Esri GeoDev Webinar Series

Using TypeScript with the ArcGIS API for JavaScript

Esri GeoDev Webinar Series

Using TypeScript with the ArcGIS API for JavaScript

Rene Rubalcava
Noah Sager

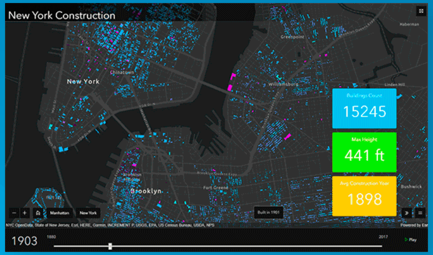
September 26, 2018

Agenda

- ArcGIS API for JavaScript 4.x
- TypeScript
- Converting a JavaScript app to TypeScript
- Development Resources
- Custom Widgets

ArcGIS API for JavaScript | Enabling Powerful and Modern Web GIS Apps

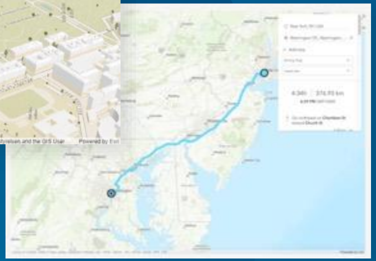
Data-Driven Visualization



Fast Interaction with Large Datasets

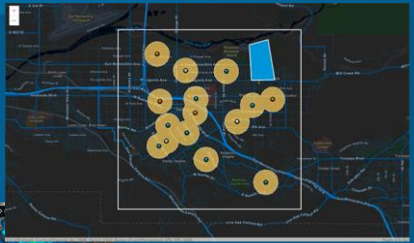
3D Measurement

Directions



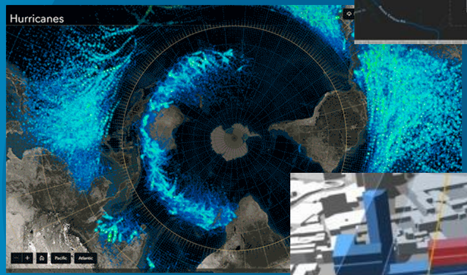
Widgets and Tools

Drawing Tools



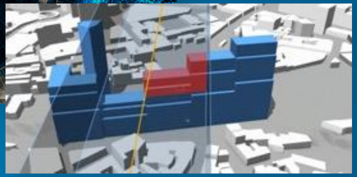
Client-Side Mapping and Processing

Client-Side Projection



Real-Time Geometric Analytics

Fast Display of Large Datasets



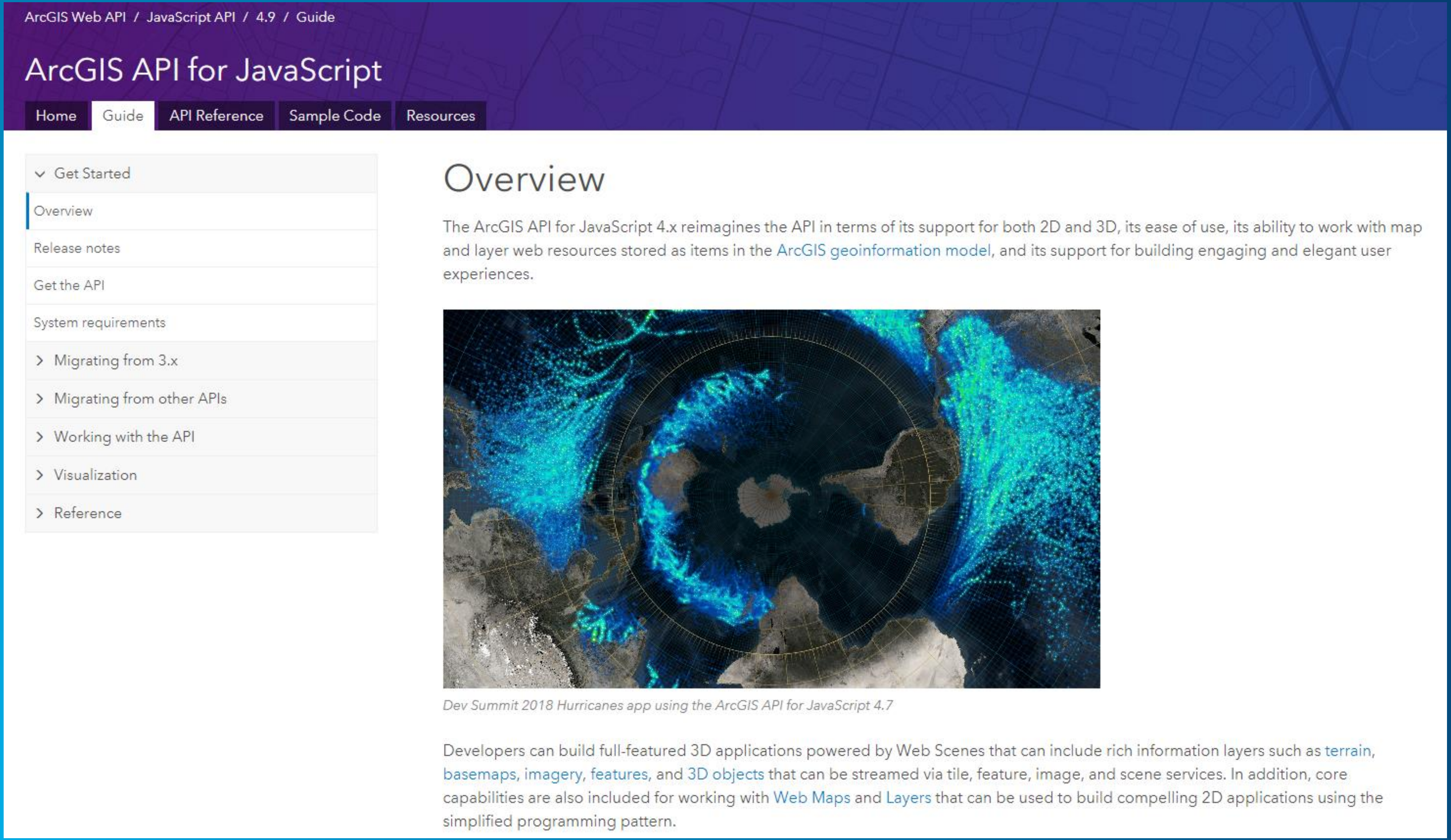
Interactive Analysis

3D Scenes

Smart Mapping

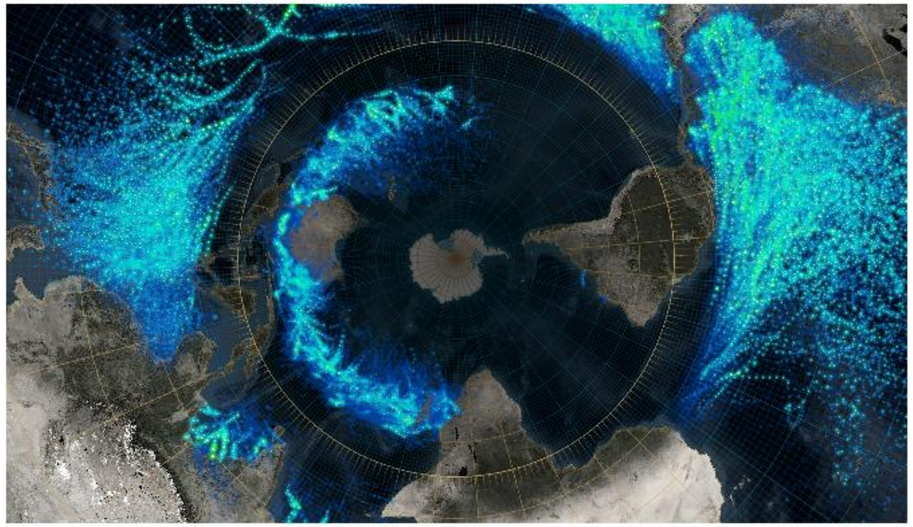
3D Mobile Web

ArcGIS API for JavaScript - 4.9



Overview

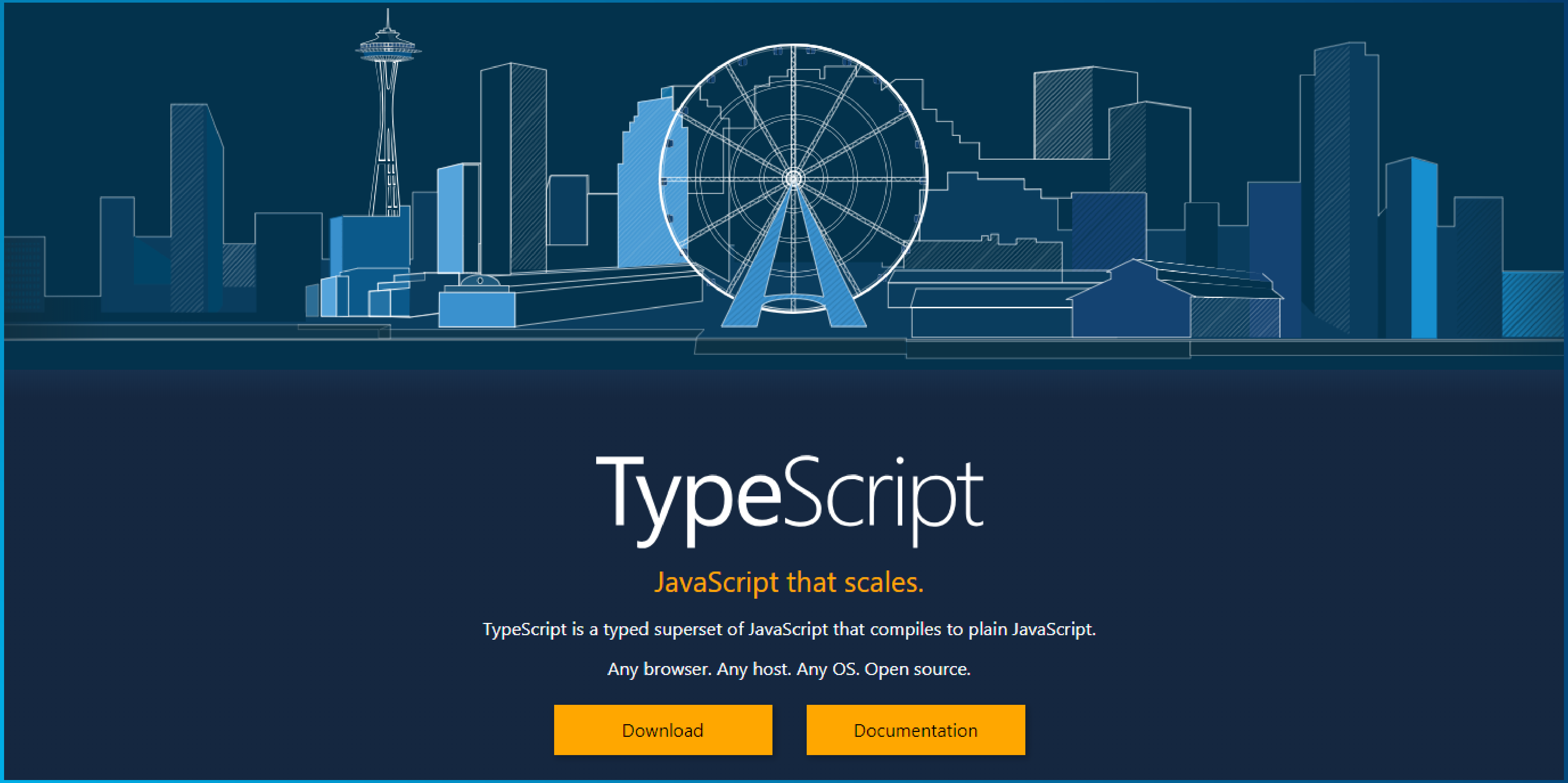
The ArcGIS API for JavaScript 4.x reimagines the API in terms of its support for both 2D and 3D, its ease of use, its ability to work with map and layer web resources stored as items in the ArcGIS geoinformation model, and its support for building engaging and elegant user experiences.



Dev Summit 2018 Hurricanes app using the ArcGIS API for JavaScript 4.7

Developers can build full-featured 3D applications powered by Web Scenes that can include rich information layers such as terrain, basemaps, imagery, features, and 3D objects that can be streamed via tile, feature, image, and scene services. In addition, core capabilities are also included for working with Web Maps and Layers that can be used to build compelling 2D applications using the simplified programming pattern.

What is TypeScript?



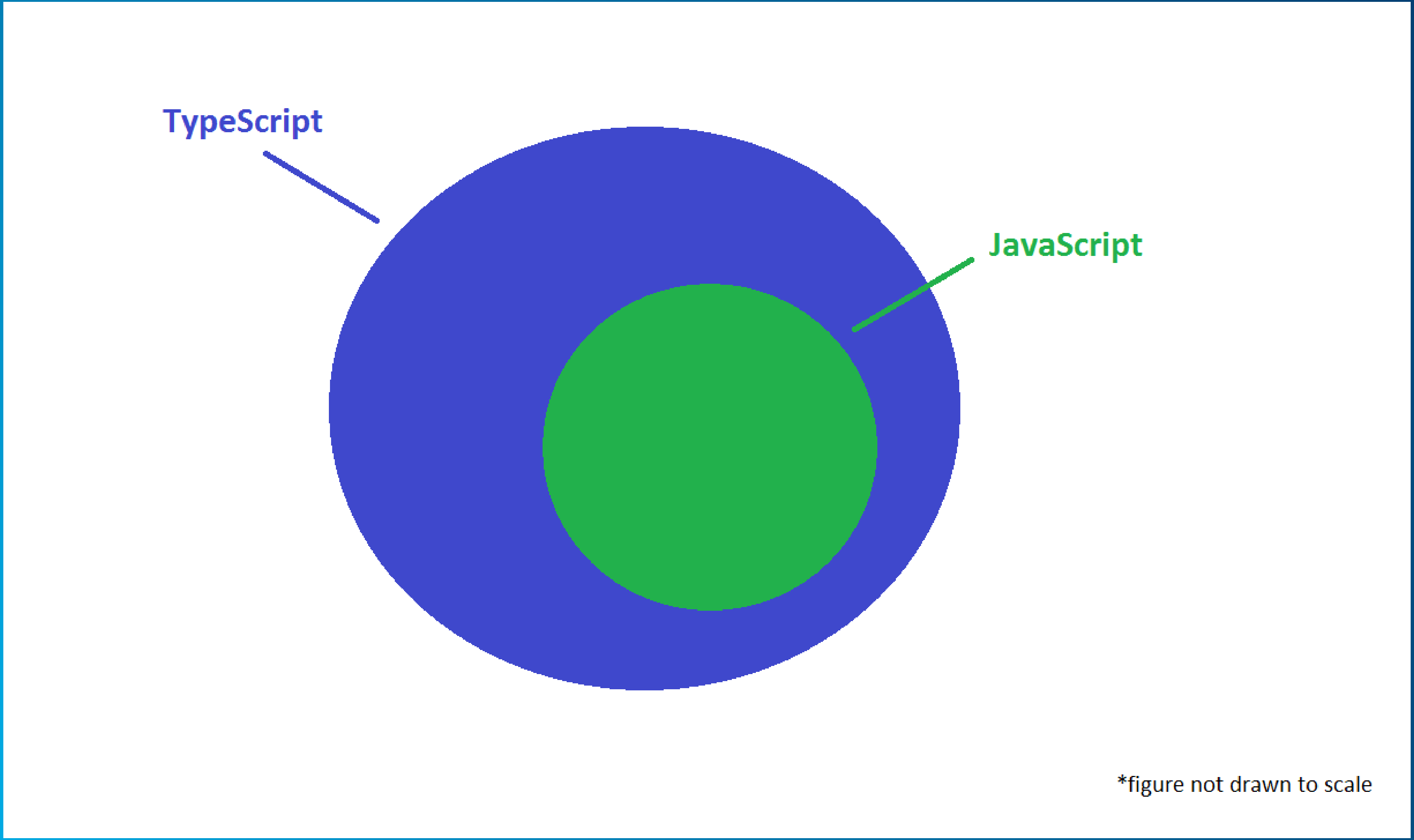
TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Any browser. Any host. Any OS. Open source.

TypeScript is a superset of JavaScript



Why TypeScript?

1. TypeScript adds type support to JavaScript

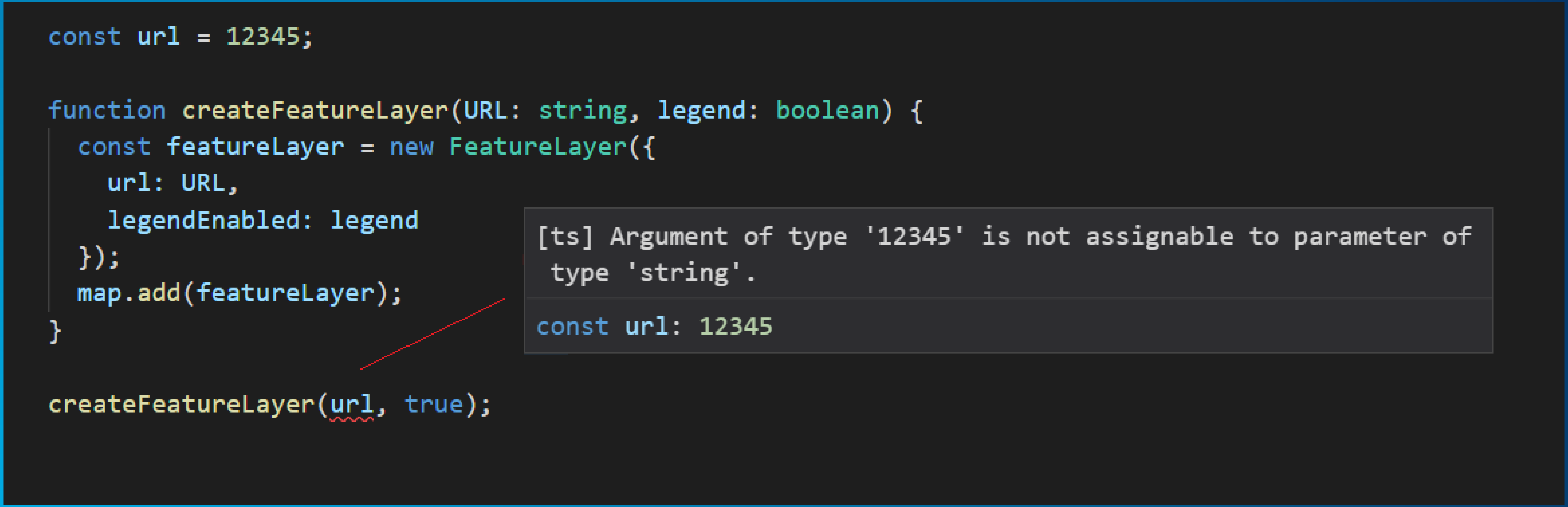
Why TypeScript?

```
const url = "https://sampleserver6.arcgisonline.com/arcgis/rest/services/Notes/FeatureServer/0";

function createFeatureLayer(URL: string, legend: boolean) {
  const featureLayer = new FeatureLayer({
    url: URL,
    legendEnabled: legend
  });
  map.add(featureLayer);
}

createFeatureLayer(url, true);
```

Why TypeScript?

1. TypeScript adds type support to JavaScript
2. Enhanced IDE support

Why TypeScript?



```
const url = 12345;

function createFeatureLayer(URL: string, legend: boolean) {
  const featureLayer = new FeatureLayer({
    url: URL,
    legendEnabled: legend
  });
  map.add(featureLayer);
}

createFeatureLayer(url, true);
```

```
[ts] Argument of type '12345' is not assignable to parameter of
 type 'string'.
const url: 12345
```

Why TypeScript?

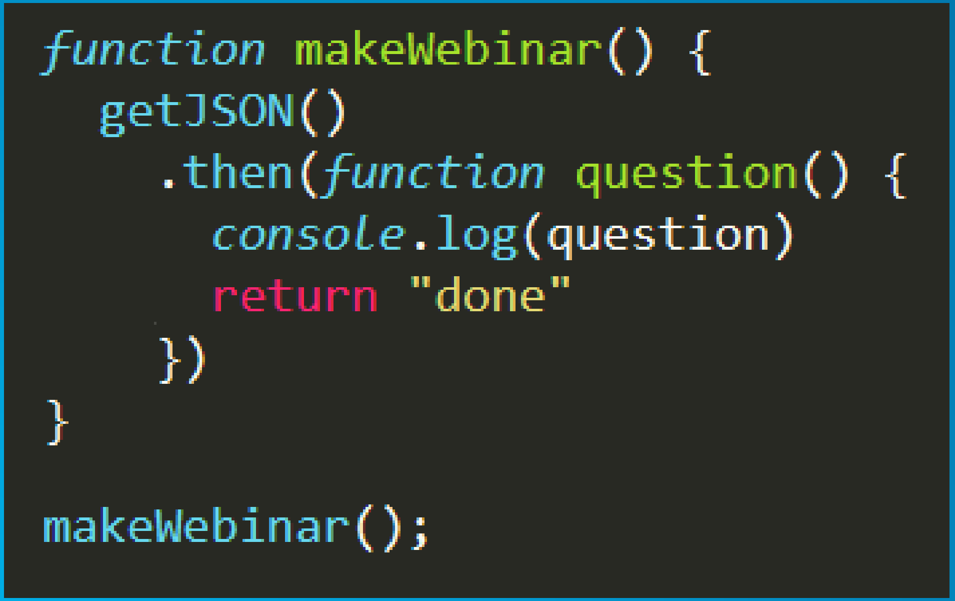
1. TypeScript adds type support to JavaScript

2. Enhanced IDE support

3. Makes use of the latest JavaScript features

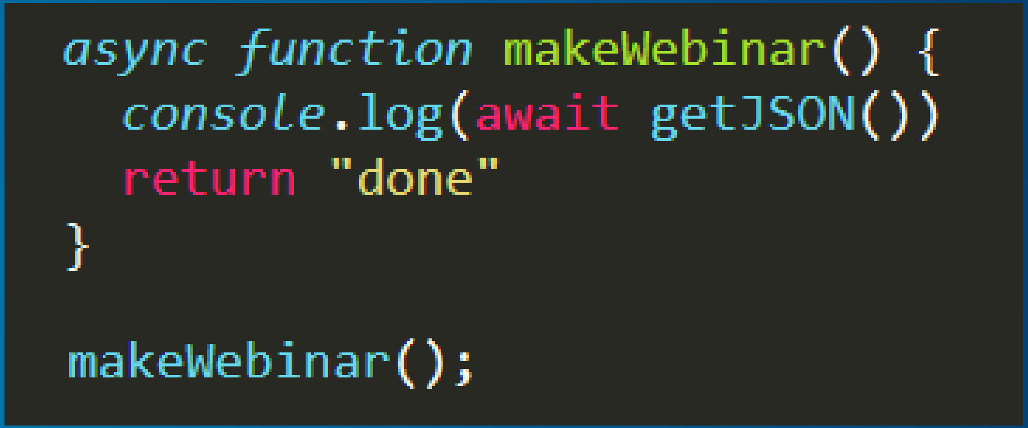
Why TypeScript? Latest JavaScript Features

promises



```
function makeWebinar() {
  getJSON()
    .then(function question() {
      console.log(question)
      return "done"
    })
}

makeWebinar();
```

async / await

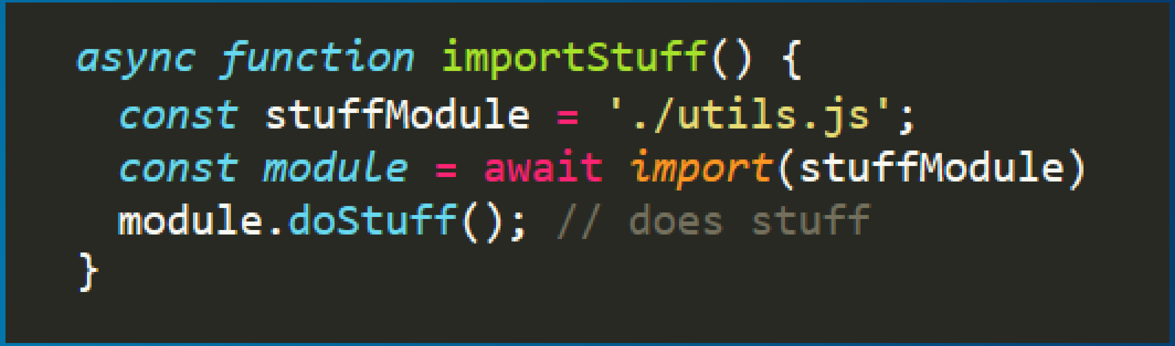


```
async function makeWebinar() {
  console.log(await getJSON())
  return "done"
}

makeWebinar();
```

Why TypeScript? Latest JavaScript Features

Dynamic imports

- compute the module at runtime
- import a module on-demand
  (or conditionally)
- import a module from within a regular script
  (as opposed to a module)



```
async function importStuff() {
  const stuffModule = './utils.js';
  const module = await import(stuffModule)
  module.doStuff(); // does stuff
}
```

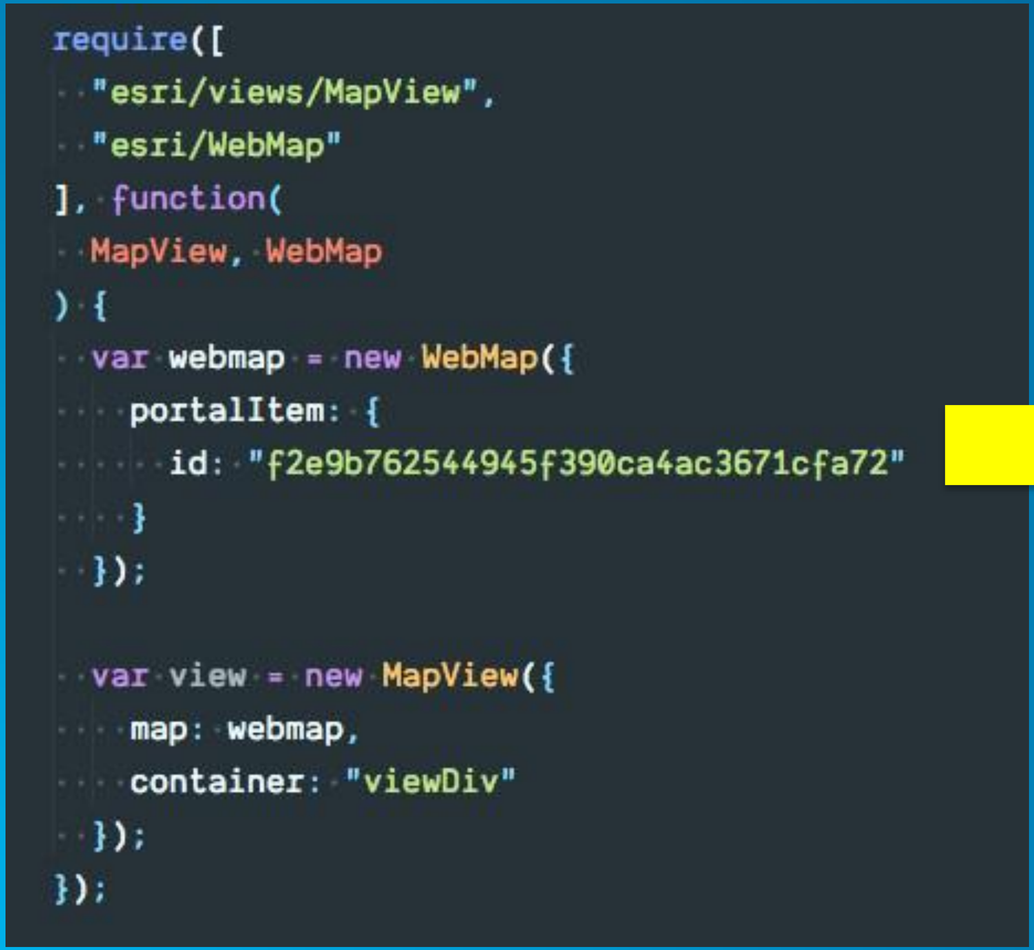
Convert JS App to TS

JavaScript to TypeScript

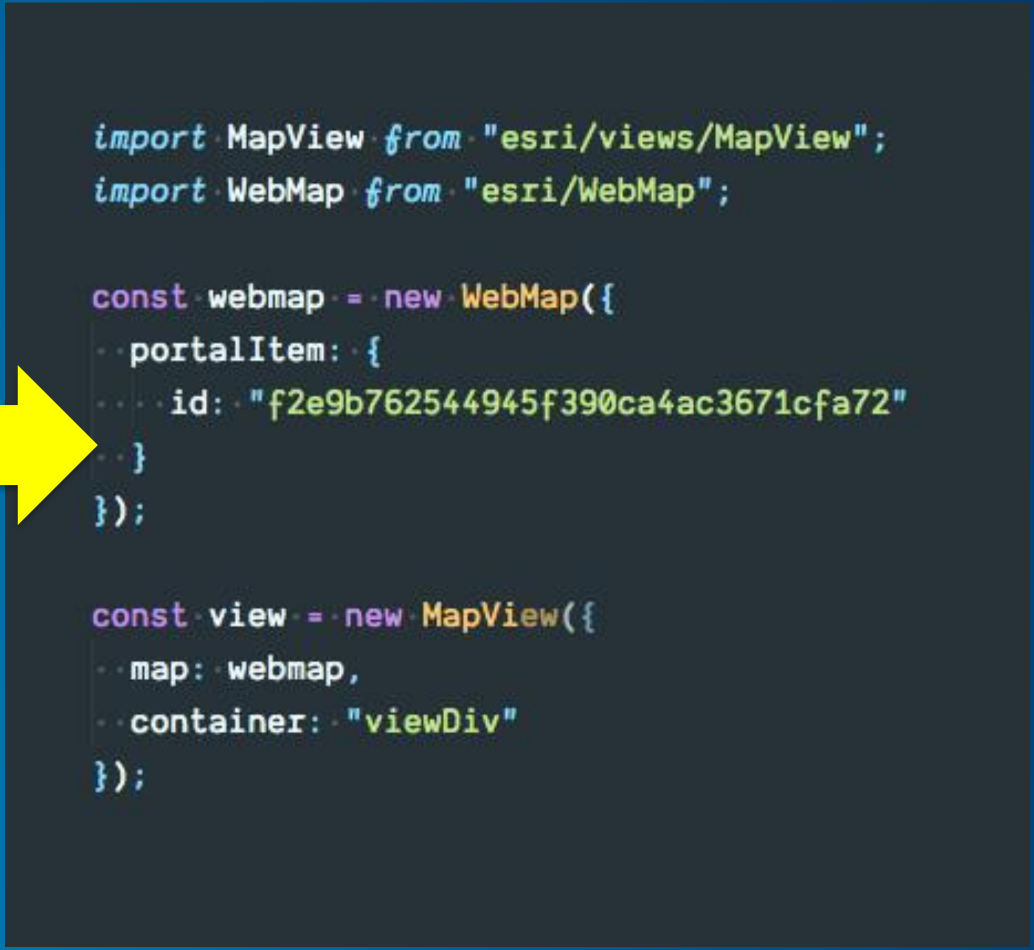
Since TypeScript is a *superset* of JavaScript …

Conversion can be done in steps

Convert JS App to TS



```
require([
  "esri/views/MapView",
  "esri/WebMap"
], function(
  MapView, WebMap
) {
  var webmap = new WebMap({
    portalItem: {
      id: "f2e9b762544945f390ca4ac3671cfa72"
    }
  });

  var view = new MapView({
    map: webmap,
    container: "viewDiv"
  });
});
```



```
import MapView from "esri/views/MapView";
import WebMap from "esri/WebMap";

const webmap = new WebMap({
  portalItem: {
    id: "f2e9b762544945f390ca4ac3671cfa72"
  }
});

const view = new MapView({
  map: webmap,
  container: "viewDiv"
});
```

Convert JS App to TS

Step 1

1. Do not need `require` statements.
2. Use `import` statements instead.

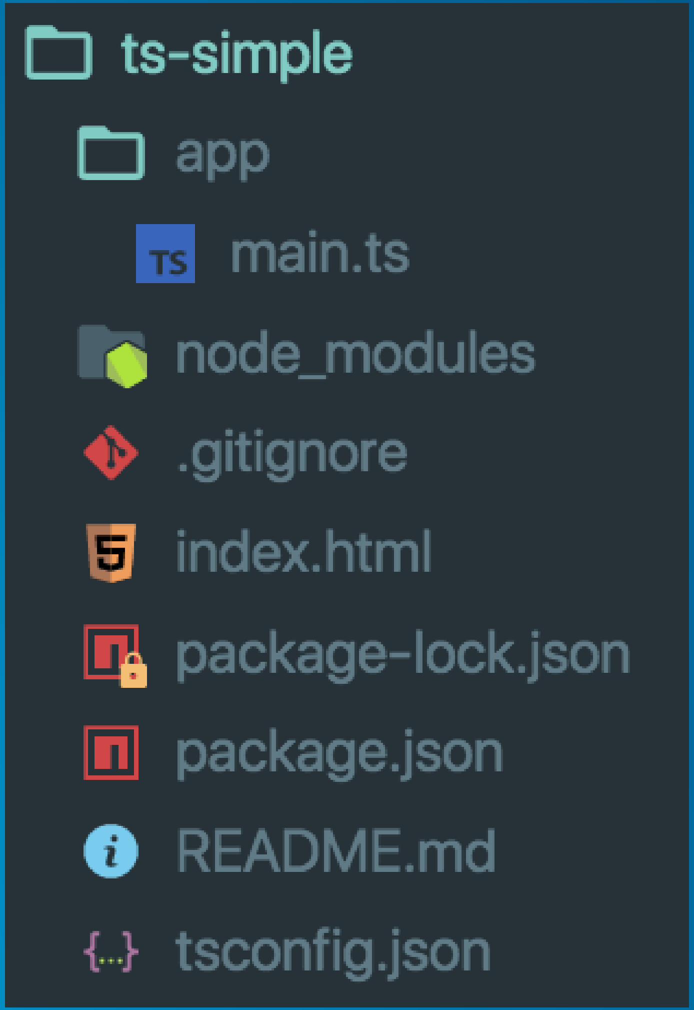
Convert JS App to TS

Step 2

1. Replace `var` with `const` or `let`.
2. Define Types and/or Interfaces

TypeScript

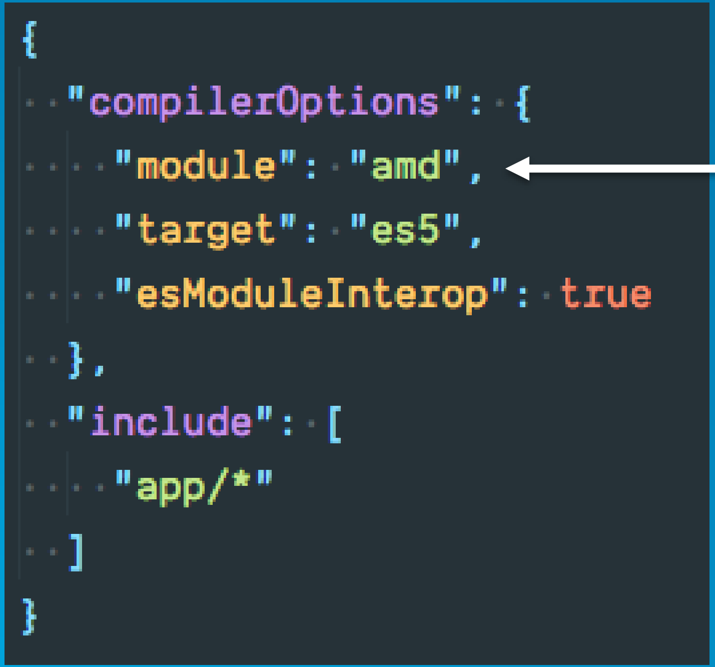
Basic Application Structure



```
ts-simple
    app
        main.ts
    node_modules
    .gitignore
    index.html
    package-lock.json
    package.json
    README.md
    tsconfig.json
```

TypeScript – tsconfig.json

Bare minimum configuration

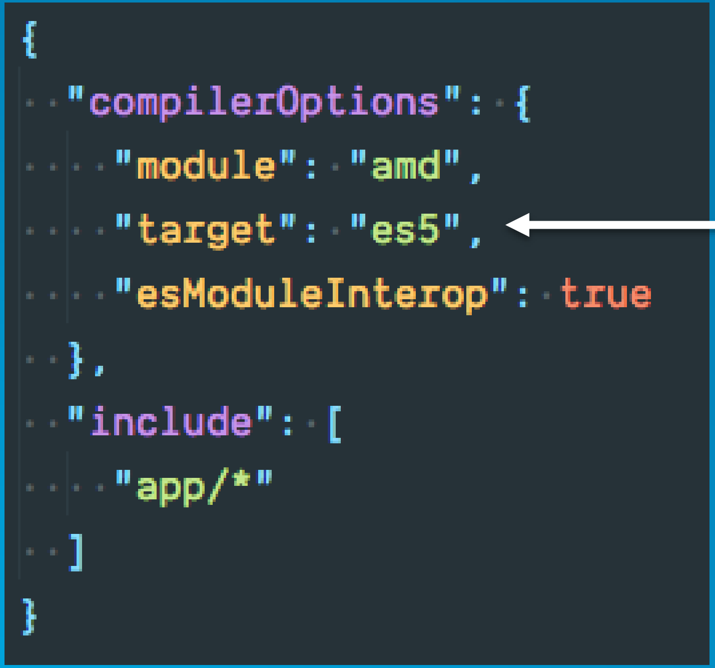


```json
{
  "compilerOptions": {
    "module": "amd",
    "target": "es5",
    "esModuleInterop": true
  },
  "include": [
    "app/*"
  ]
}
```

Output files as AMD modules

TypeScript – tsconfig.json

Bare minimum configuration

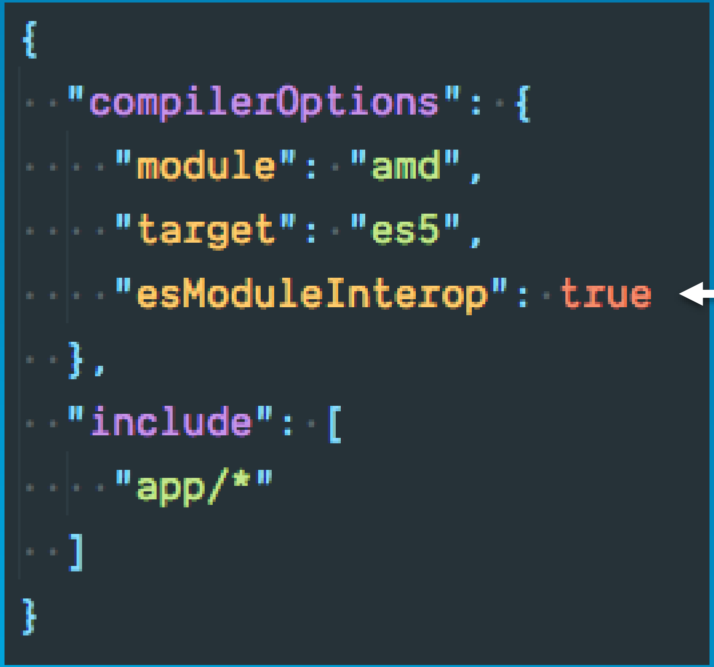


```json
{
  "compilerOptions": {
    "module": "amd",
    "target": "es5",
    "esModuleInterop": true
  },
  "include": [
    "app/*"
  ]
}
```

Output JavaScript as ES5

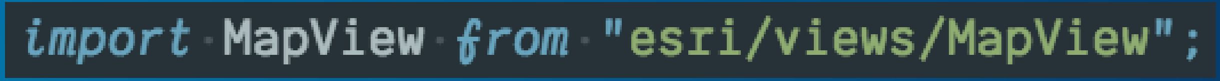
TypeScript – tsconfig.json

Bare minimum configuration



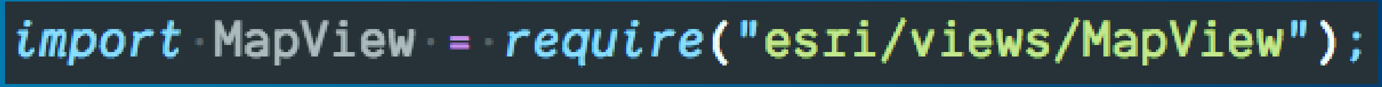
```json
{
  "compilerOptions": {
    "module": "amd",
    "target": "es5",
    "esModuleInterop": true
  },
  "include": [
    "app/*"
  ]
}
```

Use



```ts
import MapView from "esri/views/MapView";
```

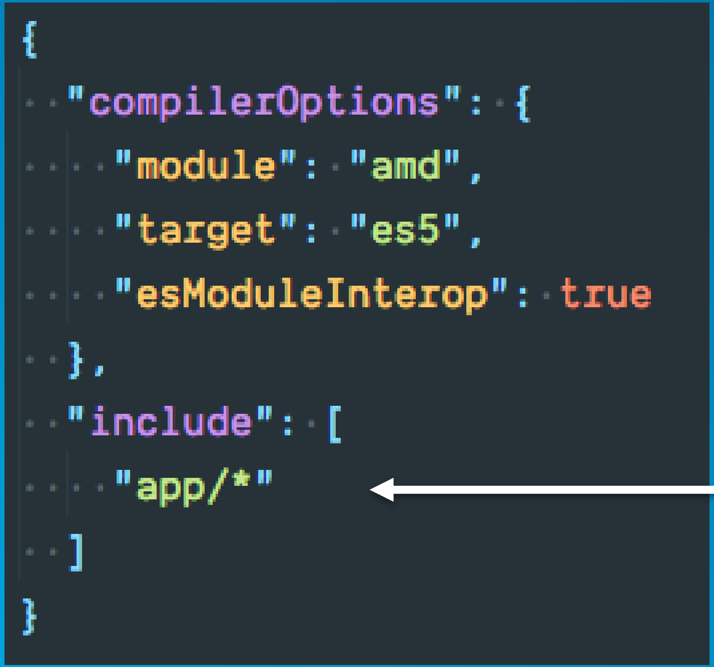
Instead of



```ts
import MapView = require("esri/views/MapView");
```

TypeScript – tsconfig.json

Bare minimum configuration

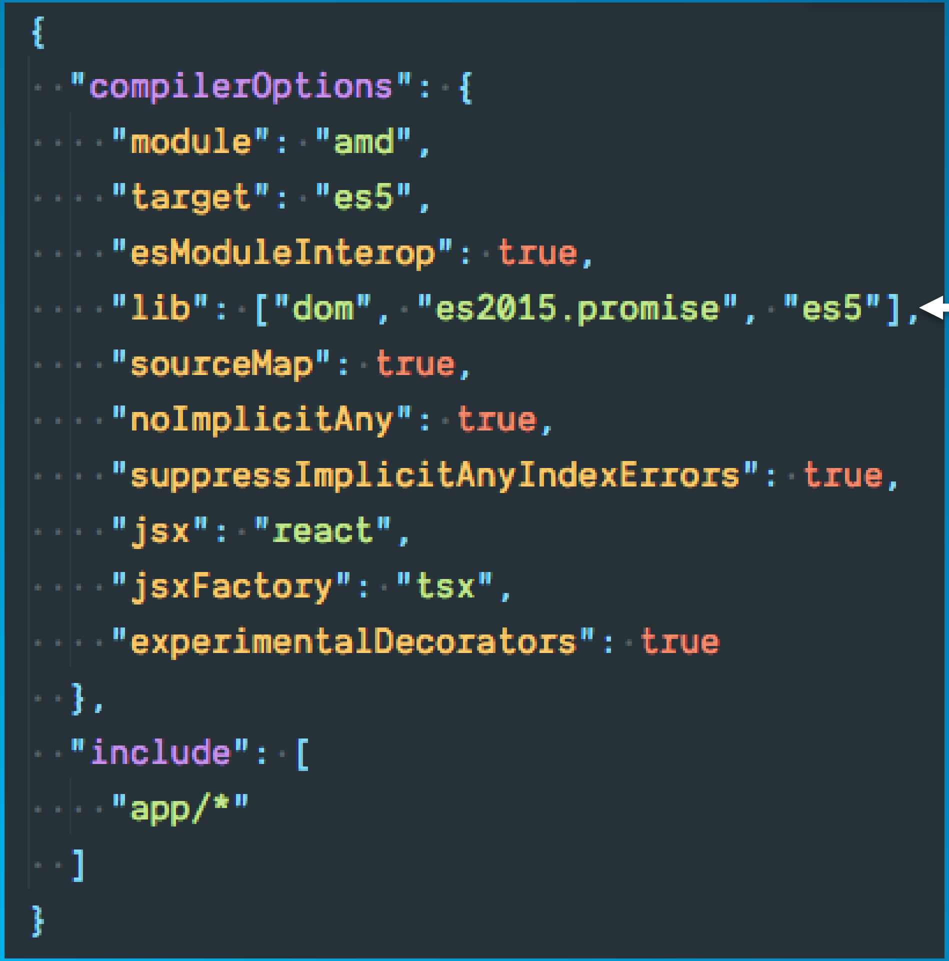


```json
{
  "compilerOptions": {
    "module": "amd",
    "target": "es5",
    "esModuleInterop": true
  },
  "include": [
    "app/*"
  ]
}
```

Where are my TypeScript files?

TypeScript – tsconfig.json

Optional Configuration

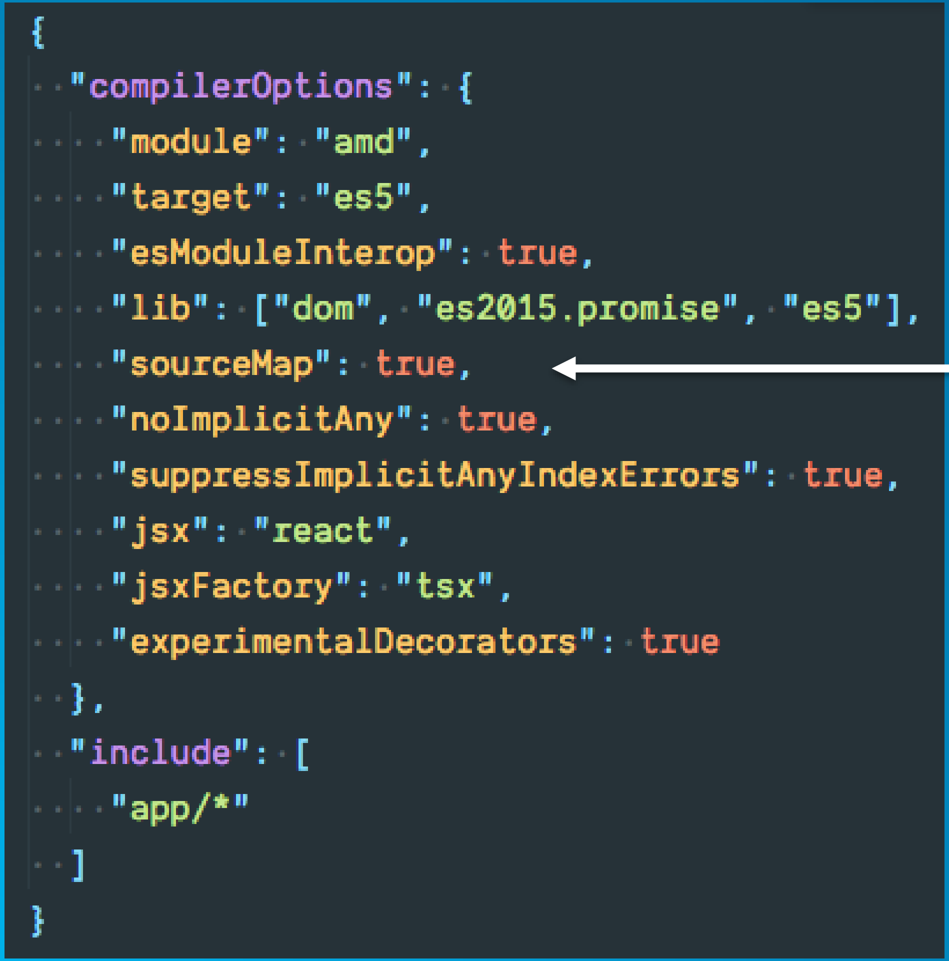


```json
{
  "compilerOptions": {
    "module": "amd",
    "target": "es5",
    "esModuleInterop": true,
    "lib": ["dom", "es2015.promise", "es5"],
    "sourceMap": true,
    "noImplicitAny": true,
    "suppressImplicitAnyIndexErrors": true,
    "jsx": "react",
    "jsxFactory": "tsx",
    "experimentalDecorators": true
  },
  "include": [
    "app/*"
  ]
}
```

Needed for async/await

TypeScript – tsconfig.json

Optional Configuration

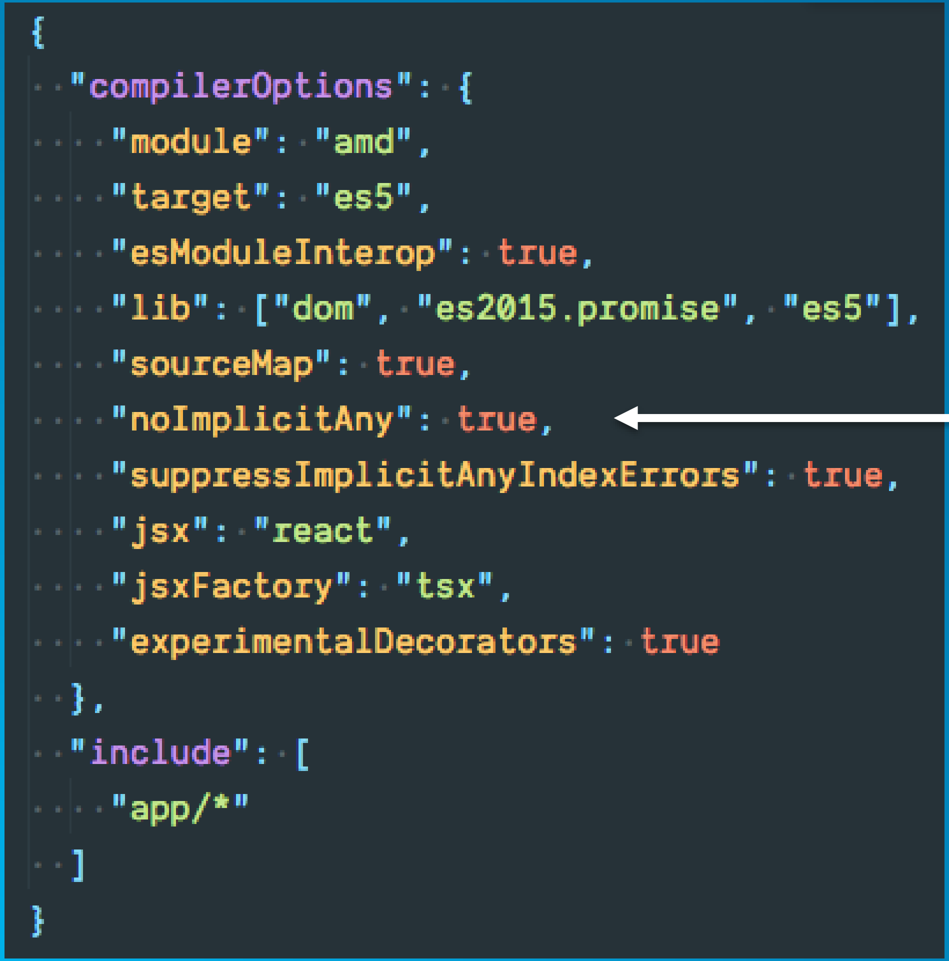


```
{
  "compilerOptions": {
    "module": "amd",
    "target": "es5",
    "esModuleInterop": true,
    "lib": ["dom", "es2015.promise", "es5"],
    "sourceMap": true,
    "noImplicitAny": true,
    "suppressImplicitAnyIndexErrors": true,
    "jsx": "react",
    "jsxFactory": "tsx",
    "experimentalDecorators": true
  },
  "include": [
    "app/*"
  ]
}
```

Output sourcemaps for debugging

TypeScript – tsconfig.json

Optional Configuration

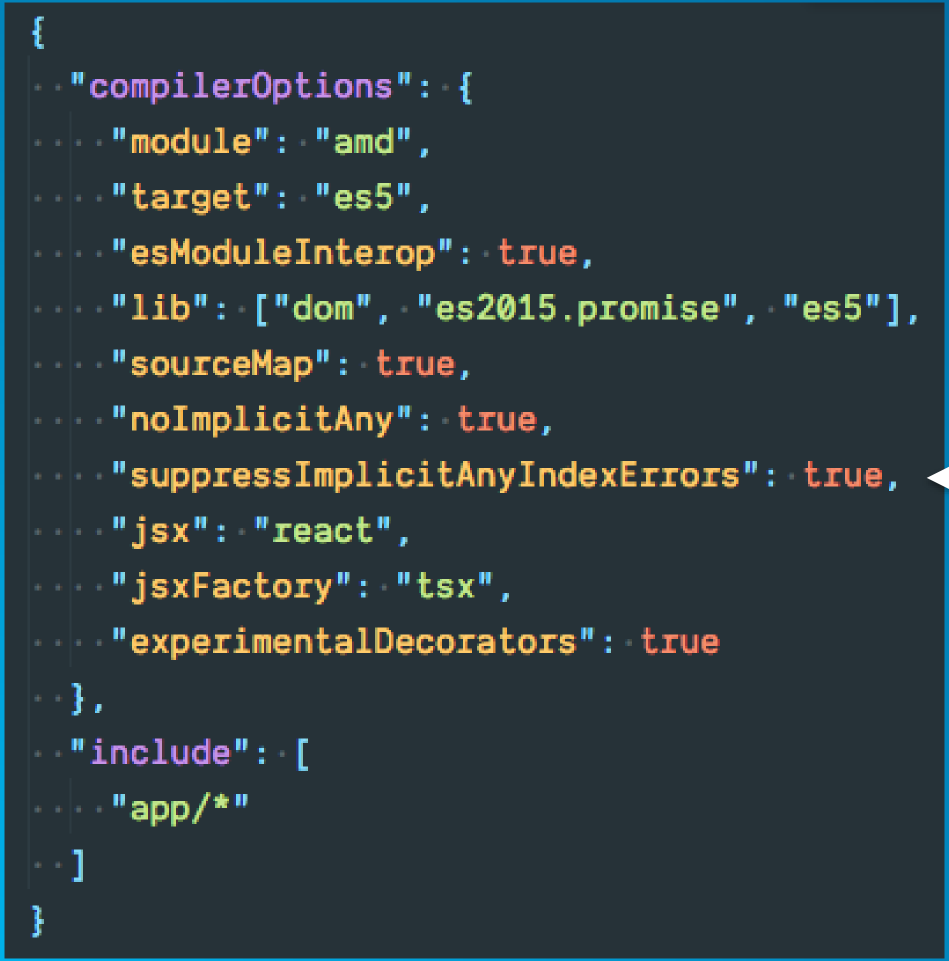


```json
{
  "compilerOptions": {
    "module": "amd",
    "target": "es5",
    "esModuleInterop": true,
    "lib": ["dom", "es2015.promise", "es5"],
    "sourceMap": true,
    "noImplicitAny": true,
    "suppressImplicitAnyIndexErrors": true,
    "jsx": "react",
    "jsxFactory": "tsx",
    "experimentalDecorators": true
  },
  "include": [
    "app/*"
  ]
}
```

You can use `any` type, but must declare it

TypeScript – tsconfig.json

Optional Configuration

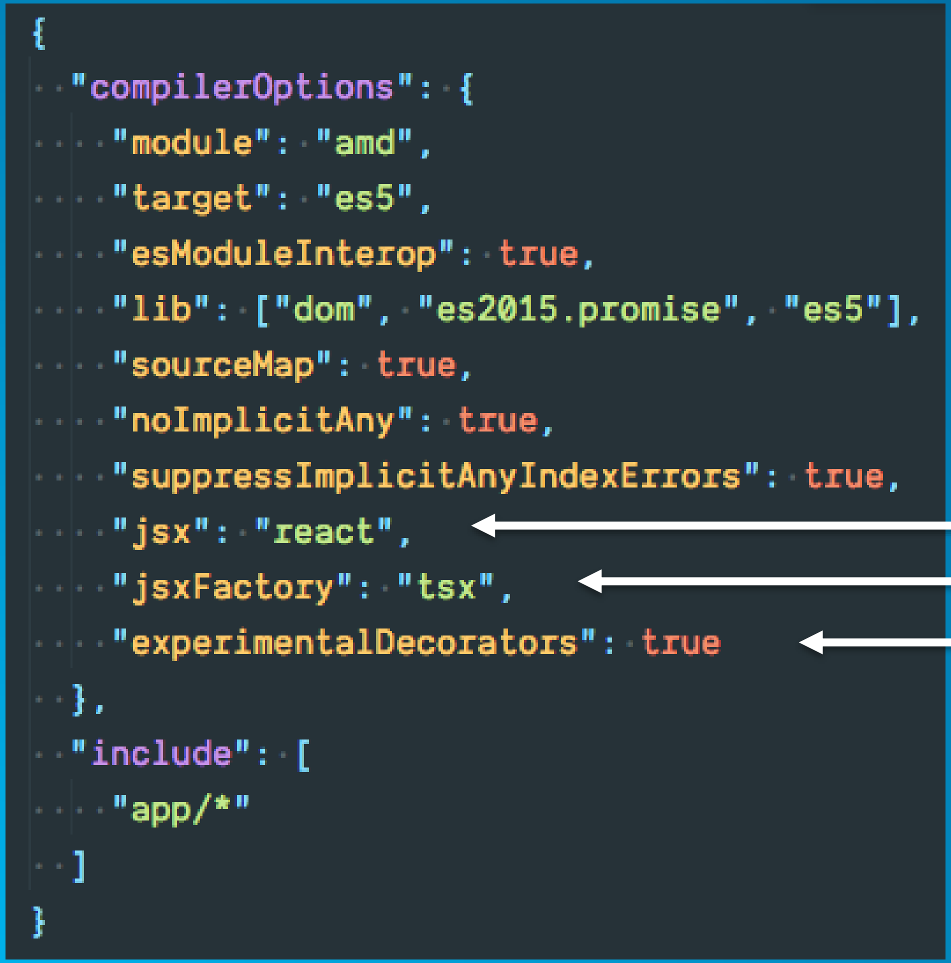


```json
{
  "compilerOptions": {
    "module": "amd",
    "target": "es5",
    "esModuleInterop": true,
    "lib": ["dom", "es2015.promise", "es5"],
    "sourceMap": true,
    "noImplicitAny": true,
    "suppressImplicitAnyIndexErrors": true,
    "jsx": "react",
    "jsxFactory": "tsx",
    "experimentalDecorators": true
  },
  "include": [
    "app/*"
  ]
}
```

Suppress the noImplicityAny errors for indexing objects

TypeScript – tsconfig.json

Optional Configuration



```
{
  "compilerOptions": {
    "module": "amd",
    "target": "es5",
    "esModuleInterop": true,
    "lib": ["dom", "es2015.promise", "es5"],
    "sourceMap": true,
    "noImplicitAny": true,
    "suppressImplicitAnyIndexErrors": true,
    "jsx": "react",
    "jsxFactory": "tsx",
    "experimentalDecorators": true
  },
  "include": [
    "app/*"
  ]
}
```

Used for custom widget development

TypeScript Features

- Types and Interfaces
- Type Guards
- Dynamic Imports

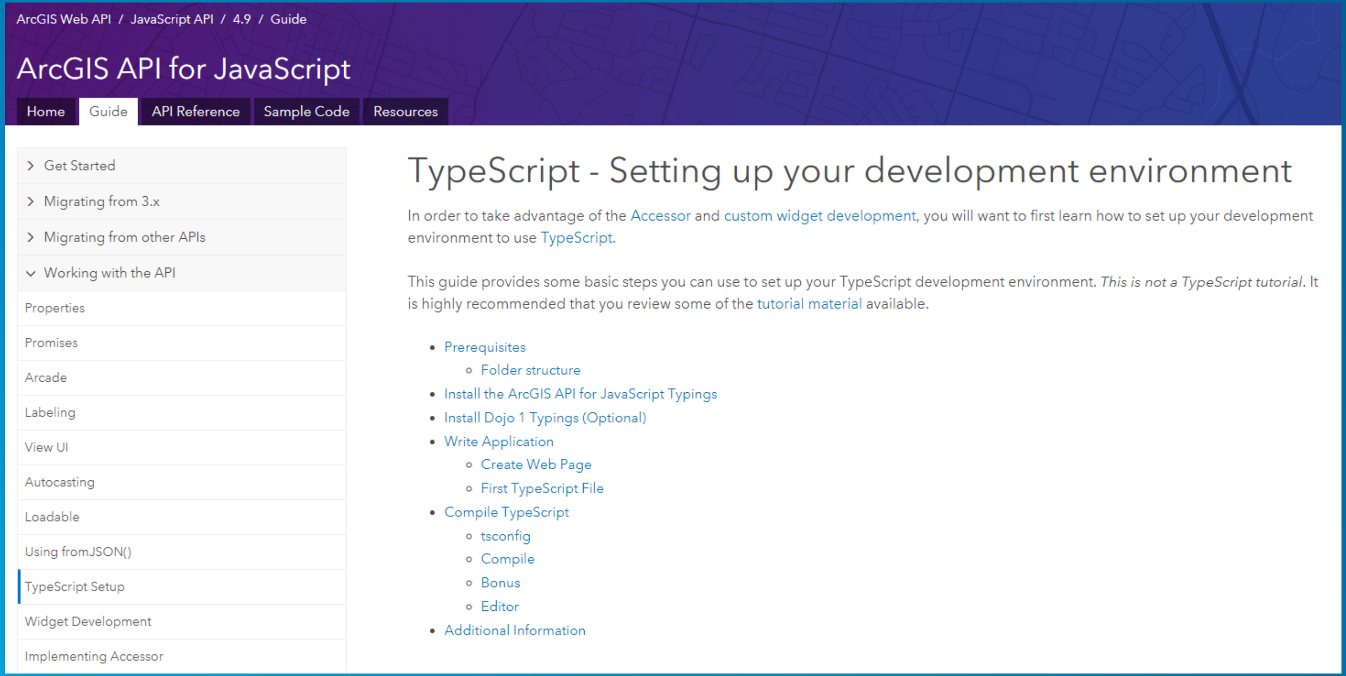
TypeScript

Simple Example

TypeScript

More Involved Example

Resources


ArcGIS Web API / JavaScript API / 4.9 / Guide

ArcGIS API for JavaScript

Home | Guide | API Reference | Sample Code | Resources

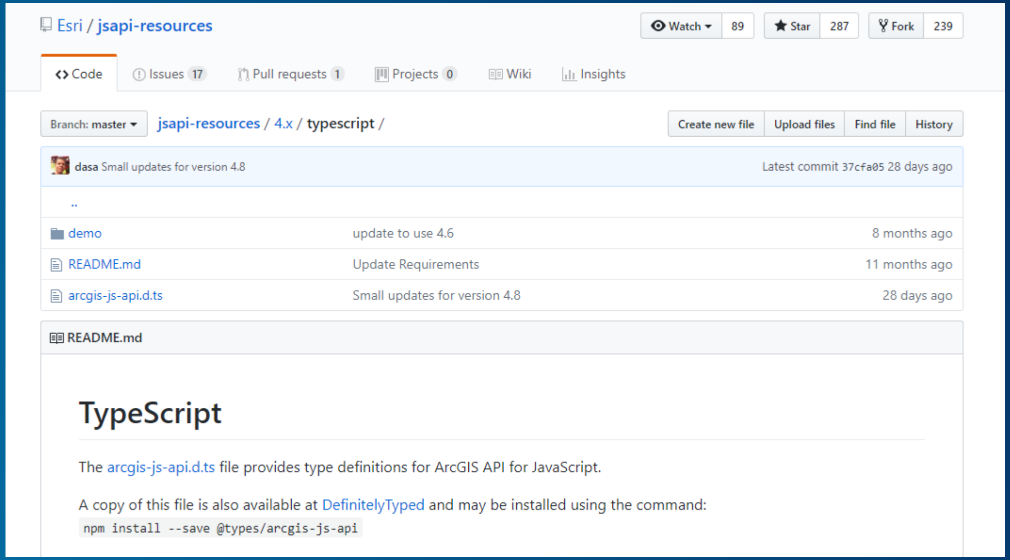
- Get Started
- Migrating from 3.x
- Migrating from other APIs
- Working with the API
  - Properties
  - Promises
  - Arcade
  - Labeling
  - View UI
  - Autocasting
  - Loadable
  - Using fromJSON()
  - TypeScript Setup
  - Widget Development
  - Implementing Accessor

TypeScript - Setting up your development environment

In order to take advantage of the Accessor and custom widget development, you will want to first learn how to set up your development environment to use TypeScript.

This guide provides some basic steps you can use to set up your TypeScript development environment. *This is not a TypeScript tutorial*. It is highly recommended that you review some of the tutorial material available.

- Prerequisites
  - Folder structure
- Install the ArcGIS API for JavaScript Typings
- Install Dojo 1 Typings (Optional)
- Write Application
  - Create Web Page
  - First TypeScript File
- Compile TypeScript
  - tsconfig
  - Compile
  - Bonus
  - Editor
- Additional Information


Esri / jsapi-resources

Watch 89 | Star 287 | Fork 239

Code | Issues 17 | Pull requests 1 | Projects 0 | Wiki | Insights

Branch: master | jsapi-resources / 4.x / typescript /

Create new file | Upload files | Find file | History

dasa Small updates for version 4.8 — Latest commit 37cfa05 28 days ago

..		
demo	update to use 4.6	8 months ago
README.md	Update Requirements	11 months ago
arcgis-js-api.d.ts	Small updates for version 4.8	28 days ago

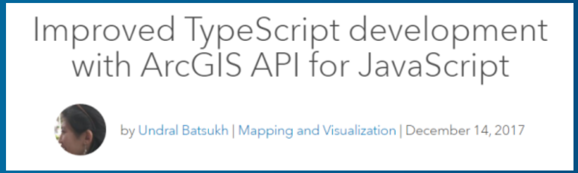
README.md

TypeScript

The arcgis-js-api.d.ts file provides type definitions for ArcGIS API for JavaScript.

A copy of this file is also available at DefinitelyTyped and may be installed using the command:

```
npm install --save @types/arcgis-js-api
```


Improved TypeScript development with ArcGIS API for JavaScript

by Undral Batsukh | Mapping and Visualization | December 14, 2017

Using TypeScript with the ArcGIS API for JavaScript

Nick Senger & Jesse van den Kieboom

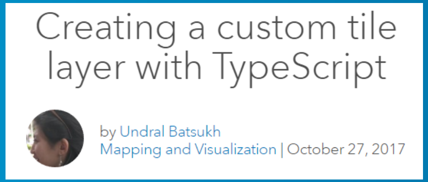
Esri Developer Summit 2018

TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Any browser. Any host. Any OS. Open source.


Creating a custom tile layer with TypeScript

by Undral Batsukh
Mapping and Visualization | October 27, 2017

Video

Using TypeScript with ArcGIS API for JavaScript

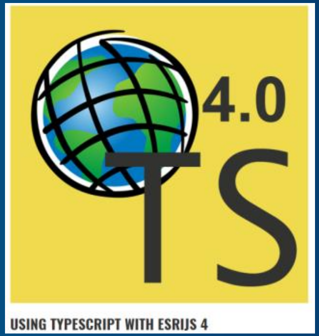
Created March 23, 2017

Duration: 58 Minutes
Cost: Requires Maintenance

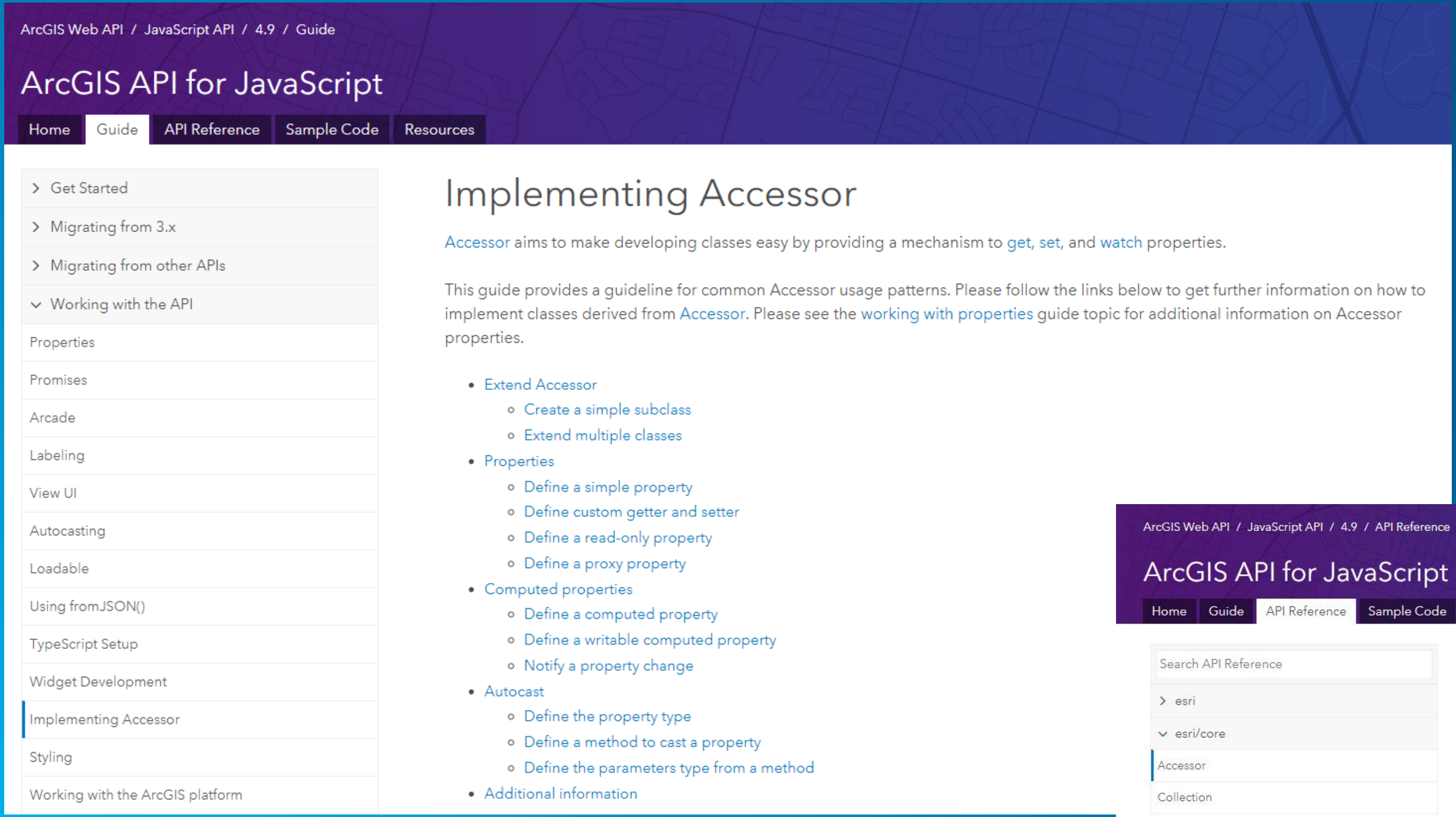
About this Video

Esri Developer Summit 2017 - Technical Workshop

Learn some of the basics of TypeScript and how it can help you as a developer using the ArcGIS API for JavaScript. This session will also show you how to set up a development environment and use the provided TypeScript definition files.



Implementing Accessor - esri/core/Accessor class

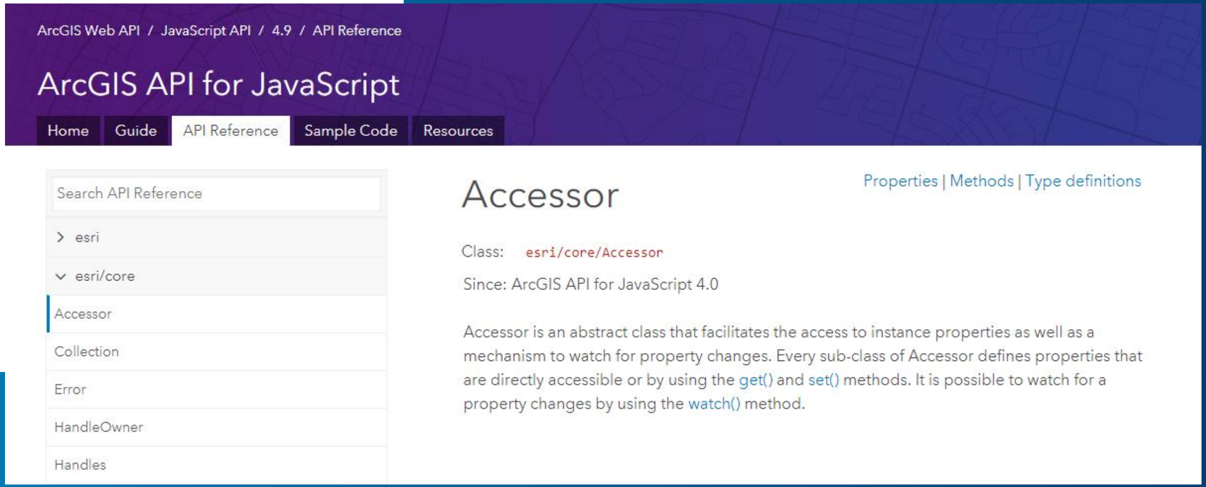


Implementing Accessor

Accessor aims to make developing classes easy by providing a mechanism to get, set, and watch properties.

This guide provides a guideline for common Accessor usage patterns. Please follow the links below to get further information on how to implement classes derived from Accessor. Please see the working with properties guide topic for additional information on Accessor properties.

- Extend Accessor
  - Create a simple subclass
  - Extend multiple classes
- Properties
  - Define a simple property
  - Define custom getter and setter
  - Define a read-only property
  - Define a proxy property
- Computed properties
  - Define a computed property
  - Define a writable computed property
  - Notify a property change
- Autocast
  - Define the property type
  - Define a method to cast a property
  - Define the parameters type from a method
- Additional information


ArcGIS Web API / JavaScript API / 4.9 / API Reference
ArcGIS API for JavaScript
Home
Guide
API Reference
Sample Code
Resources
Search API Reference
esri
esri/core
Accessor
Collection
Error
HandleOwner
Handles
Properties | Methods | Type definitions

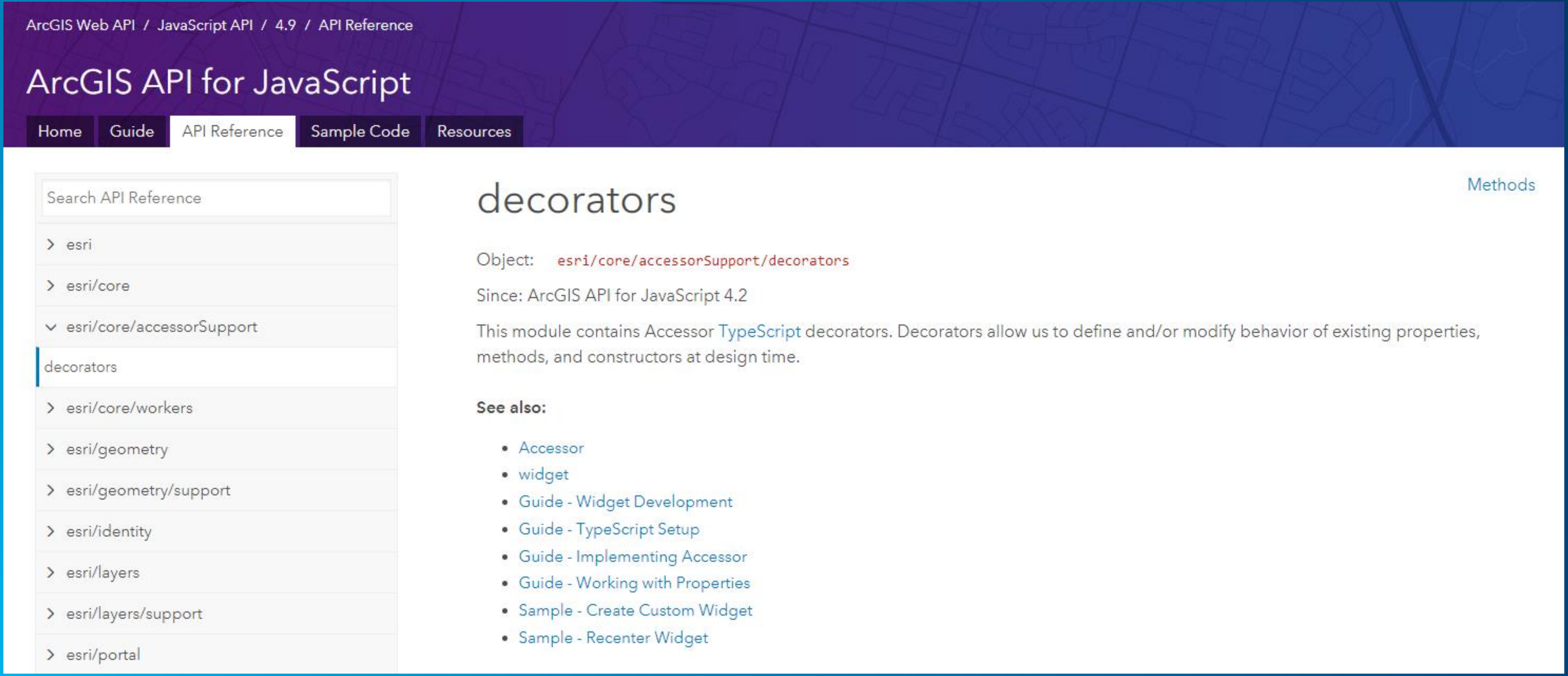
Accessor

Class: `esri/core/Accessor`

Since: ArcGIS API for JavaScript 4.0

Accessor is an abstract class that facilitates the access to instance properties as well as a mechanism to watch for property changes. Every sub-class of Accessor defines properties that are directly accessible or by using the get() and set() methods. It is possible to watch for a property changes by using the watch() method.

esri/core/accessorSupport/decorators module



ArcGIS Web API / JavaScript API / 4.9 / API Reference

ArcGIS API for JavaScript

decorators

Object: `esri/core/accessorSupport/decorators`

Since: ArcGIS API for JavaScript 4.2

This module contains Accessor TypeScript decorators. Decorators allow us to define and/or modify behavior of existing properties, methods, and constructors at design time.

See also:

- Accessor
- widget
- Guide - Widget Development
- Guide - TypeScript Setup
- Guide - Implementing Accessor
- Guide - Working with Properties
- Sample - Create Custom Widget
- Sample - Recenter Widget

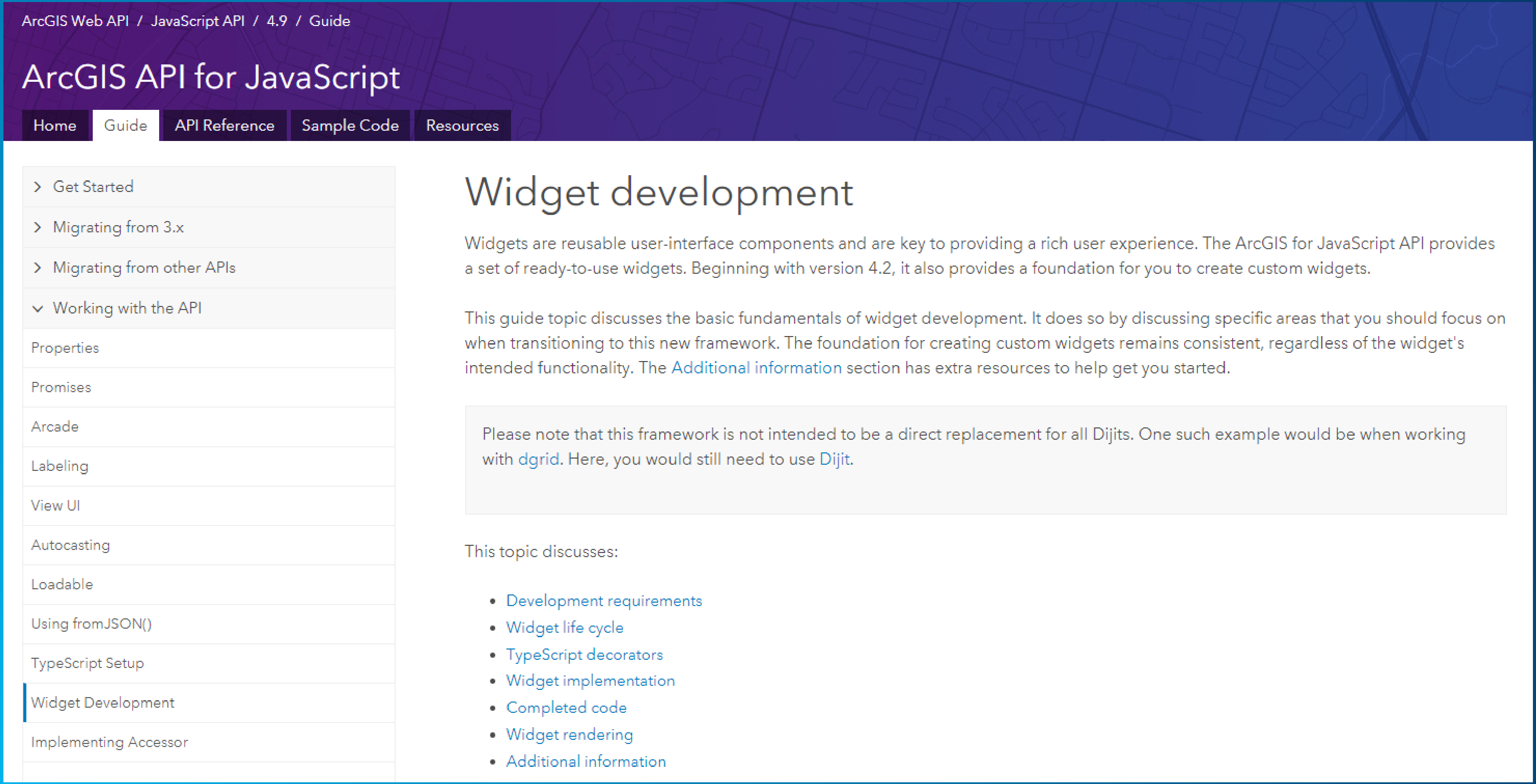
esri/core/accessorSupport/decorators – aliasOf()

```
class HelloWorld extends declared(Widget) {
  @aliasOf("viewModel.name") name: string;

  @property()
  @renderable()
  emphasized: boolean = false;

  @property({
    type: HelloWorldViewModel
  })
  @renderable("name")
  viewModel: HelloWorldViewModel;
```

esri/core/accessorSupport/decorators – property()

```
@subclass("esri.widgets.HelloWorld.HelloWorldViewModel")
class HelloWorldViewModel extends declared(Accessor) {
  @property({
    value: "Art Vandelay"
  })
  name: string;

  getGreeting() {
    return `Hello, my name is ${this.name}!`;
  }
}

export = HelloWorldViewModel;
```

Widget development



ArcGIS Web API / JavaScript API / 4.9 / Guide

ArcGIS API for JavaScript

Widget development

Widgets are reusable user-interface components and are key to providing a rich user experience. The ArcGIS for JavaScript API provides a set of ready-to-use widgets. Beginning with version 4.2, it also provides a foundation for you to create custom widgets.

This guide topic discusses the basic fundamentals of widget development. It does so by discussing specific areas that you should focus on when transitioning to this new framework. The foundation for creating custom widgets remains consistent, regardless of the widget's intended functionality. The Additional information section has extra resources to help get you started.

Please note that this framework is not intended to be a direct replacement for all Dijits. One such example would be when working with dgrid. Here, you would still need to use Dijit.

This topic discusses:

- Development requirements
- Widget life cycle
- TypeScript decorators
- Widget implementation
- Completed code
- Widget rendering
- Additional information

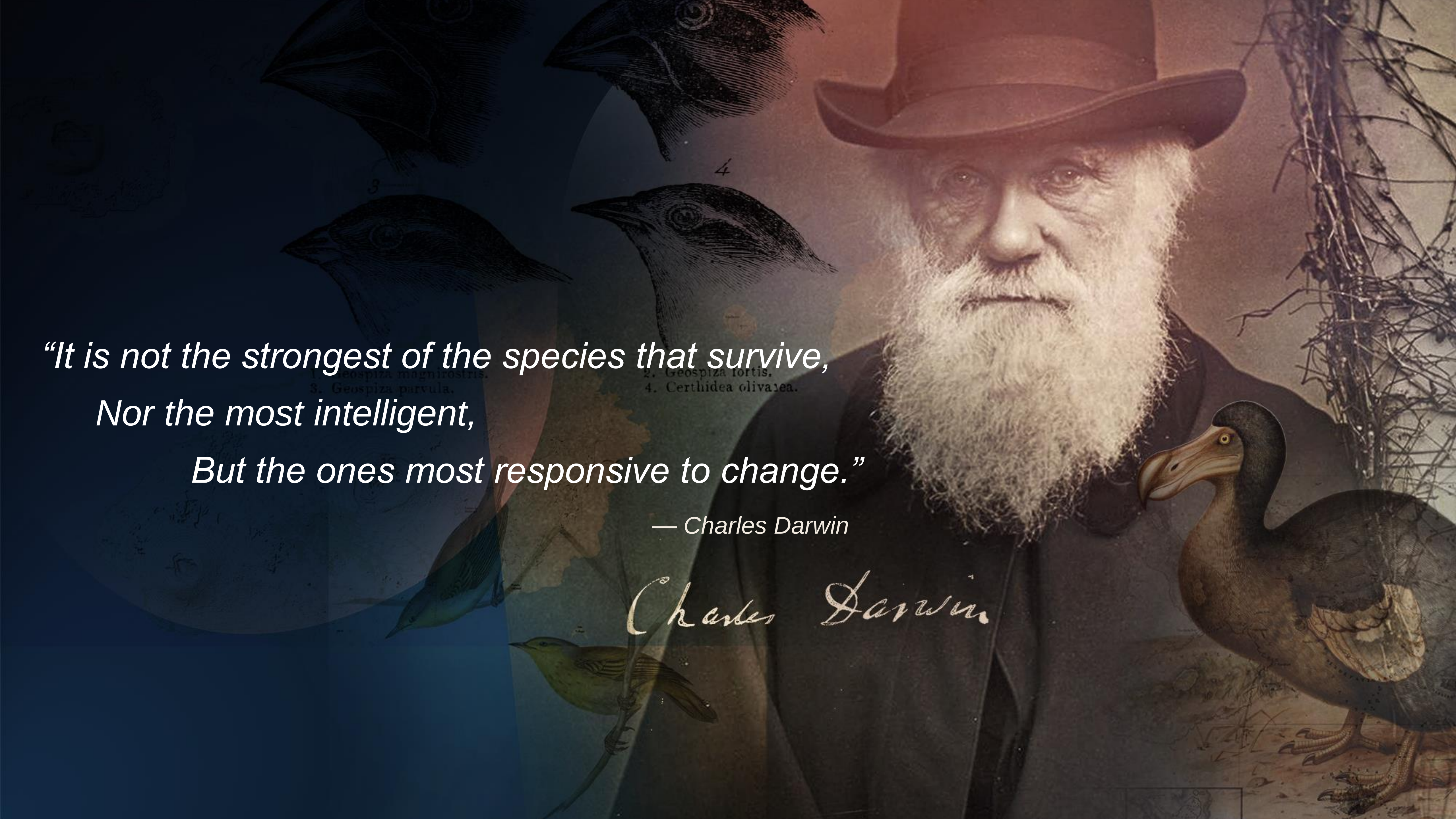
TypeScript Example

Custom Widget Example

TypeScript Example

We can help make it easier for you!

```
npm install @arcgis/cli
```



"It is not the strongest of the species that survive,

Nor the most intelligent,

But the ones most responsive to change."

— Charles Darwin

Charles Darwin

esri®

THE SCIENCE OF WHERE™