ArcGIS API for JavaScript: Customizing Widgets

Alan Sangma – @alansangma
Matt Driscoll – @driskull
JC Franco – @arfncode
Agenda

- What can be customized
- Customization approaches with demos
- Q & A
Customizing Widgets

- Theming
  - Changing styles: colors, sizing, font, etc.
- Implementing widget in a different framework
- Altering presentation of a widget
Customization Approaches

- Authoring a theme
- Recreating a view
- Extending a view
Level I

Theming
Level I: Theming

Why Theme?

- Match branding.
- Match the map.
- Contrast with the map.
- Based on the environment.
- User-specific (e.g. bigger buttons)
Theming Technology
We use Sass to create our CSS.
is a powerful scripting language for compiling CSS.

- It's modular.
- It's DRY.
- It makes theming easy.
PREVIOUSLY ON THEMING
LIFE WAS HARD.
Before, you needed to

1. Pull down the API (arcgis-js-api).
2. Create a theme directory in the right place.
3. Create a Sass file.
4. Import the core file.
5. Run the compiler.
6. Wonder if there were an easier way.
There is an easier way!

1. Get our theme utility.
2. Use the utility.
3. Customize your theme.
4. Host your CSS file.
There is an easier way!

1. Clone the utility jsapi-styles.git
2. Run npm install
3. Edit `sass/my-theme/main.scss`
4. See `dist/my-theme/main.css`

You won't need the base stylesheet.
Step 1

Clone the repo.
https://github.com/jcfranco/jsapi-styles

git clone https://github.com/jcfranco/interactive-design.git
Step 2

```
npm install
```

- Installs the necessary bits.
- Creates a sample theme directory.
- Compiles the CSS from the SCSS.
- Spins up a preview in your default browser.
Step 3

Edit your theme.
sass/my-theme/main.scss

Optionally, edit your app.
preview/index.html
Step 4

Host your stylesheet and any relevant assets.

Link your stylesheet in your app.

<!-- In your app: -->
<link href="path/to/your/theme/main.css" rel="stylesheet">
Let's have a look!
Theme Smart

Avoid adding additional CSS selectors. Instead, use Sass to your advantage. Let's look at how the core theme is structured.
Theme Structure

- Color: colorVariables.scss
- Size: sizes.scss
- Type: type.scss
Theme Structure

Default

// Inside base/_colorVariables.scss
$background_color: #fff !default;

Any value assignment overrides the !default value.

// Inside sass/my-theme/main.scss
$background_color: #cc4b09;

But wait...there's more!
Theme Structure

Override the four main color variables...

```
$text_color : #fff;  // white
$background_color : #cc4b09;  // mario
$anchor_color : #ffbaaa;  // luigi
$button_text_color : #ffbaaa;  // luigi
```

...then magic!
Magic

$button_text_hover_color: offset-foreground-color($button_text_color, 25%) !default;
$anchor_hover_color: offset-foreground-color($anchor_color, 25%) !default;
$background_hover_color: offset-background-color($background_color, 5%) !default;
// etc.

Theming Guide
So let's make a theme!
Level I: Theming Recap

- Use the utility for easy theming.
- Theme structure
  - Color
  - Size
  - Typography
- Use the core and override values.
LEVEL UP!

Ready?
LEVEL II

Views
Level II: Widget Composition

Widgets are composed of Views & ViewModels

- Reusable
- UI replacement
- Framework integration
Level II: Views

- Presentation of the Widget
- Uses ViewModel APIs to render the UI
- View-specific logic resides here
Level II: Working with Views

API Exploration

- Attribution Doc
- Attribution Sample
// AttributionViewModel in 4.7 Release

interface AttributionViewModel {
    items: Collection<AttributionItem>;
    state: "ready" | "disabled";
    view: MapView | SceneView;
}

interface AttributionItem {
    layer: Layer;
    text: string;
}
Views: Let's customize!

Let's create a custom widget view.
Demo

Create a custom Attribution Table view

- Demo Steps
- Demo Complete
- Demo Start
Level II: Views Recap

What have we learned about Widget Views?

- Face of the widget
- Present ViewModel logic
- ViewModel separation allows framework integration or custom views
- Downloadable on API docs
LEVEL UP!

Ready?
LEVEL III

Extending a View
Level III: Extending a View

- Why?
  - Reusable
  - Same ecosystem
- How?
  - JS API v4.7
  - esri/widgets/Widget
  - TypeScript
esri/widgets/Widget

- Provides lifecycle
- API consistency
Lifecycle

- constructor
- postInitialize
- render
- destroy
render

- Defines UI
- Reacts to state
- Uses JSX
- VDOM
TypeScript

- Typed JavaScript
- JS of the future, now
- IDE support
  - Visual Studio
  - WebStorm
  - Sublime
  - and more!
Demo: Extending a View

Demo | Steps
Level III: Extending a View Recap

- Reusable
  - View/ViewModel
- Same ecosystem
  - No extra libraries
- Extended existing widget
  - Lifecycle
  - TypeScript
LEVEL UP!

Ready?
LEVEL IV

Put it all together.
Conclusion

- Authored a theme
- Recreated a view
- Extended a view
Additional Resources

- Implementing Accessor
- Setting up TypeScript
- Widget Development
- JS API SDK
Please take our survey!

1. Load the Esri Events app and find your event.
2. Select the session you attended.
3. Scroll down to the “Feedback” section.
4. Complete answers, add a comment, and select “Submit.”
Questions?
For example

🤔 Where can I find the slides/source?

👉 esriurl.com/customwidgetsds2018 ➤