Building Your own Widget with ArcGIS API for JavaScript

Matt Driscoll – @driskull

JC Franco – @arfncode
Agenda

- About Widgets
- Prerequisites
- Widget framework
- Theming
- DO IT!
- Tips & tricks
About Widgets

- **What?**
  - Encapsulated
  - Cohesive
  - Single-purpose pieces of functionality

- **Why?**
  - Reusable
  - Interchangeable

- **How?**
  - Different frameworks are available
Prerequisites

- Accessor (*esri/core/Accessor*)
- TypeScript
Accessor

- JavaScript API Foundation
- Consistent developer experience
- TypeScript support
var view = new MapView({
    container: "viewDiv",
    map: map
});

var symbol = new SimpleMarkerSymbol({
    style: "square",
    color: "blue"
});

var widget = new BasemapToggle({
    view: view,
    nextBasemap: "hybrid"
});
Accessor - Defining Properties (getters + setters)

```javascript
var Foo = Accessor.createSubclass({
  properties: {

    // read-only
    foo: { readOnly: true, value: new Foo() },

    // aliased
    bar: { aliasOf: "foo" },

    // autocast
    baz: { type: SomeClass }
  }
});
```
Accessor - Property watching

```javascript
// watch for changes using a property
view.watch("map.basemap.title", handleTitleChange);

// watch for changes to multiple properties
view.watch("stationary, interacting", handlePropChange);
```
TypeScript

- Superset of JavaScript
- Compiled to JavaScript
- Statically type-checked
- Syntactic sugar... sweet!
  - Use ES6 syntax while targeting ES5 environments
Type safety

```typescript
let view: MapView | SceneView;

// ...

/*
 * TS2322: Type '"not-a-view"' is not
to type 'MapView | SceneView'.
*/
view = "not-a-view";
```
Typings

Help describe what things are:

```javascript
type PresenterName = "Alan" | "Matt";

interface Person {
    name: string;
    age: number;
}

interface Presenter extends Person {
    name: PresenterName;
}
```
JS of the future, now

Fat arrow functions

```javascript
const someFn = () => { /* ... */ };

// instead of
const someFn = function () { /* ... */ }
```
JS of the future, now

Template strings

```javascript
const text = `Hello. Nice to meet you,

// instead of

const text = "Hello. Nice to meet you,
```
JS of the future, now

Destructuring

```javascript
const { map, zoom, scale } = view;

// instead of
const map = view.map;
const zoom = view.zoom;
const scale = view.scale;
```
JS of the future, now

Rest Parameters

```javascript
function ignoreFirst(first, ...theRest)
    console.log(theRest);
}

// instead of

function ignoreFirst() {
    var theRest = Array.prototype.slice.call(console.log(theRest);
    console.log(theRest);
}
```
JS of the future, now

Decorators

```javascript
@log()
foo = "foo";
```
TypeScript IDE Support

- WebStorm
- Sublime Text
- Atom
- Eclipse
- Brackets
- Emacs
- Vim
TypeScript + JS API 4

- Install TypeScript
- Install JavaScript API typings
- Start writing code!

TypeScript setup
Let's see some widget decorators
Creating a class

@subclass + declared

```java
@subclass("example.Foo")
class Foo extends declared(Accessor) {
    // ...
}
```
Creating a class: multiple inheritance pattern

@subclass + declared

```java
interface Foo extends Bar, Baz {}

@subclass("example.Foo")
class Foo extends declared(Accessor, Bar, Baz) {
    // ...
}
```
Defining a property

@property

@property()
foo = new Foo();
Custom setter

@property

@property()

set myProperty(value: string) {
    // note internal `_set`
    this._set("myProperty", value);
    this._ensureValidity(value);
}
Computed properties

@property

```javascript
@property({
    dependsOn: ["firstName, lastName"]
})
get fullName() {
    return `${this.firstName} ${this.lastName}`
}
```
Read-only value

@property

```javascript
@property({
    readOnly: true
})
myProperty = "I'm read-only";
```
Autocast

@property

```javascript
@property({
  type: MyClass
})
myProperty;
```

```javascript
instance.myProperty = { /* params */ }
console.log(instance.myProperty instance
```
Alias a property

@property

@property({ aliasOf: "bar.baz" })
foo;
Alias a property

@aliasOf

@aliasOf("bar.baz")
foo;
Handle click and key events

@accessibleHandler

```javascript
@accessibleHandler
private function _doSomething() {
   // ...
}
```
Rendering when properties change

```javascript
@renderable

@property()
@renderable()
title = "hello";

@property()
@renderable()
viewModel = new ViewModel();
```
More details in the SDK

- **Implementing Accessor**
- **Widget Development**
Widget framework

- JSX
- Lifecycle
- Properties
- Methods
- Events
Widget Framework: About

esri/widgets/Widget: Our new widget framework

- Accessor-based
- Built with TypeScript
Widget Framework: **JSX**

- JavaScript extension **syntax**
- adds XML syntax to JavaScript
- Looks similar to HTML
- Can use JS inline!

```jsx
<div class={classLookup.hello}
     onclick={this._handleClick}
     tabIndex={0}>
   Hello World
</div>
```
Widget Framework: Lifecycle

- constructor()
- postInitialize()
- render()
- destroy()
constructor()

```typescript
constructor(params?: any) {
    super();
    // Do some stuff!
}
```
postInitialize()
render()

- Return JSX
- Virtual DOM

```javascript
render() {
    return (
        <button>{this.title}</button>
    );
}
```

**Widget rendering (SDK)**
destroy()

destroy() {
    // cleanup listeners
    // destroy other widgets
    // dereference variables
    // etc.
}

Framework: Getting/Setting Properties

// normal setting of a prop
myWidget.property = value;

// normal getting of a prop
console.log(myWidget.property);

// internal set property
// will not trigger setter
this._set("property", propertyValue);

// internal get property
// will not trigger getter
this._get("property");
Framework: ViewModels

(The brain)

- Core logic of widget resides here
- Provides necessary APIs for the view to do its thing
- No DOM/UI concerns (think business logic)
ViewModels: Why?

- Framework integration
- Reusability
- Separates concerns
Widget Framework: Views

(The face)

- `esri/widgets/Widget`
- Uses ViewModel APIs to render the UI
- View-specific logic resides here
Views: Why?

- Separates concerns
- Framework compatibility
Views: Defining ViewModel

```javascript
@property({
    type: MyViewModel
})
viewModel: MyViewModel = new MyViewModel
```
Widget Framework: Method Convention

Public Methods

```java
public myMethod() {}
```

Private Methods

```java
private _myMethod() {}
```
Widget Framework: Events

- Views have ability to `emit()` an event.
- ViewModel needs to import `dojo/evented` in order to `emit()`
- Views can alias an event with `vmEvent` decorator.
Widget Framework: View Events

- Widget views extend `dojo/Evented`

```javascript
this.emit("my-event", {});
```
Widget Framework: ViewModel

1. Import evented

```javascript
import Evented = require("dojo/Evented");
```

2. Extend Evented

```javascript
interface MyViewModel extends Evented {
...
}
```

3. Emit event when necessary

```javascript
this.emit("my-event", {...});
```
Widget Framework: Aliased View Events

```javascript
@vmEvent("my-event")
@property(
   
   type: MyViewModel

   )
viewModel: MyViewModel = new MyViewModel
```
Widget Theming

- Out of the box themes
- SDK: Styling topic
- **Sass**
- **BEM**
Widget Theming: Guide

- **SDK Guide: Styles**

Styling

Default stylesheets

The following section describes the default stylesheet options available for the ArcGIS API for JavaScript.

main.css

Includes styles for everything in the API. This will be the simplest, albeit bigger-sized, option.

```
<link rel="stylesheet" href="https://js.arcgis.com/4.3/esri/css/main.css">
```

view.css
Widget Theming: Out of the box

Themes Demo
Widget Theming: Sass

- CSS preprocessor
- Variables
- `@mixin` (group statements)
- `@include` - (use mixins)
- `@import` - (split up files)
- `@extend` - (inheritance)
- More power!
Sass makes it easier to...

- Restyle
- Theme
- Modular / DRY
- Be organized
- Write less code :)

SOUNDS GOOD TO ME!
Sass Install

- **Installing Sass**
- **Themes source on Github**
Widget Theming: BEM

- **BEM**: Block Element Modifier
- Methodology to create reusable components
- Uses delimiters to separate block, element, modifiers
- Provides semantics (albeit verbose)
- Keeps specificity low
- Scopes styles to blocks

```css
/* block */
.example-widget {} 

/* block__element */
.example-widget__input {}
.example-widget__submit {}

/* block--modifier */
.example-widget--loading {}

/* block__element--modifier */
.example-widget__submit--disabled {}
DO IT!

Build a widget!

Demo
Setup HTML & CSS with Sass

- HTML Setup Steps
- Sass Setup Steps
Tips & Tricks

- Collections
- Accessibility
- i18n
Rethinking Widget APIs

- Using `esri/core/Collection` instead of Arrays
- Using `esri/core/Accessor` instead of plain objects
- Adding properties to manage widget state
- Hiding nodes with Widget framework JSX instead of CSS
  - setting JSX node to null when necessary
- Private/protected variables
Accessibility

- tabindex
- **aria roles**
- keyboard events
i18n

Import language file

```python
import * as i18n from "dojo/i18n!./MyWidget/nls/MyWidget"
```

Setup root language file with languages

```javascript
// "/nls/MyWidget"
define({
  root: {
    helloWorld: "Hello World"
  },
  "es": 1
});
```

Setup other languages. (\es)

```javascript
// "/nls/es/MyWidget"
define({
  helloWorld: "Hola Mundo"
});
```
Suggested Sessions

- **Building Classes Using Accessor and the ArcGIS API for JavaScript**
- **Using TypeScript with ArcGIS API for Javascript**
- **Deep Dive on How ArcGIS API for JavaScript Widgets Were Built**
Additional Resources

- **Styling**
- **Implementing Accessor**
- **Setting up TypeScript**
- **Widget Development**
- **JavaScript Sessions at DevSummit**
- **Documentation - 4.3**
Use the source luke

esriurl.com/buildwidgets2017

```javascript
import { aliasOf, declared, property, subclass } from "esri/core/accessorSupport/decorators";

import {
  accessibleHandler, jxfactory, renderable
} from "esri/widgets/support/widget";

type PathType = "microphone" | "processing";

import YoEsriViewModel = require("./YoEsri/YoEsriViewModel");

import MapView = require("esri/views/MapView");
import SceneView = require("esri/views/SceneView");
import Widget = require("esri/widgets/Widget");

const CSS = {
  base: "demo-yo-esri esri-widget",
  icon: "demo-yo-esri__icon",
  listening: "demo-yo-esri--listening",
  disabled: "demo-yo-esri--disabled",
  processing: "demo-yo-esri--processing"
};

@subclass("esri.demos.YoEsri")
class YoEsri extends declared(Widget) {
```
Please Take Our Survey!

1. Download the Esri Events app and go to DevSummit
2. Select the session you attended
3. Scroll down to the "Feedback" section
4. Complete Answers, add a Comment, and Select "Submit"
Questions?
Thank you!