Extending Gadgets with APIs

2016 OmniUpdate User Training Conference Activity Guide

OmniUpdate, Inc.
1320 Flynn Road, Suite 100
Camarillo, CA 93012

Contents

Extending Gadgets with APIs

This “Undo Page Parameters Gadget” will go through a couple of the OU Campus Endpoints and how to interact with them via HTTP requests. We will talk about the events that the OU Campus app itself sends to your gadgets, and about something new and really groundbreaking for the gadget world, the Metadata API.

Gadget Setup / Installation

1. Download the undo-page-parameters starter from http://github.com/undo-page-parameters.
2. Create a new folder in OU Campus named “Undo-Parameters-Gadget.”
3. Upload the zip/files into this folder.
4. Publish the “Undo-Parameters-Gadget” folder so that the files are uploaded to your production server.

After you have successfully uploaded the gadget starter files and published them to your production server, we will need to add this as a gadget in OU Campus. (Refer to the **Introduction to Gadgets Guide** to see how to install your gadget into OU Campus)

Plan of Attack / Index.html

Before we jump into coding, let's take a step back and think about how this gadget is actually going to work, and make a plan of what information we will need to retrieve, and how / when that is going to happen. Here is the basic order of operations:

1. Retrieve the parameters of the current file.
2. Wait until the user saves the page parameters.*
3. Save the previous page parameters.
4. Utilize the data to revert the page parameters back to a previous save.

*We want to wait until the user has clicked save, and then save our page parameters, so that each time the gadget is loaded a version of the page parameters is not unnecessarily saved.

If we take a look at the index.html, we can see that we are registering a ‘params-save’ event on the gadget object. This will allow the gadget to execute an action when the user saves Page Parameters. The html for this simple gadget is in place along with some starting CSS.

Start Coding

First we need to get information about the current file we are on. Create an **init()** function that will be used to jumpstart our gadget. We will call this function in the document `ready callback function`, located in the index.html of the gadget starter project.

Inside the init function our first step is to discover which file we are currently viewing. The gadget lib has a method **oucGetCurrentFileInfo()** which will return an object whose properties represent the current file.

Example of oucGetCurrentFileInfo:

```
filename: "index.html"
id: "/gadget/index.html"
isBinary: false
isImage: false
lastModified: 1455046620000
lockStatus: "checkedIn"
previewUrl: "http://localhost/files/content?path=/gadget/index.html&site=old_gallena"
productionUrl: "http://bdoughertydev.oudemo.com/tester2/gadget/index.html"
site: "old_gallena"
stagingPath: "/gadget/index.html"
type: "html"
```

Here's what your **init()** function should look like.

```
function init() {
    gadget.oucGetCurrentFileInfo().then(function (data) {
        console.log(data); //optional
    });
}
```

/files/properties API Call

Now that we have information regarding which file we are currently using, we have enough information to utilize the OU Campus API to fetch the Page Parameters of that file. Create a new function named `getPageProps` to create an AJAX request to the `'/files/properties'` endpoint, which will return us all the Page Parameters of this file.

```
function getPageProps (page) {
          $.ajax({
              type : 'GET',
              dataType : 'json',
              url : gadget.apihost + '/files/properties',
              data : {
                  site : gadget.site,
                  path : page.stagingPath,
```

```
                authorization_token : gadget.token
            },
            success : function (data) {
                console.log(data);
                //next, prepare data for metadata
            },
            error : function (data) {
            }
        });
    }
```

Our **oucGetCurrentFileInfo()** function callback will need to call the `getPageProps` function passing the **oucGetCurrentFileInfo()** data. You will see that the `path` parameter used is from the **oucGetCurrentFileInfo()** data. After reloading the gadget, if we use the browser developer tools to inspect the GET request of `'/files/properties'` call we should see an HTTP request to `/files/properties`, and a response structured like this:


```
▼{title: "Campus Map", meta_tags: [{http_equiv: null, name: "Description", content: ""}],…}
  ▼meta_tags: [{http_equiv: null, name: "Description", content: ""}]
    ▼0: {http_equiv: null, name: "Description", content: ""}
      content: ""
      http_equiv: null
      name: "Description"
  ▼parameters: [{type: "text", name: "heading", prompt: "Page Heading", section: null, exclude-gadget: false,…},…]
    ▼0: {type: "text", name: "heading", prompt: "Page Heading", section: null, exclude-gadget: false,…}
      alt: "Page title that displays above the main content region."
      exclude-gadget: false
      name: "heading"
      prompt: "Page Heading"
      rows: 0
      section: null
      type: "text"
      value: "Campus Map cccc"
    ▼1: {type: "select", name: "image-slider", prompt: "Image Slider Region", section: null,…}
      alt: "Turns on/off the image slider region above the main content region. (LDP Gallery Asset required)."
      exclude-gadget: false
      name: "image-slider"
    ▼options: [{value: "true", selected: false, option: "On"}, {value: "false", selected: true, option: "Off"}]
      ▶0: {value: "true", selected: false, option: "On"}
      ▶1: {value: "false", selected: true, option: "Off"}
      prompt: "Image Slider Region"
      section: null
      type: "select"
    ▶2: {type: "select", name: "translator", prompt: "Language Translation", section: null,…}
    ▶3: {type: "select", name: "left-enable", prompt: "Enable Column", section: "Left Column Options",…}
    ▶4: {type: "radio", name: "left-nav", prompt: "Directory Navigation", section: null, exclude-gadget: false,…}
    ▶5: {type: "radio", name: "left-quicklinks", prompt: "Quick Links", section: null, exclude-gadget: false,…}
    ▶6: {type: "radio", name: "left-content", prompt: "Content Region", section: null, exclude-gadget: false,…}
    ▶7: {type: "select", name: "right-enable", prompt: "Enable Column", section: "Right Column Options",…}
    ▶8: {type: "radio", name: "right-nav", prompt: "Directory Navigation", section: null,…}
    ▶9: {type: "radio", name: "right-quicklinks", prompt: "Quick Links", section: null, exclude-gadget: false,…}
    ▶10: {type: "radio", name: "right-content", prompt: "Content Region", section: null, exclude-gadget: false,…}
  title: "Campus Map"
```


We now have successfully discovered which page we are on with the **gadgetlib.js** method **oucGetCurrentFileInfo()**, and then used the OU Campus API to retrieve the Page Parameters of that file. Next we will need to prepare our data to be saved to our New Metadata API so that we can refer back to it later!

Preparing the Data and Listening for a Gadget Event

If we use the browser developer tools to inspect the GET request of `/files/properties` call, we can see that the response is significatly different from the request parameters required in a POST request to `/files/properties`. This suggests that we need to do some data parsing to format our data into the proper form that the save (POST) call will be expecting. Create a `prepareData` function which will process our data from the `'/files/properties'` API call.

```
var oldParamMap;
 function prepareData (data) {
       var preppedData = {};
       preppedData.title = data.title;
       if (data.meta_tags.length){
           $.each(data.meta_tags, function (i, data) {
               preppedData['_'+data.name] = data.content;
           });
       }
       if (data.parameters.length) {
           $.each(data.parameters, function (i, data) {
               var val;
               if (data.type === 'select' ||
                       data.type === 'radio' ||
                   data.type === 'checkbox') {
                   $.each(data.options, function(i, opt){
                       if (opt.selected == true) {
                           val = opt.value;
                       }
                   });
               } else {
                   val = data.value;
               }
               preppedData['_'+data.name] = val;
           });
       }
       oldParamMap = preppedData;
   }
```

This function will correctly parse our incoming data and format it so that it is ready to be saved. But before we go on to save this data using our new Metadata API (which we will go over shortly), we should wait until the user clicks save in OU Campus before saving the results. This will prevent unnecessary saves. Otherwise every time the gadget is loaded in a Page view it will trigger a save, creating multiple duplicate save versions.

Receiving the gadget event, and storing the data

As you can see our starter code is listening for a `params-save` event. This event is triggered when the Page Parameters are saved in OU Campus. Create a new function named `saveOldParams` function; this function will be called when the `params-save` event has been triggered.

```
saveOldParams function:
    function saveOldParams (params) {
        //Save in Metadata
    }
params-save event listener:
    'params-save': function (evt, notification) {
        saveOldParams(oldParamMap);
    }
```

When the `params-save` event is triggered we will then save a map of our old page properties via the Metadata API. Inside the `saveOldParams` function we will use our first Metadata API call to create a new metadata record. The Metadata `create` call takes two parameters, `mime_type` and `metadata`. Create a global variable named `MIME_TYPE` to specify the Metadata namespace/bucket/identifier that the data will be saved under. The metadata parameter will come from our `oldParamMap` global variable, and we will need to stringify the JavaScript object because the metadata API only stores strings.

```
var MIME_TYPE = 'UNIQUE_IDENTIFIER_STRING'
function saveOldParams (params) {
    //Save in Metadata
    gadget.Metadata.create({
        mime_type : MIME_TYPE,
        metadata  : JSON.stringify(params)
    }, {
        success : function (data) {
            console.log(data);
            //link to the page
        },
        error : function () { }
    });
}
```

Now we have successfully saved an entry in the Metadata database under our specific namespace that we can refer back to later. Next, we must link our newly created metadata record to an OU Campus data model. You can link metadata to Pages, Directories, Assets, or Sites. In the success callback of the Metadata.create method, we will add a function call to `linkMetaData` passing the response of the create call. Create a new function `linkMetaData`, which will be responsible for creating this link.

```
        function linkMetaData (data) {
            gadget.Metadata.link({
                link_type : 'page',
                mime_type : MIME_TYPE,
```

```
                id        : data.metadata,
                item      : currentFile.stagingPath
            }, {
                success : function (data) {
                    init();
                },
                error : function () { }
            })
        }
```

The Metadata.link method needs a few parameters. We need to specify the `link_type` which tells the backend what type of link or relationship we are creating (in this case `page`), next the id of the newly created metadata, and lastly `item`, which is the path of the page we are linking to.

```
function saveOldParams (params) {
        //Save in Metadata
        gadget.Metadata.create({
            mime_type : MIME_TYPE,
            metadata : JSON.stringify(params)
        }, {
            success : function (data) {
                console.log(data);
                //link to the page
                linkMetaData(data);
            },
            error : function (data) { }
        });
    }
 function linkMetaData (data) {
    gadget.Metadata.link({
        link_type : 'page',
        mime_type : MIME_TYPE,
        id        : data.metadata, //the metadatas id
        item      : currentFile.stagingPath
    }, {
        success : function (data) {
            init();
        },
        error : function (data) {

        }
    });
  }
```

Now we have successfully waited for the `params-save` OU Campus event to be triggered, then created a new metadata representing the old page parameters, stored this in the Metadata database under our custom namespace identifier, and lastly linked this metadata record to an OU Campus page. Let's retrieve this data and actually use it now to revert back to a previous save!

Listing the saved Metadata

When the gadget loads we want to retrieve a list of the last **Page Property** saves that have occurred to give the user a reference point of how far the revert is going back. To do this we will use the Metadata.list API call to retrieve a list of our saved Metadata. Create a `getSaveHistory` function.

```
function getSaveHistory () {
        gadget.Metadata.list({
            mime_type : MIME_TYPE,
            item      : currentFile.stagingPath,
            item_type : 'page'
        }, {
            success : function (data) {},
            error : function () {}
        })
    }
```

This will retrieve us a list of metadatas given the `MIME_TYPE` identifier. Specifying the item and `item_type` will limit the results to metadata records that have been linked to, in this case, `'page'`. With this data we can render a list of the save history.

Create a `renderView` function. This function will be called in the success callback of the `getSaveHistory` function. We will use it to enable/disable the revert button and render the save history list.

```
function renderView (data) {
   $('#main ol').empty();
   $('#revert-btn').attr('disabled', true);

   if (data.length) {
       $('#revert-btn').attr('disabled', false);
   } else {
       $('#main ol')
       .prepend('<div class="list-group-item">No Last Saves</div>');
   }
   $.each(data, function (i, item) {
        var div = '<div class="list-group-item">' +
                  new Date(item.created).toGMTString() +
```

```
                    '</div>';
            $('#main ol').prepend(div);
    });
}
var metadataList;
function getSaveHistory () {
    gadget.Metadata.list({
        mime_type : MIME_TYPE,
        item      : currentFile.stagingPath,
        item_type : 'page'
    }, {
        success : function (data) {
            metadataList = data;
            renderView(data);
        },
        error : function () {}
    });
}
```

As you can see we also set a value to a global variable `metadataList`. We will use this to reference our `metadataList` shortly. After refreshing you should now see a rendered history of page parameter saves or “No Last Saves.”

Reverting Page Properties

Lastly, let's make this revert button work. In the document ready function just below the **init()** call add a jQuery on click listener to the revert button.

```
$('#revert-btn').on('click', function () {
    revertProperties();
});
```

When this button is clicked it will call a method named `revertProperties`. The `revertProperties` method will send a POST API call to `` `/files/properties` `` which will save and update the properties back to the old values based on what we have stored with the Metadata API. Here we will create another AJAX request.

```
function revertProperties () {
    var props = JSON.parse(metadataList[metadataList.length - 1].metadata);
    props.site = gadget.site;
    props.path = currentFile.stagingPath;
    props.authorization_token = gadget.token
    $.ajax({
        type     : 'POST',
```

```
        dataType : 'json',
        url      : gadget.apihost + '/files/properties',
        data     : props,
        success : function (data) {
            //delete last metadata
            removeLastSave();
            //reload page
            gadget.oucSetCurrentLocation('/pageparameters' +
currentFile.stagingPath);
            gadget.oucSetCurrentLocation('/preview' +
currentFile.stagingPath);         },
        error : function () {}
    });
}
```

First we set some parameters required by the `` `/files/properties` `` endpoint, and in the success callback we also call a method we will create `removeLastSave`, which will delete the last saved metadata, so that our next revert is not the same as the previous. We also utilize the `oucSetCurrentLocation` function from the gadgetlib to "reload" the page so that we can see our changes. This navigates us to the params view then immediatley back to the preview of the page.

The last order of business is to implement the `removeLastSave` method. Create a function `removeLastSave`.

```
function removeLastSave () {
        gadget.Metadata.delete({
            mime_type : MIME_TYPE,
            id        : metadataList[metadataList.length - 1].id
        }, {
            success : function (data) {},
            error : function () {}
        });
    }
```

This method calls the Metadata API delete function which deletes a metadata record. The required parameters are to specify the `MIME_TYPE` and id of the metadata that is to be deleted.