Web AppBuilder for ArcGIS
Creating New Themes
Yiwei Ma & Kevin Gao
Kevin Gao
User Interface Engineer
Yiwei Ma
Software Developer
Web AppBuilder Themes
Out-of-Box Themes

Billboard  Box  Dart  Foldable  Dashboard

Jewelry Box  Launchpad  Plateau  Tab  Custom
Themes: Components
Major Components In A WAB Theme

- Layout
- Panel
- Style
- Controller
What Composes a WAB Theme?

- Layout
- Panel
- Style
- Controller
A Typical Layout Structure

- **widgetOnScreen**: Includes layout properties for widgets placed on the UI, such as a controller, predefined widgets, and widget placeholders. And the panel to use to open the widgets.

- **map**: Defines the position of the map.

- **widgetPool**: Includes a list of widgets placed in the controller widget if there is a controller defined in the widgetOnScreen section. And the panel to use to open the widgets.

- **mobileLayout**: Optional. It provides the layout for mobile devices, overriding the properties configured in the widgetOnScreen, map, and widgetPool sections.
A Typical Layout Structure

- **widgetOnScreen**: Includes layout properties for widgets placed on the UI, such as a controller, predefined widgets, and widget placeholders. And the panel to use to open the widgets.

- **map**: Defines the position of the map.

- **widgetPool**: Includes a list of widgets placed in the controller widget if there is a controller defined in the widgetOnScreen section. And the panel to use to open the widgets.

- **mobileLayout**: Optional. It provides the layout for mobile devices, overriding the properties configured in the widgetOnScreen, map, and widgetPool sections.
A Typical Layout Structure

- `widgetOnScreen`: Includes layout properties for widgets placed on the UI, such as a controller, predefined widgets, and widget placeholders. And the panel to use to open the widgets.
- `map`: Defines the position of the map.
- `widgetPool`: Includes a list of widgets placed in the controller widget if there is a controller defined in the `widgetOnScreen` section. And the panel to use to open the widgets.
- `mobileLayout`: Optional. It provides the layout for mobile devices, overriding the properties configured in the `widgetOnScreen`, `map`, and `widgetPool` sections.
A Typical Layout Structure

- **widgetOnScreen**: Includes layout properties for widgets placed on the UI, such as a controller, predefined widgets, and widget placeholders. And the panel to use to open the widgets.

- **map**: Defines the position of the map.

- **widgetPool**: Includes a list of widgets placed in the controller widget if there is a controller defined in the widgetOnScreen section. And the panel to use to open the widgets.

- **mobileLayout**: Optional. It provides the layout for mobile devices, overriding the properties configured in the widgetOnScreen, map, and widgetPool sections.
A Typical Layout Structure

- **widgetOnScreen**: Includes layout properties for widgets placed on the UI, such as a controller, predefined widgets, and widget placeholders. And the panel to use to open the widgets.
- **map**: Defines the position of the map.
- **widgetPool**: Includes a list of widgets placed in the controller widget if there is a controller defined in the widgetOnScreen section.
- **mobileLayout**: Optional. It provides the layout for mobile devices, overriding the properties configured in the widgetOnScreen, map, and widgetPool sections.
What Composes a WAB Theme?

- Layout
- Panel
- Style
- Controller
Existing Panels

Foldable Panel:
Can be collapsed and closed

Launchpad Panel:
Can be dragged minimized, and
closed

Dart Panel:
Can be dragged, collapsed, and
closed
What Composes a WAB Theme?

- Layout
- Panel
- Style
- Controller
What Makes Up The WAB UI?

WAB UI

Jimu widgets

Dojo dijits

ArcGIS API widgets
How UI Libraries Work In WAB:

- **Theme:** common.css, style.css
- **Jimu:** jimu.css, jimu-override.css, etc.
- **ArcGIS API for JavaScript:** esri.css
- **Dojo dijets:** claro.css
What Composes a WAB Theme?

- Layout
- Panel
- Style
- Controller
## Compare to a Functional Widget

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Controller Widget</th>
<th>Functional Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displays app information, defines app behaviors, interactions, workflows, etc.</td>
<td>Provides one specific functionality to the app</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Folder Structure</th>
<th>Controller Widget</th>
<th>Functional Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Very similar</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manifest</th>
<th>Controller Widget</th>
<th>Functional Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>isController</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>isThemeWidget</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>inPanel</td>
<td>Always false</td>
<td>May vary</td>
</tr>
<tr>
<td>Others</td>
<td></td>
<td>Very similar</td>
</tr>
</tbody>
</table>
Let’s Create A New Theme
Before we start...
Required Technical Skills

HTML
CSS
JS

dojo toolkit
Different Levels of UI Customization

- **No designer or developer needed;**
  - May not need a custom theme
  - Less than 8 hours

- **Developer needed;**
  - At least 16 hours

- **Designer and developer needed;**
  - At least 40 hours

- **Designer and developer needed;**
  - At least 80 hours

- **logo and branding colors**

- **custom layout, widget or panel**

- **Have a specific UI guideline to follow;**
  - Only a portion of widgets need to be restyled

- **Have a specific UI guideline to follow;**
  - All widgets need to be restyled
Theme Mockup

![Map of California Farmers Markets with search feature for locations and distances to markets.](image-url)
Mockup UI Break Down
STEP 1: Create Theme Folder
Minimum Required Folder Structure

- **MyTheme**
  - images
    - icon.png
  - layouts
  - default
    - config.json
    - icon.png
  - nls
    - strings.js
  - styles
    - default
    - style.css
    - common.css
  - main.js
  - manifest.json
USE DEMOTHEME TO START

• DemoTheme contains all components needed to create a new theme
• Location: client\stemapp\themes\DemoTheme
"name": "SidebarTheme",
"label": "This is my theme, and it's awesome",
"platform": "HTML",

"layouts": [{...}],
"panels": [{...}],
"styles": [{...}],
"widgets": [{...}],

"version": "2.3",
"wabVersion": "2.3",
"author": "Esri Professional Services",
"description": "",
"copyright": "",
"license": "http://www.apache.org/licenses/LICENSE-2.0"
Step 2: Update Layout
Remove Unused Widgets

For example, the following widgets will not be used in Sidebar Theme, remove them from "widgets" under "widgetOnScreen":

- "widgets/Scalebar/Widget"
- "widgets/HomeButton/Widget"
- "widgets/Coordinate/Widget"
- "widgets/BasemapGallery/Widget"

Read more on widgetOnScreen
Add More Widget(s)

- Add the Search widget to the `widgetOnScreen`:

```json
"widgets": [
    ...
    {
        "uri": "widgets/Search/Widget",
        "position": {
            "top": 24,
            "right": 24
        }
    }
]
```
Reposition Widget (Controller)

```json
"widgets": [
  {
    "position": {
      "left": 0,
      "top": 0,
      "bottom": 0,
      "width": 100,
      "relativeTo": "browser"
    }
  }
]
```
Reposition Panel (in widgetPool)

```json
"widgetPool": {
  "panel": {
    "uri": "themes/SidebarTheme/panels/SimpleBorderPanel/Panel",
    "position": {
      "top": 0,
      "left": 100,
      "bottom": 0,
      "width": 410,
      "relativeTo": "browser",
      "zIndex": 101
    }
  }
}
```
Reposition Map

```
"map": {
    "position": {
        "left": 510,
        "top": 0,
        "right": 0,
        "bottom": 0
    }
}
```
Step 3: Controller Widget
Controller Widget

• This Sidebar Controller widget will be able to:
  - Read app config and display app logo, and user information
  - Read widget configs and display their icons
  - Open the first widget defined in the “widgetPool” or the widget with “openAtStart” turned on
  - Switch between different widgets and open one at a time in the panel when a widget icon is clicked
The development process is very similar to develop a functional widget
Controller Widget

- Create a folder call `widgets`
- Copy the `CustomWidgetTemplate` folder from `~/client/stemapp/widgets/samplewidgets`
- Rename it to `SidebarController`
- Open `manifest.json`:
  - make sure these properties are set as follows:
    - "inPanel": false,
    - "isController": true,
    - "isThemeWidget": true
  
  So it does not need a panel to display
  It is a controller instead of a default WAB widget
  It is a theme specific widget

Read more...
“isController” and “isThemeWidget” Properties

• **isController**: tells Web AppBuilder’s backend (Jimu framework) that this is a controller, so it will be dealt differently from other on-screen widgets*.

• **isThemeWidget**: is solely a property to mark the controller as a widget in a theme.

* In layout config.json, controller widget is added as an on-screen widget
Add HTML Template

- Add the following code to Widget.html:

```html
<nav>
  test
  <div class="logo">
    <img data-dojo-attach-point="logoNode">
  </div>
  <div data-dojo-attach-point="containerNode">
  </div>
</nav>
```

Read more on creating widgets with Dojo Toolkit
Add Controller widget to the layout

- Open the layout `config.json` and add the controller to the placeholder defined in the “Layout” session:

```json
"widgets": [{
    "uri": "themes/SidebarTheme/widgets/SidebarController/Widget",
    "position": {
        "left": 0,
        "top": 0,
        "bottom": 0,
        "width": 100,
        "relativeTo": "browser"
    }
}]
```
Add App Logo

• Use “this.appConfig.logo” to get the url of the app logo image and populate it on the UI:

```javascript
// add logo
if (this.appConfig) {
  if ('logo' in this.appConfig) {
    this.logoNode.src = this.appConfig.logo;
    this.logoNode.alt = this.appConfig.logo;
  } else {
    this.logoNode.style.display = 'none';
  }
}
```
Add User Thumbnail

- Use “jimu/portalUtils” class to read a user’s information from the Portal:

```javascript
// add user thumbnail image
var portal = portalUtils.getPortal(appConfig.portalUrl),
    _this = this;
portal.getUser().then(function(user){
  this.userInfo = user;
  console.log(user);
  var token = portal.credential.token,
    thumbnailContainer = document.createElement('A'),
    thumbnailImg = document.createElement('IMG');
  thumbnailContainer.className = 'icon-node user';
  if(user.thumbnail) {
    var imgUrl = _this.appConfig.portalUrl+ 'sharing/rest/community/users/' +
                  user.username + '/info/' + user.thumbnail + '?token=' + token;
    thumbnailImg.src = imgUrl;
    thumbnailContainer.className += ' has-thumbnail-img';
  } else {
    thumbnailImg.src = _this.folderUrl + 'user.png';
  }
  thumbnailContainer.appendChild(thumbnailImg);
  _this.containerNode.appendChild(thumbnailContainer);
});
```
Read And Add Widget Icons

- Use "this.getAllConfigs();" to get the configs for all widgets defined in the “widgetPool”, and then loop through to add their icons to the UI:

```javascript
// add widget icons
if(allConfigs && allConfigs.length) {
    var hasOpenAtStartWidget = false,
        firstIconNode;
    for (var i = 0; i < allConfigs.length; i++) {
        var iconName = this._createIconNode(allConfigs[i]);
        if(i === 0) firstIconNode = iconName;
        // check if the widget is set to open at start
        // (only open the first widget that is set to open at start)
        if(allConfigs[i].openAtStart && !hasOpenAtStartWidget) {
            this._showWidgetContent(allConfigs[i]);
            this._activateIconNode(iconName);
            hasOpenAtStartWidget = true;
            activeIconNode = iconName;
        }
    }
    // open the first widget if no widget is set to open
    // at start
    if(!hasOpenAtStartWidget) {
        this._showWidgetContent(allConfigs[0]);
        this._activateIconNode(firstIconNode);
    }
}
Read more on widgetPool
```
Step 4: Panel Widget
Panel Widget

• This Panel widget will be able to:
  - Show the name of the opened widget in the header section
  - Display the dom node of the widget in its content section

• We will modify on top of the SimpleBorderPanel
Panel Widget

- Rename the folder name of `SimpleBorderPanel` to something else. e.g.: `SimpleHeaderPanel`
- Update panel information in `manifest.json`:

```json
"panels": [
    {
        "name": "SimpleHeaderPanel",
        "description": "This is a panel with a header"
    }
]
```

- Update the panel path in `layouts/default/config.json`:

```json
"widgetPool": {
    "panel": {
        "uri": "themes/SidebarTheme/panels/SimpleHeaderPanel/Panel",
        "position": {
            ...
        }
    }
}
```
Add HTML Template

- Add an HTML template named Panel.html in panels/SimpleHeaderPanel, and add two html elements for header and content:

```html
<div>
  <header class="jimu-panel-title">
    <div class="title-label" data-dojo-attach-point="titleLabelNode">
    </div>
  </header>
  <div class="jimu-panel-content" data-dojo-attach-point="containerNode"></div>
</div>
```
Work On Panel.js

• Since we added an HTML template, the following dependencies need to be added in the "define" area in Panel.js:

```javascript
'dijit/_TemplatedMixin',
'dojo/text!./Panel.html'
```

• Use `this.config` to read current widget’s configuration settings to add the header:

```javascript
var appConfig = this.config;
if(!appConfig) return;

// add label
if(appConfig.label) {
    this.titleLabelNode.textContent = appConfig.label;
} else {
    this.titleLabelNode.textContent = "";
}
```
Work On Panel.js

- No need to add code to read and add a widget to the `containerNode`. This is all taken care of by JIMU’s `BaseWidgetPanel` class.
BaseWidgetPanel Class

- All panels are inherited from Jimu’s BaseWidgetPanel class
- It provides a list of getters / setters, such as:
  - Getters:
    - getWidgetById()
    - getWidgets()
    - getAllWidgetConfigs()
    - getPosition()
  - Setters:
    - setPosition()
BaseWidgetPanel Class

- And the following events:
  - onOpen
  - onClose
  - onMaximize
  - onMinimize
  - onNormalize
  - onActive
  - onDeActive

These are evoked by PanelManager.js
Step 5: Styles (CSS)
common.css

- Effective on ALL styles within the same theme
- Use common.css to apply the basic style rules such as:
  - Typography
  - Icons
  - Normalization
  - Spacings
  - Overrides of Dojo dijits and ArcGIS API widgets
styleFolder/style.css

- Effective ONLY when it is selected
- Use style.css to update color themes
Add widget overrides in common.css

- Add CSS overrides for the following to restyle them:
  - Zoom Slider
  - Search
  - Near Me
Add a New Style

• Add a new style to the theme to apply a different color scheme to the app.

```css
.jimu-main-background {
  background-color: #042833;
}

.jimu-panel-title {
  background-color: #06b0d8;
  background-image: linear-gradient(to right, #06b0d8, #37d4e6);
}

.jimu-widget-sidebar-controller .icon-node.jimu-state-active:after {
  background-color: #37d4e6;
}
```
Recap
Useful Resources

WAB Theme Concepts
https://developers.arcgis.com/web-appbuilder/guide/concepts.htm

Detailed online tutorial on creating a WAB theme:
https://developers.arcgis.com/web-appbuilder/sample-code/create-a-new-theme.htm

Source Code of the SidebarTheme:
https://goo.gl/6UdQ8R

Web AppBuilder on GeoNet:
https://geonet.esri.com/community/gis/web-gis/web-appbuilder
Q&A