",
        "title": "${share}",
        "content": "Embedded Experience posted from Bootstrap Page",
        "updated": "2012-01-01T12:00:00.000Z",
        "object": {
                "summary": "Bootstrap EE",
                "objectType": "note",
                "id": "bootstrapEE",
                "displayName": "Bootstrap EE",
                "url": "http://www.ibm.com/"
        },
        "target": {
                "summary": "Bootstrap EE",
                "objectType": "Bootstrap EE",
                "id": "bootstrapEE",
                "displayName": "bootstrapEE",
                "url": "http://www.ibm.com/"
        },
```

1 - the URL being posted to
2 - the JSON data being sent (editable)
3 - Perform HTTP POST

The bootstrap page allows you to see your gadget render (without the Embedded Experience outline) with a dataModel you provide, and to post this Embedded Experience to your Activity Stream (you must be logged in). A new Activity Entry then appears on your "I'm Following" page (not on the other pages) and, when it is clicked, your gadget will be rendered.

It is possible to edit the JSON post body before sending it to the Activity Stream. Note that, to produce a new Activity Entry, you must change the connections.rollupid value; otherwise, the previous entry will simply be replaced.

Moving Embedded Experience gadgets from bootstrap page to IBM Connections

The process for moving an Embedded Experience gadget is similar to that of a Home page gadget. For an Embedded Experience, you simply select the "Show for Activity stream events" checkbox instead of specifying any of the other render points.

7.1 Best practices for Embedded Experience gadgets

Accessing the dataModel

Functions to use:

- `opensocial.data.getDataContext().registerListener`
- `opensocial.data.getDataContext().getDataSet`
- `opensocial.data.getDataContext().getData`

There are many ways to access the dataModel passed to your Embedded Experience gadget; for example, you can register a listener or access the content directly, as shown in listing 6, and you must use the "org.opensocial.ee.context" key, as defined in the OpenSocial specification.

Listing 6. Accessing dataModel

```
// make sure we can access opensocial.data
gadgets.util.registerOnLoadHandler(function() {
  var dataModelKey = "org.opensocial.ee.context",
    dataModel1 = opensocial.data.getDataContext().getData()[dataModelKey],
    dataModel2 = opensocial.data.getDataContext().getDataSet(dataModelKey);

  console.debug("printing the dataModel from getData", dataModel1);
  console.debug("printing the dataModel from getDataSet", dataModel2);

  opensocial.data.getDataContext().registerListener(dataModelKey,
function(key) {
      var context = opensocial.data.getDataContext().getDataSet(key);

      console.debug("printing the dataModel from registerListener", context);
  });
});
```

8 Conclusion

The OpenSocial container provided in IBM Connections is extremely useful for displaying custom content at various integration points, but it's essential that you have a good development environment and knowledge of the OpenSocial APIs.

To this end, it is the authors' recommendation that any aspiring OpenSocial developer takes time to use and examine the sample gadgets provided on the bootstrap page, which uses a large range of OpenSocial features, before diving into their own gadgets. Once the features are understood, the bootstrap page remains useful as an area for quickly testing your work.

9 Resources

- developerWorks® IBM Connections product page.

- OpenSocial Home, includes further resources for gadget developers.

- OpenSocial Specifications, provides a list of OpenSocial Specifications (IBM Connections supports the 2.0 spec).

IBM Connections 4.0 product documentation topics:

- Adding custom widgets to the Home page
- Registering third-party applications

10 Author biographies

David McMullin is a Software Engineer at IBM's Dublin Software Laboratory, working as part of the IBM Connections development team since 2011. You can reach David at dmcmulli@ie.ibm.com.

Bernadette Carter is a Software Engineer at IBM's Research Triangle Park Software Laboratory. She currently contributes to the IBM Connections product primarily in the areas of Embedded Experience and Share dialog OpenSocial gadgets, as well as the Media Gallery. You can reach Bernadette at bacarter@us.ibm.com.

Acknowledgements

The authors would like to thank Michael Ahern for suggesting the paper be written; and Ryan Baxter, Bruce Roberts, Cesar Wong, and Vincent Burckhardt for their reviews, suggestions, and encouragement.

Trademarks