akumina

Akumina Digital Workplace

Developer Guide

Version 1.1 (May 2016)

Table of Contents

An Introduction to the Digital Workplace framework

Before getting started with the rest of this document, it makes sense to understand the overall strategy that has been used to develop the Digital Workplace. We tried to create a framework that was both friendly to basic and advanced Javascript developers.

There are 4 basic concepts to understand to get started with the DW:

- "Callbacks" (i.e. functions) – Mostly everything in javascript is an object
  - Assign a function to a variable
  - Store a function as a value in an array or object
  - Pass a function to a function
  - Return a function from a function
- Eventing – pub sub model
  - First you subscribe to an event, with a 'function'
  - Then you publish an event, and that function is executed.
- Binding Data to an HTML 'Template' – Handlebars/Mustache
- Initialization vs. Render

Understanding "Callbacks"

Callbacks in short, is away to pass a function to another function and let it execute it in the context of its scope. Lets take a look at a simple piece of code that shows this better:

```
function myCustomCallback(data){
  alert('hello '+ data );
}

function internalCall(callBack){
  var test = "testing!";
  callBack(test);
}

internalCall(myCustomCallback);
```

The above could will output 'hello testing!' – as you can see this concept is very powerful and will be used throughout the DW framework.

Understanding Eventing

Below is a sample of a 'subscribing' to an EVENT, and wiring up a function to be executed when that event is 'published'

```
Akumina.Digispace.AppPart.Eventing.Subscribe('/my event/', myfunction);

function myfunction(){
  alert('my function executed');
}
```

Is the same as:

```
Akumina.Digispace.AppPart.Eventing.Subscribe('/myevent/', function(){ alert('my
function exected'} );
```

The important thing to understand is that the above 'my function executed alert' will not be executed until the event '/myevent/' has been 'published'. This allows us to separate code throughout the application without creating direct dependencies. We can simply make note of which parts of the application are firing events so that we can subscribe to them to extend various functionality without changing the core code base or having access to the code. As long as you know which events are fired when, you can subscribe to them and extend the application in very powerful ways.

So now that the above code has subscribed to those events, anytime I call the following code, my alert message will appear:

```
Akumina.Digispace.AppPart.Eventing.Publish('/myevent/');
```

This eventing concept will be important to understand as you create widgets that are 'Initialized' and then 'Rendered'

There are also many events that are fired throughout the lifecycle of the page load of the DW system that we will cover later on in the document.

Binding data to an HTML Template

The DW framework is utilizing an templating system known as "Handlebars" or "Mustache", this allows developers to separate data from presentation in a very straight forward manner.

Suppose you have the following data:

```
var myData = { FirstName: "John", LastName: "Doe", Skills:
['Javacript','PHP','.NET']};
```

Suppose you have the following HTML Markup given to you by your design team:

```
<div id="sampledata">
<h1>Sample</h1>
First Last has the following Skills:
<ul>
  <li>Skill 1</li>
  <li>Skill 2</li>
  <li>Skill 3</li>
</ul>
</div>
```

You could certainly write JS code like this:

```
$("#sampledata").append("<h1>Sample</h1>" + myData.FirstName + " " +
myData.LastName + " has the following Skills: <ul>" … etc etc
```

The problem with this approach is it tightly couples your data to your presentation and does not allow a developer to change it without having access to the source. Let's discuss how Handlebars/Mustache within the DW framework helps separate data from presentation with the following sample:

Your markup should be put in its own file and hosted within SharePoint for ease of management:

```
<div id="sampledata">
<h1>Sample</h1>
{{FirstName}} {{LastName}} has the following Skills:
<ul>
{{#Skills}}
  <li>{{.}}</li>
{{/Skills}
</ul>
</div>
```

So lets expose the above markup at the following URL:
/Style%20Library/DigitalWorkplace/Content/Templates/Sample.html

The following DW APIs will automatically map your data through the HTML template so it can be injected into a target selector.

```
var myData = { FirstName: "John", LastName: "Doe", Skills:
['Javacript','PHP','.NET']};
var templateUri = "/Style%20Library/DigitalWorkplace/Content/Templates/Sample.html";

new Akumina.Digispace.AppPart.Data().Templates.ParseTemplate(templateUri,
myData).done(function (html) {
                $("#sampledata").html(html);
});
```

So with a few lines of code we can take data and map it to any HTML template.

Initialization and Render

Please make note that anytime you see Init() or Initialize, think SUBSCRIBE to an EVENT.

Whenever you see Render() or Render, think Go Get Data…

The widget framework is built on these 2 premises.

So Init() within a widget, will most likely show a 'Loading…' message. Render() will get data and update the view.

Core Components of the Digital Workplace

Important Sharepoint Lists specific to Digital Workplace

This section was not meant to discuss all lists created and used in DW, simply to provide information on lists geared towards a developer's understanding.

- DigispaceConfigurationIDS_AK

  This is the primary list used for settings and configuration around the DW site. There is a control app 'Digital Workplace Configuration' that controls the data in this list. This information is mapped to the ConfigurationContext that will be covered later on in the document

- DashboardWidgets_AK

  Available widgets that you want to be made available to the user's dashboard. We will cover more information around how to do this later on in the document. Widgets can be enabled and disabled here. There will be a record for each widget that you want to be enabled. Currently there is no Management App for this, it will exist in later versions of the DW.

- MyDashboard_AK

  Storage of selected widgets for the logged in users dashboard. There will be a record for each user who configures their dashboard

- Rail_AK

  The user nav AKA, "The RAIL" is bound to the data in this list. If you would like to disable or enable items in the Rail, please see information around the 'Rail Manager' Management App inside of your App Manager.

Master Page References

There are a total of 6 references included in the DW master page. 4 scripts and 2 styles.

- Scripts
  - digitalworkplace.vendor.min.js
  - digitalworkplace.vendor-nomin.js
    - 3rd party un-minififed vendor files
  - digitalworkplace.min.js
    - The minfied CORE script files from Akumina Engineering – this file is off limits and will be overwritten during upgrades
  - digitalworkplace.custom.js
    - A 'blank' reference created to add additional functionality specific to your implementation. This allows you to make changes without changing the core framework and is not touched during the upgrade process.
- Styles
  - digitalworkplace.css

        - The CORE style files from Akumina Engineering – this file is off limits and will be overwritten during upgrades
    - digitalworkplace.custom.css
        - A 'blank' reference created to add additional styling specific to your implementation. This allows you to make changes without changing the core styles and is not touched during the upgrade process.

Adding Additional Markup to the masterpage procedure

During the page load lifecycle the contents of '/Style%20Library/DigitalWorkPlace/Content/Templates/AdditonalMasterMarkup.html' is appended to the following DOM element within the Master Page – this allows you to additional markup dynamically instead of altering the master page.

```
<div class="interAction" id="digiAdditionalMarkup"></div>
```

App Manager apps

The following apps have been added in v3.3 of the App Manager

- Management Apps
    - Rail Manager
        - Manages the contents of 'RAIL_AK'
    - Department Manager
        - Manages the department COLOR and handles user department mapping to the department site collection
    - People Directory Sync
        - Downloads the data from Graph into local cache for more advanced querying used by the DW
- Control Apps
    - Digital Workplace Configuration
        - Managed the contents of 'DigiSpaceConfigurationIDS_AK'

Page Lifecycle - Shipped Steps

The Digital Workplace has its own Page Lifecycle with various events that fire as each step is executed. These steps do anything from getting data, populating markup, provide "Loading…" feedback to the user, updating the User and Configuration Contexts. You must understand where in the lifecycle you should execute your code. For instance, if you use the $(document).ready() event, you may not have the markup populated when that event fires. The Digital Workplace actually starts its page lifecycle on $(document).ready() and when the SP.js file is loaded by Sharepoint.

From here on out it is Best Practice TO NOT USE $(document).ready()

```
$(document).ready(function () {
    SP.SOD.executeFunc('sp.js', 'SP.ClientContext', function () {
Akumina.Digispace.Loader.Init(GetSteps()); });
    SP.SOD.executeOrDelayUntilScriptLoaded(function () {}, "sp.js");
});
```

If you wanted to do something in the $(document).ready() you should change your code to use the '/loader/completed/'event

```
Akumina.Digispace.AppPart.Eventing.Subscribe('/loader/completed/', function(){ myFunction(); });
```

The following steps represent page load lifecycle of DW:

1. Get Footer Markup template ($(document).ready())
    a. Get contents of
2. Get Additional Markup template
3. Display Loader
4. Event Subscription
5. Get Digispace Configuration (ConfigurationContext populated)
    a. This binds the configuration from DigiSpaceConfigurationIDS_AK into the ConfigurationContext
6. Get Loading Template
7. Display Site Logo
8. Initialize Search Page
9. Index Page Data
10. Get Interchange LoginURL
    a. This sets up the correct App Redirect URL and adds it to the ConfigurationContext
11. Get Graph Token
12. Get User Properties (UserContext populated)
13. Initializing Rail
14. Initialize Generic Controls
15. Load Dashboard Widgets
16. Initialize Search Fields
17. Debug Info ('/loader/completed' event fired)

Caching

The DW framework utilizes a 3rd party component called 'LSCACHE' for managing localStorage and cache expiration

```
Akumina.AddIn.Cache.Get(key);
Akumina.AddIn.Cache.Set(key, data, seconds);
```

Caching Strategy

The overall caching strategy for DW can be configured in the control app 'Digital Workplace Configuration', this means any widget that has not been configured for caching will fall back to this strategy. This is quick way to apply caching throughout the site. Not all widgets are wired up to this caching strategy.

```
Akumina.Digispace.ConfigurationContext.CachingStrategyInterval

"light" - Akumina.AddIn.Constants.MIN_CACHE_EXPIRATION;//1 min
"medium" - Akumina.AddIn.Constants.HOUR_CACHE_EXPIRATION;//1 hr
"heavy" - Akumina.AddIn.Constants.LONG_CACHE_EXPIRATION;//6 hrs
"extreme" - Akumina.AddIn.Constants.DAY_CACHE_EXPIRATION;//24 hrs
```

Important Javascript APIs

This section will provide a quick list of helpful APIs that will speed up your development

1. __getTemplatePrefix()
    a. Gets your site collection url
2. Akumina.Digispace.ConfigurationContext.TemplateFolderName

```
__getTemplatePrefix() + "/Style%20Library/" + Akumina.Digispace.ConfigurationContext.TemplateFolderName
+ "/Content/Templates/contentBlock/Default.html";
```

3. Akumina.AddIn.Utilities.getEditMode();
    a. Will tell you if you are in edit mode or not
4. Akumina.AddIn.Logger.WriteInfoLog
5. Akumina.AddIn.Logger.WriteErrorLog
6. Akumina.AddIn.Logger.Start/StopTraceLog

```
Akumina.AddIn.Logger.StartTraceLog('Tracing for whatever');
Akumina.AddIn.Logger.StartStopLog('Tracing for whatever');
```

7. More to come…

UserContext

The UserContext is an object that contains important information about the logged in user, you can access this object while inside the DW framework. The follow code should be self explanatory and should give you a good idea of its use. This object will be expanded over time.

```
Akumina.Digispace.UserContext.City
Akumina.Digispace.UserContext.State
Akumina.Digispace.UserContext.PostalCode
Akumina.Digispace.UserContext.Department
Akumina.Digispace.UserContext.DisplayName
Akumina.Digispace.UserContext.JobTitle
Akumina.Digispace.UserContext.userGroups
```

ConfigurationContext

The ConfigurationContext is an object that stores information around the configuration of the DW. Most of this information can be found in the ‘DigiSpaceConfiguration_AK’ and is automatically mapped to this object by one of the Shipped Steps. The below are some of the configuration settings available, this object will be expanded over time.

```
Akumina.Digispace.ConfigurationContext.EnableAzureAD
Akumina.Digispace.ConfigurationContext.EnableDebugMode
Akumina.Digispace.ConfigurationContext.CachingStrategyInterval
Akumina.Digispace.ConfigurationContext.GraphSubscriptionId
Akumina.Digispace.ConfigurationContext.GraphClientId
Akumina.Digispace.ConfigurationContext.InterchangeURL
Akumina.Digispace.ConfigurationContext.InterchangeLoginURL
Akumina.Digispace.ConfigurationContext.SiteLogoURL
Akumina.Digispace.ConfigurationContext.TenantUrl
Akumina.Digispace.ConfigurationContext.LoadingTemplateHtml
```

Developer Debug Page

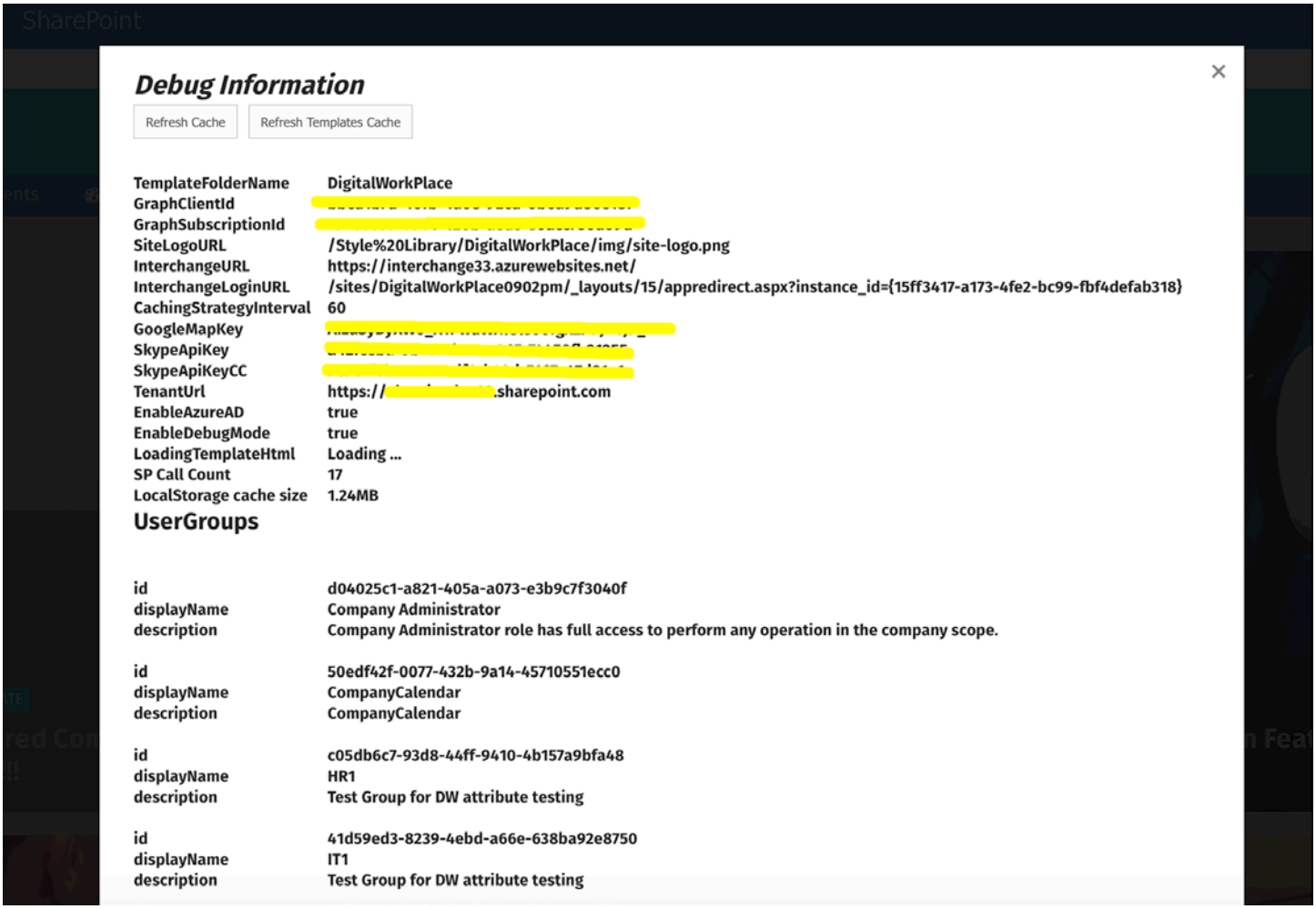
The following functions are exposed to a developer in order to provide quick access to:

- Resetting the localStorage cache
- Resetting the DisplayTemplate localStorage cache
- Understanding the ConfigurationContext and what settings are currently applied
- Understanding the UserContext and information around the currently logged in user

You can access the debugger screen by using the following key strokes: CTRL+UP ARROW

This can be disabled by setting EnableDebug to “false” in the ‘DigiSpaceConfigurationIDS_AK’

Here is an example of what happens when you press CTRL+UP ARROW when EnableDebug is set to ‘true’


SharePoint
Debug Information
Refresh Cache
Refresh Templates Cache
TemplateFolderName DigitalWorkPlace
GraphClientId
GraphSubscriptionId
SiteLogoURL /Style%20Library/DigitalWorkPlace/img/site-logo.png
InterchangeURL https://interchange33.azurewebsites.net/
InterchangeLoginURL /sites/DigitalWorkPlace0902pm/_layouts/15/appredirect.aspx?instance_id={15ff3417-a173-4fe2-bc99-fbf4defab318}
CachingStrategyInterval 60
GoogleMapKey
SkypeApiKey
SkypeApiKeyCC
TenantUrl https://.sharepoint.com
EnableAzureAD true
EnableDebugMode true
LoadingTemplateHtml Loading ...
SP Call Count 17
LocalStorage cache size 1.24MB
UserGroups
id d04025c1-a821-405a-a073-e3b9c7f3040f
displayName Company Administrator
description Company Administrator role has full access to perform any operation in the company scope.
id 50edf42f-0077-432b-9a14-45710551ecc0
displayName CompanyCalendar
description CompanyCalendar
id c05db6c7-93d8-44ff-9410-4b157a9bfa48
displayName HR1
description Test Group for DW attribute testing
id 41d59ed3-8239-4ebd-a66e-638ba92e8750
displayName IT1
description Test Group for DW attribute testing

Creating your own Widget to be used on all pages with a snippet

If you want to create your own widget to interact with SharePoint data, or maybe to query a 3rd party system, you can speed up your development by utilizing the DW snippet framework.

1. Determine the name of your custom widget and what you want to define for properties that may be accepted by the end users of your widget. This will be the HTML used when inserting the widget into a content editor.

```
<div id="randomguid" class="ak-controls ak-mycustom-widget"
ak:test="value"></div>
```

2. The above snippet class 'ak-mycustom-widget' can be called whatever you want. If you want to collect properties, simply prepend them with 'ak:' – such as the example, 'ak:test'
3. The 'id' attribute with a value of randomguid should be replaced with a unique id, it doesn't have to be a guid, it can be something like 'mycustomwidget1234'
4. Next, download digiworkplace.custom.js from "/Style Library/DigiWorkplace/JS" folder in SharePoint

```
if ((typeof Akumina.AddIn.MyCustomWidget) === 'undefined') {
    Akumina.AddIn.MyCustomWidget = function () {
        var _cur = this;

        this.Init = function (properties) {
            this.properties = properties;
            _cur.Prerender();
        };

        this.Prerender = function () {
            $("#" + _cur.properties.id).html("Loading...");
            Akumina.Digispace.AppPart.Eventing.Subscribe('/loader/completed/',
_cur.Render);
        };

        this.Render = function () {

            $("#" +
_cur.properties.id).html("Akumina.AddIn.Custom.MyCustomWidget has loaded!!
WOW!!!! Here is your property: " + _cur.properties.test);

        };
    }
}
```

5. Now, create a 'callback' that will be utilized to initialize your widget, this callback will Create an instance of your widget and call the Init() method. The Init() method will bind the properties passed into it and also call the PreRender() method which subscribes to the '/loader/completed/' event. This event is fired at the end of the DW page lifecycle and will cause the Render() method to be executed. The Render() method is where you

should go get your data. Lets call this 'callback' – 'initMyCustomWidget()'. This gives your users the "Loading…" experience and is much more enjoyable then holding up the presentation of the page.

```
function initMyCustomWidget() {

    $('.ak-mycustom-widget').each(function (index) {
            var props = Akumina.AddIn.Utilities.getProperties($(this));
            var myCustomWidget = new Akumina.AddIn.MyCustomWidget();
            myCustomWidget.Init(props);
    });
}
```

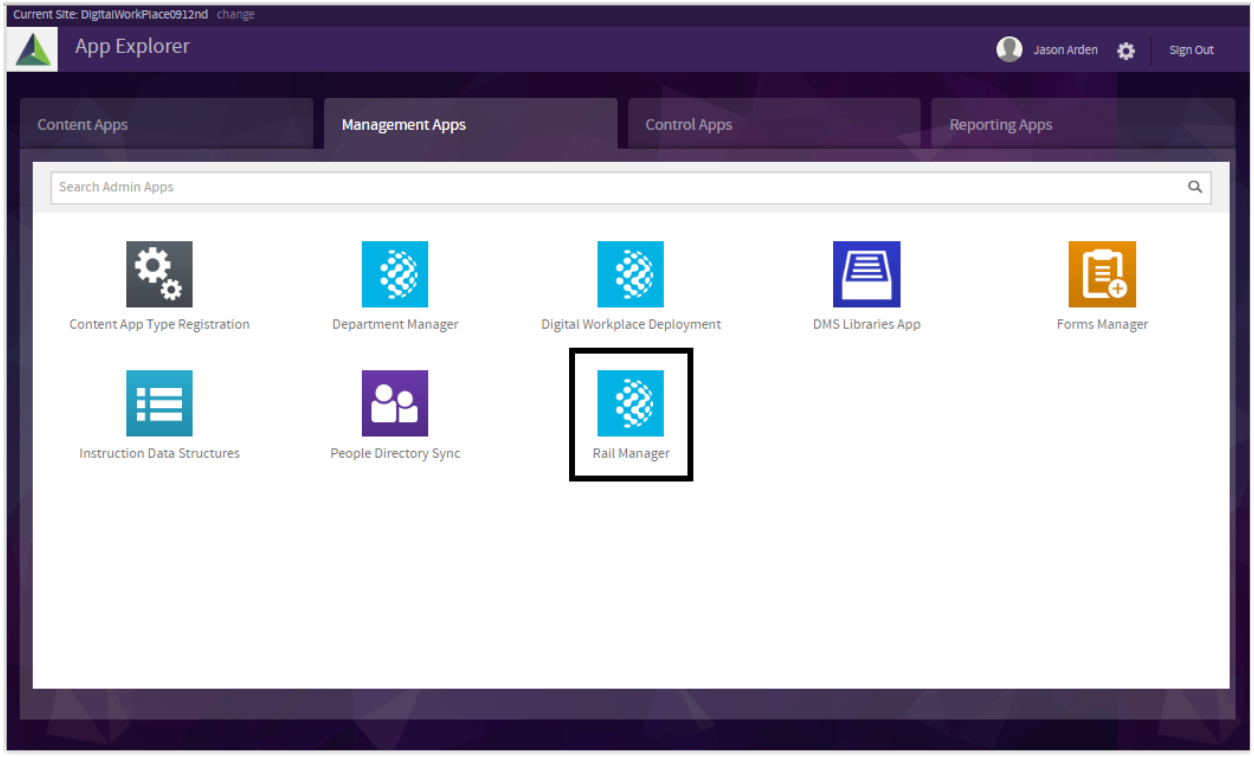
6. Looking at the above code, it is important to note the Helper method 'getProperties'. This method handles getting the 'ak:' properties from the HTML snippet you created above. Those properties are then passed into the Init() method.
7. Finally, we need to subscribe the initMyCustomWidget to an event as we do not want to call the initMyCustomWidget too early. The DW framework has a specified order in which it loads data and properties that we talked about above under Shipped Steps. If you want access to the UserContext or ConfigurationContext you would not want to initialize your widget before this information is available.

```
Akumina.Digispace.AppPart.Eventing.Subscribe('/loader/eventsubscription/',
initMyCustomWidget);
```

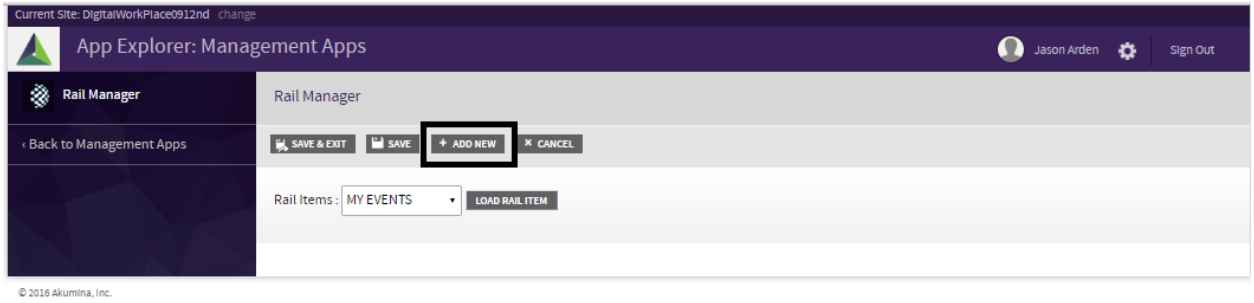
8. Upload updated digiworkplace.custom.js to "/Style Library/DigiWorkplace/JS" folder in SharePoint
9. Be sure to FLUSH your cache by using the CTRL+UP arrow and clicking on Refresh Cache, or by clearing localStorage with localStorage.clear();

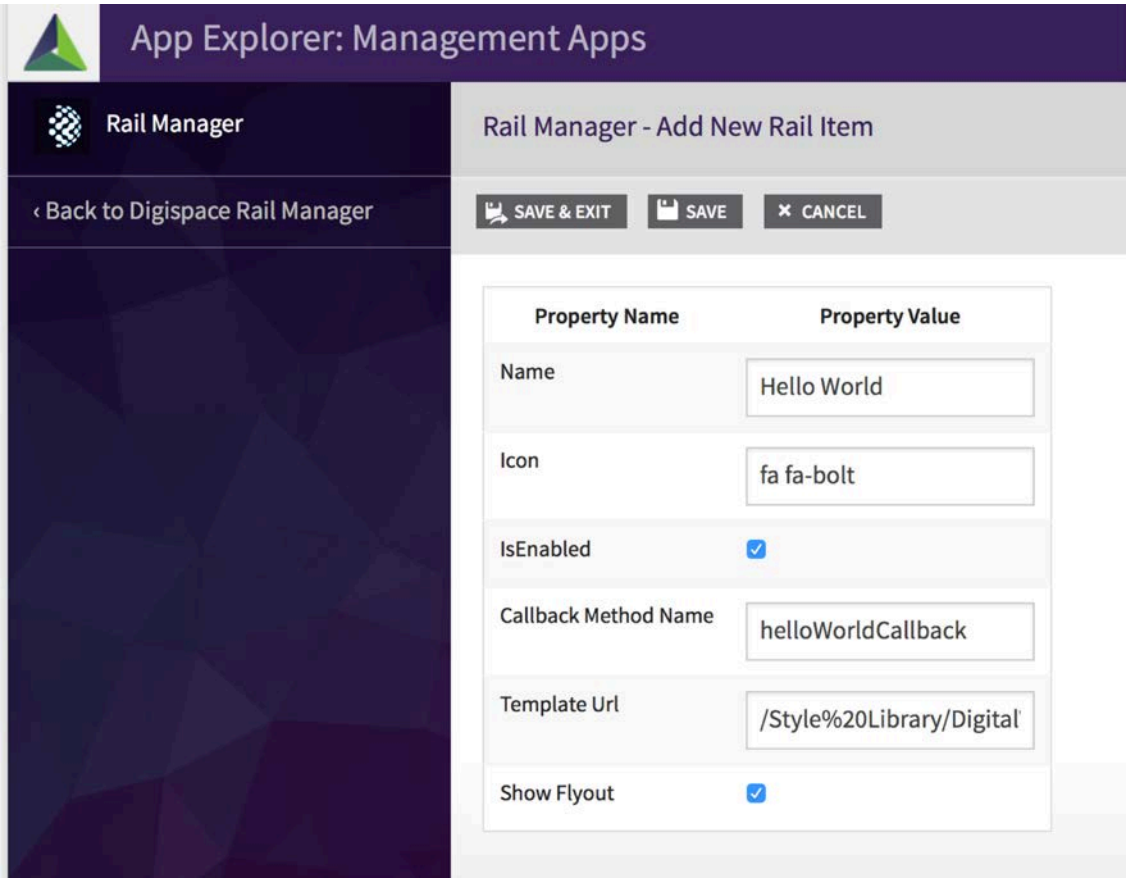
Adding a new item to the left rail

1. Login into InterChange. Go to "Management Apps" and Click "Rail Manager"



2. Click “Add New” button and fill in the details for the Rail Item





3. In the TemplateURL field use the following example path: /Style%20Library/DigitalWorkPlace/Content/Templates/Rail/RailHelloWorldTemplate.html

The contents of the file looks like this:

```
<h1>Hello World!!!</h1>

<p>
    {{MyContent}}
</p>
```

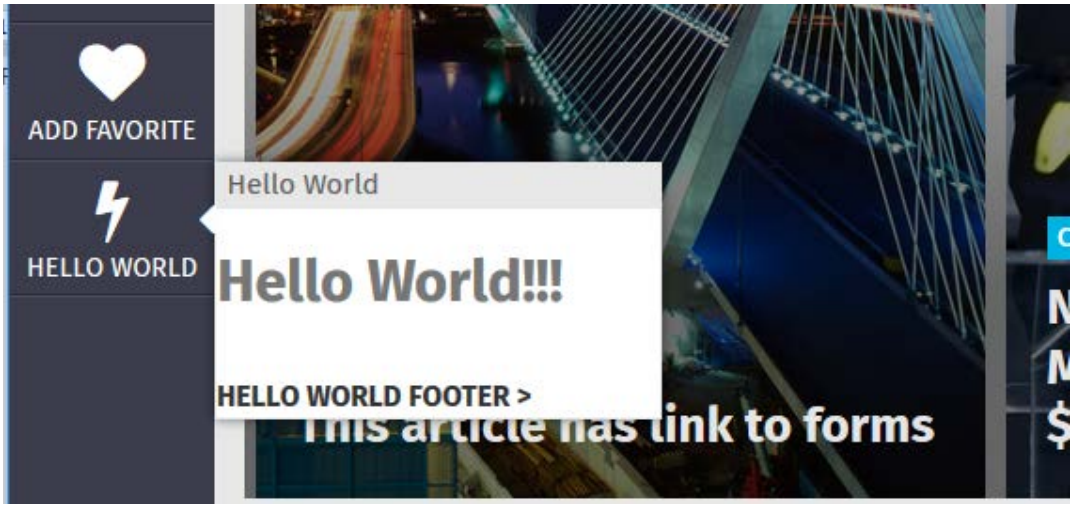
4. Download digiworkplace.custom.js from "/Style Library/DigiWorkplace/JS" folder in SharePoint
5. Add the code for callback method of the "Hello World" rail item in digitalworkplace.custom.js
    a. The "helloWorldCallback()" was the method name used in the Rail Manager App under 'Call back method
    b. The railItemModel is the model used to populate the HEADER, the contents of the fly out and the FOOTER, where you would usually present an option for "more"..

```
function helloWorldCallback(item) {
    var def = $.Deferred();
    var title = $(item).find('span.ak-nav-title').text();
    var railItemModel = {
        "Header": title, "MyContent": "Hello World Rail Item Content", "Footer":
{ Text:"Hello World Footer", Link:"#"}
    };
    def.resolve(railItemModel);
    return def;
}
```

If you wanted to simply have the click go somewhere, instead of a fly out, change the flout property on the Rail Manager and then use this code in your call back

```
function helloWorldCallback (item) {
    window.location = __getTemplatePrefix() + "/Pages/Dashboard.aspx";
}
```

6. Upload the updated digitalworkplace.custom.js to "/Style Library/DigitalWorkplace/JS" folder on SharePoint
7. The newly added rail item will appear as follows:



Adding a new dashboard widget that uses the GenericList widget

Down below we cover a more advanced scenario for wrapping a generic list widget in your custom widget, but maybe you just want to add a new dashboard widget by utilizing the existing widgets. In this scenario we use the 'Akumina.AddIn.GenericListControlWidget' and a custom callback, and a custom display template.

1. Download digiworkplace.custom.js from "/Style Library/DigiWorkplace/JS" folder in SharePoint
2. Add your custom callback to digiworkplace.custom.js

```
//Callback for Digispace Configuration Dashboard Widget based on Generic List
Control
function DashboardShowConfiguration(data) {
    var configuration = {};
    configuration.Items = [];
    for (var i = 0; i < data.Items.length; i++) {
        configuration.Items.push({ Key: data.Items[i].Title, Value:
data.Items[i].Value });
    }
    configuration.HasItems = configuration.Items.length > 0;
    configuration.WebPartTitle = "My Configuration Dashboard Widget";

    return configuration;
}
```

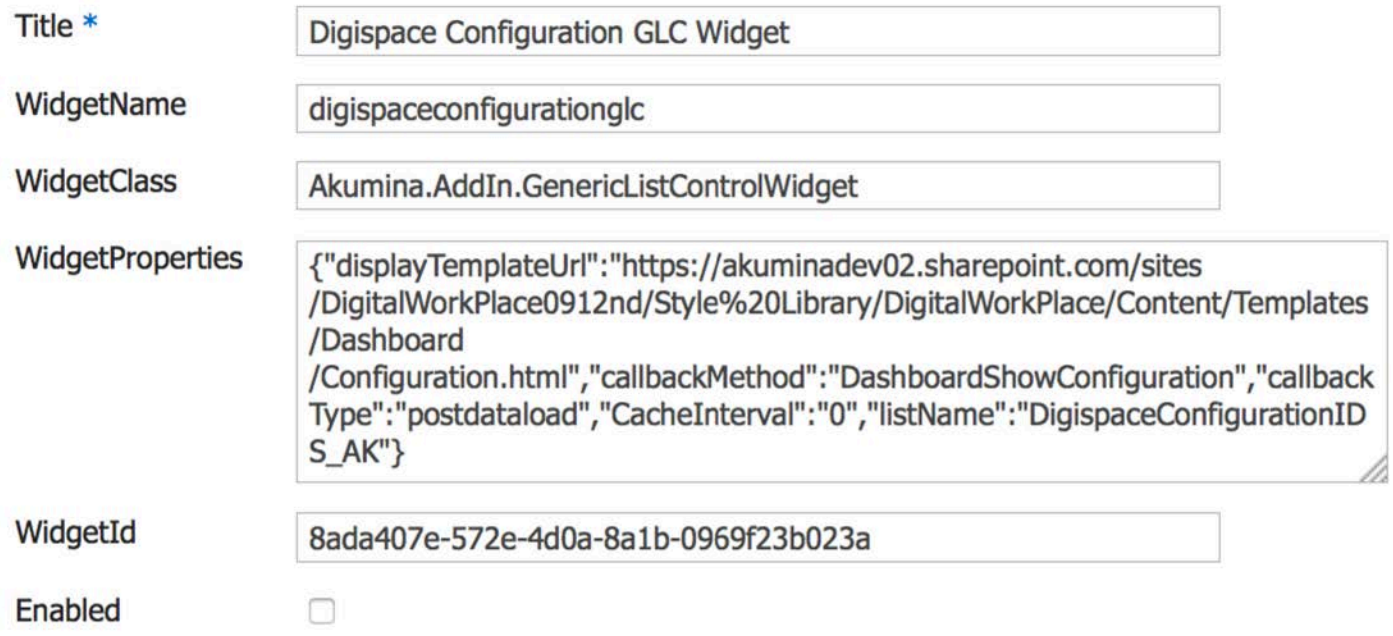
3. Upload updated digiworkplace.custom.js to "/Style Library/DigiWorkplace/JS" folder in SharePoint
4. Define a name for your "View" and upload it into SharePoint, we recommend a name such as 'Configuration.html' and a path such as: /Style%20Library/DigitalWorkPlace/Content/Templates/Dashboard/Configuration.html

The contents of the file looks like this:

```
<h3 class="ak-dashboard-header">{{WebPartTitle}}</h3>
<div>
    {{#if HasItems}}
    <table>
        <thead>
            <tr><th>Key</th><th>Value</th></tr>
        </thead>
        {{#Items}}
            <tr><td>{{Key}}</td><td>{{Value}}</td></tr>
        {{/Items}}
    </table>
    {{else}}
        No configuration Items available.
    {{/if}}
```

```
</div>
```

5. Upload the Configuration.html to "/Style Library/DigiWorkplace/Content/Templates/Dashboard/" folder in SharePoint
6. Now we need to wire up your CALLBACK, and the VIEW so that the GenericListControlWidget can utilize them by registering your Widget code with the Dashboard framework by inserting a new record into the 'DashboardWidgets_AK" list. In the future we will provide a management app to accomplish this and make it easier.



Title – unique title for your widget

WidgetName – this is the internal name for your widget, do not use spaces

Widget Class – We are reusing an existing widget – Akumina.Addin.GenericListControlWidget

WidgetProperties – this is where you define additional properties to be passed into the GenericListControl widget. We are using the following JSOM properties, they should be self explanatory:

```
{"displaytemplateurl":"https://tenant.sharepoint.com/sites/sitecollection/Style%20Library
/DigitalWorkPlace/Content/Templates/Dashboard/Configuration.html","callbackmethod":"Dashb
oardShowConfiguration","callbacktype":"postdataload","cacheinterval":"0","listname":"Digi
spaceConfigurationIDS_AK"}
```

- displayTemplateUrl
- callbathMethod
- callbackType
- CacheInterval
- listName

WidgetId – this can be any string you would like we recommend using GUIDs so that it will never conflict with other widgets

7. Be sure to FLUSH your cache by using the CTRL+UP arrow and clicking on Refresh Cache, or by clearing localStorage with localStorage.clear();

All Done, you just created a widget that takes data out of SharePoint and binds it to a view with NO SHAREPOINT code.

Adding a new custom dashboard widget

1. Download digiworkplace.custom.js from "/Style Library/DigiWorkplace/JS" folder in SharePoint
2. Add dashboard control code to digiworkplace.custom.js

For our sample widget we are going to write a widget that gets data out of the SharePoint list called 'DigispaceConfigurationIDS_AK' and bind its data through a callback called 'DashboardShowConfiguration'

```
if ((typeof Akumina.AddIn.DigispaceConfigurationWidget) === 'undefined') {
    Akumina.AddIn.DigispaceConfigurationWidget = function () {
        var _cur = this;
        this.GetPropertyValue = function (propertyValue, defaultValue) {
            return (propertyValue == undefined ||
propertyValue.toString().trim() == "") ? defaultValue : propertyValue;
        };
        //Set any property overrides here
        this.SetDefaultsProperties = function (request) {
            request.callbackMethod = "DashboardShowConfiguration";
            request.callbackType = "postdataload";
            request.CacheInterval = "0";
            request.listName = "DigispaceConfigurationIDS_AK";
            request.displayTemplateUrl =
_cur.GetPropertyValue(request.displaytemplateurl, "");
            request.SenderId = _cur.GetPropertyValue(request.SenderId, "");
            return request;
        };

        this.Init = function (dashboardWidgetRequest) {
            _cur.widgetRequest =
_cur.SetDefaultsProperties(dashboardWidgetRequest);
            _cur.genericListControl = new
Akumina.AddIn.GenericListControlWidget();
            _cur.genericListControl.Init(_cur.widgetRequest);
```

```
        };
        this.Render = function () {
            _cur.genericListControl.Render();
        };
    }
}
```

The callback used above is what the Generic List control will use to bind the data to once it gets the data back from SharePoint. This method name must match the property 'callback' value above. The input parameter to the call back, 'data' will contain the data from SharePoint list. It is up to you to decide how what UI model you want to create a bind to the UI. Simply return the model.

```
//Callback for Digispace Configuration Dashboard Widget based on Generic List
Control
function DashboardShowConfiguration(data) {
    var configuration = {};
    configuration.Items = [];
    for (var i = 0; i < data.Items.length; i++) {
        configuration.Items.push({ Key: data.Items[i].Title, Value:
data.Items[i].Value });
    }
    configuration.HasItems = configuration.Items.length > 0;

    return configuration;
}
```

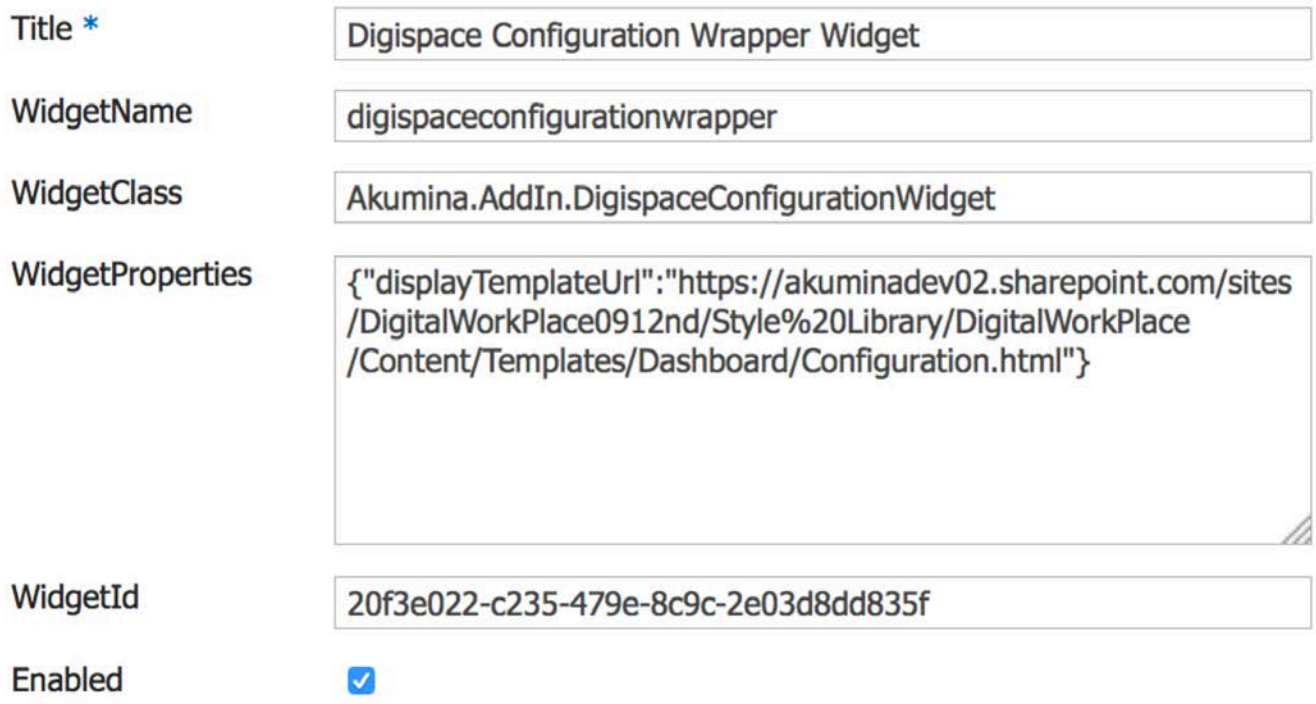
3. Upload updated digiworkplace.custom.js to "/Style Library/DigiWorkplace/JS" folder in SharePoint
4. Define a name for your "View" and upload it into SharePoint, we recommend a name such as 'Configuration.html' and a path such as: /Style%20Library/DigitalWorkPlace/Content/Templates/Dashboard/Configuration.html

The contents of the file looks like this:

```
<h3 class="ak-dashboard-header">{{WebPartTitle}}</h3>
<div>
    {{#if HasItems}}
    <table>
        <thead>
            <tr><th>Key</th><th>Value</th></tr>
        </thead>
        {{#Items}}
            <tr><td>{{Key}}</td><td>{{Value}}</td></tr>
        {{/Items}}
    </table>
```

```
    {{else}}
        No configuration Items available.
    {{/if}}
</div>
```

5. Upload the Configuration.html to "/Style Library/DigiWorkplace/Content/Templates/Dashboard/" folder in SharePoint
6. Finally, register your Widget code with the Dashboard framework by inserting a new record into the 'DashboardWidgets_AK" list. In the future we will provide a management app to accomplish this and make it easier.



Title – unique title for your widget

WidgetName – this is the internal name for your widget, do not use spaces

WidgetClass – this is the namespace/class of your widget, make sure your widget has an Init() and Render() method!

WidgetProperties – this is where you define additional properties to be passed into your widget, you can hardcode these in your widget, however in the future we will be providing an editing interface to these properties. For now we are only using the displayTemplateUrl property:

```
{"displaytemplateurl":"https://tenant.sharepoint.com/sites/sitecollection/Style%20Library/DigitalWorkPlace/Content/Templates/Dashboard/Configuration.html"}
```

WidgetId – this can be any string you would like we recommend using GUIDs so that it will never conflict with other widgets

7. Be sure to FLUSH your cache by using the CTRL+UP arrow and clicking on Refresh Cache, or by clearing localStorage with localStorage.clear();

Additional Resource Links

LSCACHE

https://github.com/pamelafox/lscache/blob/master/README.md

HANDLEBARS

http://handlebarsjs.com/

LOCAL STORAGE

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API