precisely

Spectrum Spatial Analyst

Extensibility User Guide

Version 2022.1

Table of Contents

8 - Referencing Spectrum Spatial Analyst APIs in your Component

9 - Gloassary of Terms

Appendix A: Appendix

1 - Introduction

Spectrum Spatial Analyst is an extensive web mapping platform that developers can customize. Its framework is based on the Angular component model, which lets you create and add new components to Spectrum Spatial Analyst or alter the behavior of an existing components. This document describes how to extend Spectrum Spatial Analyst, explains related concepts, and gives examples. Read this guide along with the extensibility API documentation.

In this section

Overview

Spectrum Spatial Analyst lets you add Angular based components dynamically at run-time to an already deployed and running instance.

Components in Spectrum Spatial Analyst are built as standard Angular 11 components (written in TypeScript) and are included in an Angular 11 module. An Angular 11 module can include one or more components. Components are injected dynamically into the application at designated places called injection points. The image on page 6 showcases some of the extension points, such as the places to add new components. You can create components to replace existing Spectrum Spatial Analyst components, such as menus, or to dynamically inject into existing Spectrum Spatial Analyst components. Injection points have a unique identifier. The parent components where new extensions will be injected are called containers.

It is not necessary to compile components or build a custom version of Spectrum Spatial Analyst.

Static Container

In the Spectrum Spatial Analyst Extensibility Platform, the Static containers are available for injection into the Settings panel, Layer panel, and Right panel.

These injection points are available for the developers to add their components. Regardless of the type of data or map you host onSpectrum Spatial Analyst, from Spectrum Spatial, a Web Mapping Service (WMS), or Vector layer.

Dynamic Container

Dynamic containers are available based on the data that the Spectrum Spatial Analyst Extensibility Platform hosts. For example, a Spectrum Spatial legend item is only available when a Spectrum Spatial layer is included in the Spectrum Spatial Analyst Map configuration. Dynamic containers also pass in context data to the injected component. For example, if a component is a child of the Annotation legend item, then that child component will have access to annotation information like annotation name, annotation center, extents of annotation, and so on (which the child component can use to determine how it behaves and even whether it is rendered).

Removable Component

The Spectrum Spatial Analyst user interface has multiple containers and components such as the Left panel, Right panel, Map, Legend, and Search box. Some of these can be removed or replaced with custom components using the Spectrum Spatial Analyst Extensibility Platform. You can remove components using the functionality profiles in Spatial Manager or using the Spectrum Spatial Analyst Extensibility Platform config file.

To inject a new component at one of the available extension points, a configuration file called `CustomAnalystModuleConfig.json` is used. This file configures containers for third-party extensions, indicates which components to remove, and includes parameters for using extensions.

Spectrum Spatial Analyst capabilities have been exposed as APIs, which third-party components can use in their logic. For example, adding and removing map layers, calling different Spectrum Spatial services such as data flows, specifying queries, thematically styling map layers, and so on. All these services have APIs which encapsulate a wide variety of third-party libraries that are part of Spectrum Spatial Analyst, such as Openlayers (mapping), Proj4JS (re-projection of vector data), JSTS (geometry operations on vector data), jsPDF (for exporting to PDF), XLSXJS (for parsing Excel spreadsheets) and Papa Parse (for parsing CSV files).

Spectrum Spatial Analyst uses an architecture based on NgRx Store (https://ngrx.io/guide/store) for maintaining state and providing inter-component communications. Many services are available to developers via store actions and their corresponding selectors. The store is a bridge between the caller and executor. To draw a layer on a map, you would dispatch an action with the relevant parameters via store.dispatch.

2 - Hello World Extension Example

This section explains how to embed an Angular 11 component into Spectrum Spatial Analyst application.

In this section

Prerequisite

- Basic knowledge of Angular 2/4
- Ability to code in Typescript
- Basic understanding of Spectrum Spatial Analyst (such as customerconfigurations, map projects, and Spatial Manager)

Embedding Angular 11 component

To embed an Angular 11 component into Spectrum Spatial Analyst

1. Create an Angular 11 component; for example, HelloWorldComponent.
2. Create an Angular 11 module; for example, DynamicModule containing the HelloWorldComponent.
3. Place the module and component file into the folder under the `customerconfigurations/analyst/theme/extensions` folder.
4. Create or update a module definition file representing that component.
5. Validate the module definition file with the Spectrum Spatial Analyst Custom Modules file validator (web page).
6. Put the module definition file into the custom configuration folder once the file is validated.
7. Refresh the browser to see the component embedded in Spectrum Spatial Analyst.

Angular Compilers

Angular provides two ways to compile your application, Just-in-Time (JIT) and Ahead-of-Time (AOT).

Just-in-Time (JIT) Mode

Just in Time (JIT) compiler provides compilation during the execution of the program at a run time. In simple words, code get compiled when it is needed, not at the build time. In JIT mode, the angular compiler gets shipped to the browser.

Ahead-of-Time (AOT) Mode

The application is compiled ahead of time on the server side. The Ahead of Time (AOT) compiler converts the code during the build time before your browser downloads and runs that code.

Creating and Invoking the Hello World Extension in JIT mode

This section describes how to add a new menu item called **Hello World Extension** to the **Add Panel** menu, which prompts an alert message when clicked.

1. Create a folder named extensions under `<ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\`. For example, a typical installation path looks like: `C:\Program Files\Precisely\SpectrumSpatialAnalyst\customerconfigurations\analyst\theme`.
2. Create a file called `dynamic.component.ts` under the `<ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\extensions` folder.
3. Paste the following code snippet into the file:

```
import {Component, Input} from '@angular/core';
            import {ComponentFactoryResolver} from '@angular/core';
            import {ViewContainerRef} from '@angular/core';
            @Component({
            selector: 'hello-world-selector',
            template: `<div (click)=sayHello() class=""><img
class="fillColor"
            src="../controller/theme/extensions/icon-circle.png"
alt="icon-circle" height="25"
            width="25">Hello World Extension</div>`,
            styles: [`
            .btnPosition {
            z-index: 1;
            right: 12%;
            }
            .iconContainer {
            padding: 10px;
           background-image: linear-gradient(90deg,#3e53a4,#cf0989);

            }
            .fillColor {margin: 3px; cursor:pointer;}
            `]
            })
            export class HelloWorldComponent{
            constructor() {
            }
            onInit() {
            }
            sayHello() {
           alert('Congratulations! You have successfully extended the
 Spectrum Spatial Analyst application.');
            }
            }
```

4. Save the following image as icon-circle.png in: `<ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\extensions`.

   Please note that theme is an existing folder in the Spectrum Spatial Analyst installation.

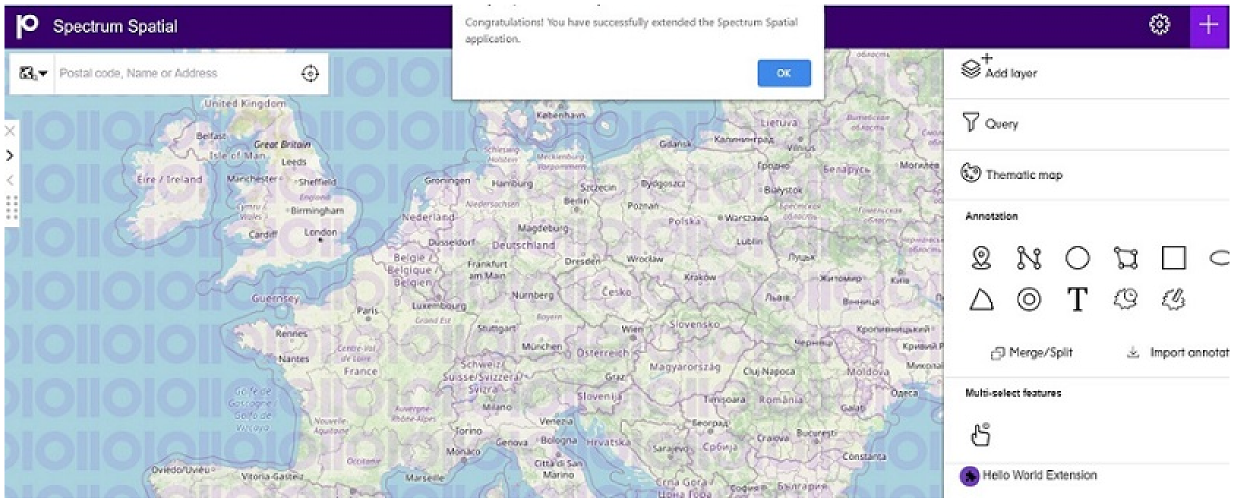
5. Create a file called dynamic.module.ts in:`<ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\extensions`.
6. Paste the following content in that file and save it:

```
import {NgModule } from '@angular/core';
          import {HelloWorldComponent} from './dynamic.component.ts';

            @NgModule({
            imports: [],
            declarations: [HelloWorldComponent],
            exports: [HelloWorldComponent]
            })
            export class DynamicModule { };
```

7. Go to the customer configuration folder in: `<ANALYST_INSTALL_PATH>\customerconfigurations\analyst`.The path looks like `C:\Program Files\Precisely\SpectrumSpatialAnalyst\customerconfigurations\analyst`.
8. Create a file called CustomAnalystModuleConfig.json under that directory.
9. Paste the following content into the CustomAnalystModuleConfig.json file and save it:

```
{
            "modules": [{
            "name": "DynamicModule",
            "description": "Hello World Extension in the Add Panel.
          "modulePath": "extensions/dynamic.module.ts#DynamicModule",

            "components": [{
            "componentName": "HelloWorldComponent",
            "parentComponentName": "AddPanel"
            }]
            }],
            "componentsToRemove": []
            }
```

10. Open the browser and launch the Spectrum Spatial Analyst URL from the address bar.
11. Go to the Add panel, (+) plus button on top right corner of application, to see your first component with the above image and label Hello World Extension
12. Click on it to display a pop up with a success message as shown in the following image:



Congratulations! You have successfully extended the Spectrum Spatial application.

Creating and Invoking the Hello World Extension in AOT mode

This section describes how to add a new menu item called **Hello World Extension** to the **Add Panel** menu, which prompts an alert message when clicked.

1. Create a folder named extensions under `<ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\`. For example, a typical installation path looks like: `C:\Program Files\Precisely\SpectrumSpatialAnalyst\customerconfigurations\analyst\theme`.
2. Create a file called `dynamic.component.ts` under the `<ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\extensions` folder.
3. Paste the following code snippet into the file:

```
import {Component, Input} from '@angular/core';
            import {ComponentFactoryResolver} from '@angular/core';
            import {ViewContainerRef} from '@angular/core';
            @Component({
            selector: 'hello-world-selector',
            template: `<div (click)=sayHello() class=""><img
class="fillColor"
            src="../controller/theme/extensions/icon-circle.png"
alt="icon-circle" height="25"
            width="25">Hello World Extension</div>`,
            styles: [`
```

```
           .btnPosition {
           z-index: 1;
           right: 12%;
           }
           .iconContainer {
           padding: 10px;
           background-image: linear-gradient(90deg,#3e53a4,#cf0989);

           }
           .fillColor {margin: 3px; cursor:pointer;}
           `]
           })
           export class HelloWorldComponent{
           constructor() {
           }
           onInit() {
           }
           sayHello() {
          alert('Congratulations! You have successfully extended the
Spectrum Spatial Analyst application.');
           }
           }
```

4. Save the following image as icon-circle.png in: `<ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\extensions`.

   Please note that theme is an existing folder in the Spectrum Spatial Analyst installation.

5. Create a file called dynamic.module.ts in:`<ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\extensions`.
6. Paste the following content in that file and save it:

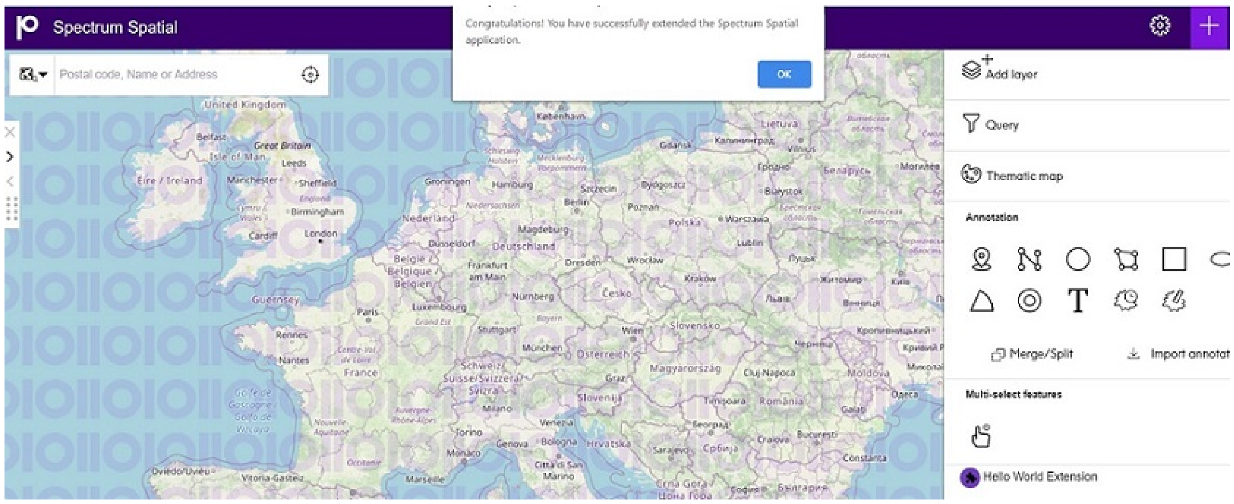
```
import {NgModule } from '@angular/core';
          import {HelloWorldComponent} from './dynamic.component.ts';

            @NgModule({
            imports: [],
            declarations: [HelloWorldComponent],
            exports: [HelloWorldComponent]
            })
            export class DynamicModule { };
```

7. Go to the customer configuration folder in: `<ANALYST_INSTALL_PATH>\customerconfigurations\analyst`.The path looks like `C:\Program Files\Precisely\SpectrumSpatialAnalyst\customerconfigurations\analyst`.
8. Create a file called CustomAnalystModuleConfig.json under that directory.

9. Paste the following content into the CustomAnalystModuleConfig.json file and save it:

```
{
            "modules": [{
            "name": "DynamicModule",
            "description": "Hello World Extension in the Add Panel.
          "modulePath": "extensions/dynamic.module.ts#DynamicModule",

            "components": [{
            "componentName": "HelloWorldComponent",
            "parentComponentName": "AddPanel"
            }]
            }],
            "componentsToRemove": []
            }
```

10. Open the browser and launch the Spectrum Spatial Analyst URL from the address bar.
11. Go to the Add panel, (+) plus button on top right corner of application, to see your first component with the above image and label Hello World Extension
12. Click on it to display a pop up with a success message as shown in the following image:



Congratulations! You have successfully extended the Spectrum Spatial application.

AOT-enabled Extension Support

AOT-enabled extension support in Spectrum Spatial Analyst:

Creating an extension Development Environment

1. npm install --global @angular/cli
2. ng new <project-name> --create-application=false

3. cd <project-name>
4. ng generate library <lib-name>
5. copy provided Spectrum Spatial Analyst typings in a dir out of <project-name> dir and provide path in package.json for property ‘analyst’ in devDependecies. For example:

   "analyst": "file:../ssa-typings",

6.
   ```
   Run npm i @angular-devkit/build-webpack @ngtools/webpack
   ```
7. Copy provided webpack.config.js and webpack.config.prod.js in <lib-name> directory
8. Change entry file and tsConfigPath in webpack.config.js
9. Replace @angular-devkit/build-angular:ng-packagr with @angular-devkit/build-webpack:webpack in angular.json
10. provide "webpackConfig": "projects/<lib-name>/webpack.config.js" in options of angular.json
11. provide "webpackConfig": "projects/<lib-name>/webpack.config.prod.js" in configurations > production of angular.json
12. ng build <lib-name>

```
Run ng generate component component-name --project my-lib to generate a
 new component
```

You can also use:

```
ng generate directive|pipe|service|class|guard|interface|enum|module
--project my-lib
```

Summary

In the first few steps of the above Hello World example (for an Angular component and Angular module) included resources for the component, such as an image. We then created the configuration to inject that component into the Spectrum Spatial Analyst Extensibility Platform.

The parentContainer tag in the `CustomAnalystModuleConfig.json` file is responsible for injecting a component into the correct container. The Spectrum Spatial Analyst Extensibility Platform provides multiple containers for third-party components to position them in the screen layout correctly. For details about containers, refer to the following sections.

Note: The name of the class in the component file and the name of the component in the configuration file should be identical because this is the main link between the component and configuration. The Angular module should have a declaration of the component it embeds. There are no restrictions on the number of components that a single angular module can have.

3 - Upgrading Extensible Components

In this section

Upgrading Extensible Components

Spectrum Spatial Analyst enables you to migrate your existing extensible components to Angular 11. The following sections will help you successfully migrate your components.

You need to make the following changes to run the extensible components in the migrated Spectrum Spatial Analyst application.

1. NgRx Store

The NgRx Store has been migrated from version 2.2.3 to version 11.1.1. This upgrade requires some syntax changes as explained below:

- First change: Remove the `new` operator while dispatching the action. This is must for every action's dispatch that you are using in your extensible component.

  In Angular 4, whenever you dispatched an action, you would use the below syntax:

```
this.store.dispatch(new actionName(payload));
```

  Now, you need to remove the `new` operator in Spectrum Spatial Analyst Angular 11's extensible components so that it looks like:

```
this.store.dispatch(actionName(payload));
```

  To create a query in Spectrum Spatial Analyst make the changes as described in the table below.

To create a query layer in Angular 4	To create a query layer in Angular 11

`const params = {`
`query: "SELECT`
`Country,Capital,Continent`
`FROM`
`\"/Samples/NamedTables/WorldTable\"`
 	
`WHERE Lower(Country) LIKE`
`Lower('%In%')",`
`layerName: "Country",`
`name: "Countries name containing`
`'In'",`
`addToMap: addToMap,`
`getResult: getResult,`
`orderIndex: 0,`
`replaceCurrentLegend: false,`
`customComponentData: {`
`customDataObj: "customDataObj" }`
	`this.store.dispatch(ExecuteQueryAction(params));`
	
	`// No change in the payload passed`
	` to ExecuteQueryAction.`
	`Only change is removing the new`
	`operator.`

To create a query layer in Angular 4	To create a query layer in Angular 11
`}; this.store.dispatch(new ExecuteQueryAction(params));`	

- Second change: Modify the payload passed to action while dispatching it. It is required only when your payload is of type string, number, Boolean or an array. If the payload is of type JSON object like in the above example of query, then there is no need to make any change in the payload. For example,

```
this.store.dispatch(new ToggleResourceSelectorVisiblity(true));
```

To dispatch action when payload is of type Boolean:

Angular 4	Angular 11
`this.store.dispatch(new ToggleResourceSelectorVisiblity(true));`	`this.store.dispatch(ToggleResourceSelectorVisiblity(` **`{toggleVisiblity: true}`** `));`

So, if the payload is of primitive type like Boolean, make the following changes:

- Remove the `new` operator as explained above.
- Wrap the Boolean payload in a JSON object. The key which has been introduced for `ToggleResourceSelectorVisiblity` action is `toggleVisiblity`. Each actions have different key name, refer to the API documentation for details on actions and the corresponding keys.

2. RxJS

It has been migrated from version 5.4.2 to 6.6.7 and requires the following changes:

- First change: If you are using map, mergeMap, concatMap, switchMap etc. operators of RxJS in your extensible component, then you have to use pipe operator as shown below:

Using RxJS map operator in Angular 4	Using RxJS map operator in Angular 11
`this.xhrIncokerService.invokeController (options).map((data ) => {`	`this.xhrIncokerService.invokeController (options).pipe(map( (data)=> {; //logic });`

Using RxJS map operator in Angular 4

```
//logic
});
```

Using RxJS map operator in Angular 11

- Second change: Importing RxJS operators has also changed. To import certain operators like map, mergeMap, concatMap, and switchMap you can use the following code:

Importing RxJS operators in Angular 4

```
// Old version
import { Observable } from
'rxjs/Observable';
// Old version
import { Subject } from
'rxjs/Subject';
// Old version
import { Observer } from
'rxjs/Observer';
```

Importing RxJS operators in Angular 11

```
// Latest version;
import { map, mergeMap, switchMap,
 concatMap }  from
'rxjs/operators';
importing very commonly used Rxjs
 stuff like Observable,
Subscription, Subject
import {Observable, Subject,
Observer, Subscription, of } from
 'rxjs';
```

You can refer to RxJS documentation for the latest updates.

3. Open Layers

It has been migrated from 4.2.0 to 6.5.0 and requires the following changes.

Using open layers in 4.2.0

```
import * as ol from 'openlayers';
const vectorSource = new
ol.source.Vector();
const point_feature = new
ol.Feature({});
const point_geom = new
ol.geom.Point([2.0000000000345,
1.0000000245]);
point_feature.setGeometry(point_geom);
vectorSource.addFeature(point_feature);
const layer = new ol.layer.Vector({
 source: vectorSource});
const map = new ol.Map({
       layers: [layer],
 interactions:
ol.interaction.defaults({
 doubleClickZoom: false,
 shiftDragZoom: false,
```

Using open layers in 6.5.0

```
import { Map, Feature } from 'ol';
import VectorSource from
'ol/source/Vector';
import VectorLayer from
'ol/layer/Vector';
import Point from 'ol/geom/Point';
import { defaults } from
'ol/interaction';
const vectorSource = new
VectorSource();
const point_feature = new
Feature({});
const point_geom = new
Point([2.0000000000345,
1.0000000245]);
point_feature.setGeometry(point_geom);
vectorSource.addFeature(point_feature);
const layer = new VectorLayer({
```

Using open layers in 4.2.0

```
keyboard: false,
mouseWheelZoom: false,
pinchZoom: false
}),
target: 'map'
      });
```

Using open layers in 6.5.0

```
source: vectorSource });
  const map = new Map({
             layers: [layer],
  interactions: defaults({
  doubleClickZoom: false,
  shiftDragZoom: false,
             keyboard: false,
  mouseWheelZoom: false,
  pinchZoom: false
  }),
  target: 'map'
  });
```

4. Angular Framework and dependencies upgrade

If you have set up a local dev environment to create or edit extensible components, then you need to upgrade the following:

- Angular framework -> 11
- NgRxStore -> 11.1.1
- RxJS -> 6.6.7
- Openlayers -> 6.5.0

4 - Setting up Extensible AoT-enabled Dev Environment

This section explains how to set up an AoT-enabled development environment.

In this section

Steps to create an Extensible AoT-enabled Dev Environment

1. npm install --global @angular/cli
2. ng new <project-name> --create-application=false
3. cd <project-name>
4. ng generate library <lib-name>
5. Copy provided Spectrum Spatial Analyst typings in a dir out of <project-name> dir and
6. Provide path in `package.json` for property ‘analyst’ in devDependecies "analyst": "file:../ssa-typings",
7. Run npm i @angular-devkit/build-webpack @ngtools/webpack
8. Copy provided `webpack.config.js` and `webpack.config.prod.js` in the <lib-name> directory
9. Change entry file and tsConfigPath in the `webpack.config.js` file
10. Replace `@angular-devkit/build-angular:ng-packagr` with `@angular-devkit/build-webpack:webpack` in `angular.json`
11. Provide "webpackConfig": "`projects/<lib-name>/webpack.config.js`" in options of `angular.json`
12. Provide "webpackConfig": "`projects/<lib-name>/webpack.config.prod.js`" in configurations > production of `angular.json`
13. ng build <lib-name>

5 - Configuring New Components

In this section

Configurations

A model is available to define and represent all the new components that are added. This is kept in the JSON format and is contained within the **CustomModulesDefinition.json** file.

It is important to have a single file for all new modules. The order in which the components are defined is important as there may be dependencies between components. For example, a component may remove out-of-the-box components as part of its definition, but another component may be using it as a container.

The example below shows how a typical **CustomAnalystModuleConfig.json** file looks:

```
  { "modules": [{
  "name": "GIQueryModule",
  "description": "Module For GI query",
  "modulePath": "extensions/dynamic.module.ts#DynamicModule",
  "components":[{componentName:"TestComponent","parentComponentName":
"SettingsPanel",
  "initParameters": {
  "initX": 0,
  "initY": 0,
  "endPointUrl": "localhost:3306/mysql/gidata",
  }
  }}],
  "externalLibraryPath": [{
  "libName": "GDAL",
  "libPath": "../controller/theme/app/gdal.js",
  }
  ],
  "mapConfigAssociated": {
  "GeoInsightMaps": ["TestComponent"],
  "GeoInsightSummaryMaps": ["TestComponent"]
  }
  }
  ],
  ,
  "componentsToRemove":[
  {"componentName":"BaseMapSwitcherComponent",
  "fromMapConfig":"Drive Time"},
  {"componentName":"MapConfigSwitcherComponent",
  " fromMapConfig":"Drive Time"}

  ]}
  }
```

Top Level Nodes

The following table describes the parameters that can be included in a custom module definition.

Field Name	Type	Required	Description
Modules	Json Array	Yes	An array of multiple module definitions as described in the definitions section above.
ComponentsToRemove	Json Array	Yes	An array of pre-existing components provided out-of-the-box with Spectrum Spatial Analyst that would be removed.

Module JSON Object

Each array element inside the Modules node defines a module as follows.

Field Name	Type	Required	Description
Name	String	Yes	The name should not be the same as any of the Spectrum Spatial Analyst modules. You can find a list of module names in the documentation..
description	String	optional	Gives details about the purpose of the module for users looking at the configuration file.
ModulePath	String	Yes	The location of the module in the file system. This will be in: customerconfigurations/analyst/theme/extensions Note: The folder containing the module should be in this path for it to be accessible #ModuleName is mandatory in the module path to allow it to be loaded. #ModuleName is the name of the Module class in the ts file. Note Each module can have a separate folder. For example, extensions/weather/weather.module.ts#WeatherModule

Field Name	Type	Required	Description
Components	Map	Yes	This is a key value pair where • Key = Name of the component • Value = The parent container in Spectrum Spatial Analyst where the component is to be injected. Note: The Component Name should match the #ModuleName that you have declared for creating Angular 4 component class and not the selector.
externalLibraryPath	JsonArray	Optional	Set of third-party libraries that component may need for it to function. This path can be CDN or a local path relative to index.html of Spectrum Spatial Analyst.
mapConfigAssociated	Map	Optional	A key value pair where: • Key = Spectrum Spatial Analyst map configuration name • Value = Array of components that will be visible for that map configuration If the component is not explicitly associated with a mapconfig file, then it will appear for all mapconfig files that are available.
initparameters	Json Object	Optional	A key value pair where: • Key = Name of the component • Value = json object of the initialization parameters which will be passed to each instance of the component. There is no restriction on the type of init parameter; it can be any type.

ComponentToRemove

Each array element inside the ComponentsToRemove node will define the following.

Field Name	Type	Required	Description
componentName	String	Yes	Name of the component to be removed. Note this is an existing Spectrum Spatial Analyst component and not a third-party component. Specific components, like BaseMapSwitcher and SearchBox, may be removed from the application.

Field Name	Type	Required	Description
fromMapConfig	String	Optional	Removes the component from a specific mapconfig file. If omitted, it removes the component from all mapconfigs files.

Containers

The Spectrum Spatial Analyst extensibility platform divides the entire layout of the product into different parts called containers. Containers are parents to the new components created by developers. Containers let a developer place their visual or non-visual components in the right place. For example, if a developer wants to place a component in the Add Panel, then he needs to specify AddPanel as a parent container for the newly created component. There is no limitation to the number of components that can be added to a given container. The look, feel, and the CSS of a new component can control the position of the component in a given container.

There can be cases where a single container is hosting more than one third-party component. For example, a “Find XY” and “Add WMTS layer” menu item can both be added to AddPanel. It is perfectly valid to specify the same parent container as many times as needed with different components. Components can belong to different modules as well. In that case, entries for the same parent container will be repeated in each module entry. The following sample illustrates this, where there are components in the same module and same container:

```
{
"name": "DynamicModule",
"description": "Find a defined x and y with a specific ICON.",
"modulePath": "extensions/dynamic.module.ts#DynamicModule",
"components": [
{
"componentName": "FindXY", "parentComponentName": "AddPanel",
},
{
"componentName": "AddWMTSLAYER", "parentComponentName": "AddPanel"
}
}
,
"componentsToRemove":[
{"componentName":"BaseMapSwitcherComponent",
"fromMapConfig":"Drive Time"},
{"componentName":"MapConfigSwitcherComponent",
" fromMapConfig":"Drive Time"}

]}
}
```

6 - Advanced Configuration Option

Spectrum Spatial Analyst is an extensive web mapping platform that developers can customize. Its framework is based on the Angular 4.2.6 component model, which lets you create and add new components to Spectrum Spatial Analyst or alter the behavior of existing components. This document describes how to extend Spectrum Spatial Analyst, explains related concepts, and gives examples. Read this guide along with the extensibility API documentation.

In this section

Referencing Third-party Libraries

The Spectrum Spatial Analyst Extensibility Platform envisages cases where new components may need to reference third-party external libraries. These libraries can be either Angular or normal JavaScript libraries. The Spectrum Spatial Analyst Extensibility Platform facilitates the on-boarding of such libraries with ease. To use new libraries in the component, follow the steps given below. Libraries can be references from the file system of the Spectrum Spatial Analyst server or can be referenced from a hosting site/CDN. The mechanism for registering the library is the same in both cases. There are certain restrictions that the Spectrum Spatial Analyst Extensibility Platform has while embedding a new library.

1. Only one version of a new library needs to be embedded.
2. If the library is already available with a certain version, one cannot embed a new version of that library. We provide a list of libraries available out-of-the-box within Spectrum Spatial Analyst via the module config validator page.
3. Checking for license terms, vulnerability, and certification of new libraries (libs) in the Spectrum Spatial Analyst Extensibility Platform is the responsibility of the component developer.
4. If someone intentionally violates point 1 and more than one version of the same library is added to the Spectrum Spatial Analyst Extensibility Platform, it cannot guarantee deterministic behavior.
5. If one module is embedding a specific version of a library, then another module cannot embed another version of the same library.
6. If one module is embedding a version of a library, that library can be used across multiple modules/components without repeating the same library in the other modules.

Specifying Map Configuration that Shows Components

The Spectrum Spatial Analyst Extensibility Platform supports the conditional rendering of newly added components based on the map project being used at that time. Consider a scenario that a component developer creates a component that should be available to only users when they browse to a specific map configuration. To achieve this, a user will create an entry in the `CustomAnalystModuleConfig.json` file and register a component for the specific map project(s). If there is more than one third-party component to be shown for a given map configuration, they can all be added as an array corresponding to the map project. Please note that the component name mentioned in the components tag should be the name of an Angular component class that is created.

Map projects associated to a third-party component should be present in allowlisting:

- If you mention a component for a map project, then that component is visible only to that map project and not others.
- To make a component available to more than one map projects but not for all of them, then allowlist the component in all map projects.
- To make a component available in all map projects, do not provide any entry in the mapconfigAssociated tag. For example:

```
"mapConfigAssociated":[{
 "mapConfigName":"defaultmap",
 "components":["TestComponent"]
 }]
```

Adding Custom Components under the Parent Components

While injecting custom components, the extensibility developers can:

- add new injection points on the left and right panels
- apecify the order of the custom component inside parent component

Note: Along with the new enhancements, the existing injection points and approach will continue to work in the same manner as it has been.

Explore new locations - add new injection points on the left and right panels

To explore new injection points, load the Spectrum Spatial Analyst in browser and inspect the parent element under which you want to insert the custom component by right clicking on that element and choosing inspect option. It opens the developer tool bar in the browser, then we need to copy the `id` of the element and put it as parentComponentName inside `CustomAnalystModuleConfig.json` file. Apart from this approach of providing the `id` of the parent element, you can also use class of the parent element or can even directly supply CSS selector for the parent element.

Note: It is recommended to use the `id` or CSS selector approach as class approach can lead to insertion of custom component at multiple places because class is not unique.

Specify the order of the custom component inside parent component

As explained above, specifying parent component in either of the approach always inserts a custom component as the last child of the parent component. But now, you can specify the position of custom component among the child elements of the parent component. For that we have introduced a new option called “childPosition” just below the “parentComponentName” option inside the CustomAnalystModuleConfig.json file.

```
{
"parentComponentName": "LegendContainer",
“childPosition” : 1
}
```

In the above childPosition is 1 which means the custom component when injected inside Spectrum Spatial Analyst, will be positioned as the first child element of LegendContainer. Similarly, if specified as 2, it will be positioned as the second child element and so on.

Spectrum Spatial Analyst allows you to add custom components. You can develop your extensions as custom components and add them where they are needed in the application. To do so, you need to register your extensions in the `CustomAnalystModuleConfig.json` file. You can find this file at: `<Analyst_install_directory>\Program Files\Precisely\SpectrumSpatialAnalyst \customerconfigurations\analyst\CustomAnalystModuleConfig.json.` After registering your custom components, when Spectrum Spatial Analyst reloads, it will load the extensible components as per the configurations in the `json` file.

In the `CustomAnalystModuleConfig.json` file, you need to specify the following information:

- path of the extensions
- the map project to which you want to associate your extension with, and
- the injection point such as add panel, left panel, or Legend where you want to insert the component

This feature enables you to add the component anywhere. Moreover, this feature will not disrupt any existing custom component because it is backward compatible. Therefore, any existing custom component will continue to work as expected.

For a list of parent components, see **Extension points** on page 63.

Adding a Custom Component at a Specific Position

You can add a component at a specific position in Setting, Add Layer, or the Legend panel and control the order of existing options in these panels using CSS to specify the order of elements. To do this, you set the order of components, including custom components, in the map project-specific brand.css file:
`<Analyst_install_directory>\customerconfigurations\analyst\theme\branding\default\brand.css`.
By default, brand.css includes sample entries.

The following example shows how you can insert a custom component in the Setting Panel just below the Print option.

```
   #createPanelContainer{
   display: flex;
   flex-direction: column;
   }
   #addLayerContainer{order: 10; }
   #addNewRecord{order: 20; }
   #create_QueryContainer{order: 30; }
   #createThematicContainer{order: 40; }
   #annotationToolContainer{order: 50; }
   #measurementToolContainer{order: 60; }
   #multiSelectContainer{order: 70; }
   #settingsPanelContainer{
   display: flex;
   flex-direction: column;
   }
   #printContainer {order: 10; }

   #imageExporterContainer {order: 20; }
   #currentMapViewContainer {order: 30; }
   #helpContainer {order: 40; }
   #localeContainer {order: 50; }
   #templateDesignerContainer {order: 60; }
   #mapProjectContainer {order: 70; }
   #authBtnContainer {order: 80; }
   #appVersionContainer {order: 90; }
   #settingsPanelContainer > <CUSTOM_COMPONENT_ELEMENT_NAME1> {order: 11;
  }
    #settingsPanelContainer > <CUSTOM_COMPONENT_ELEMENT_NAME2< {order:
 61; }
     #layersPanelContainer {
     display: flex;
     flex-direction: column;
     }
     #mapProjectSwitcherContainer{order: 10; }
     #baseMapSwitcherContainer{order: 20; }
     #legendContainer{order: 30; }
     #layersPanelContainer > <CUSTOM_COMPONENT_ELEMENT_NAME< {order: 11;
  }
```

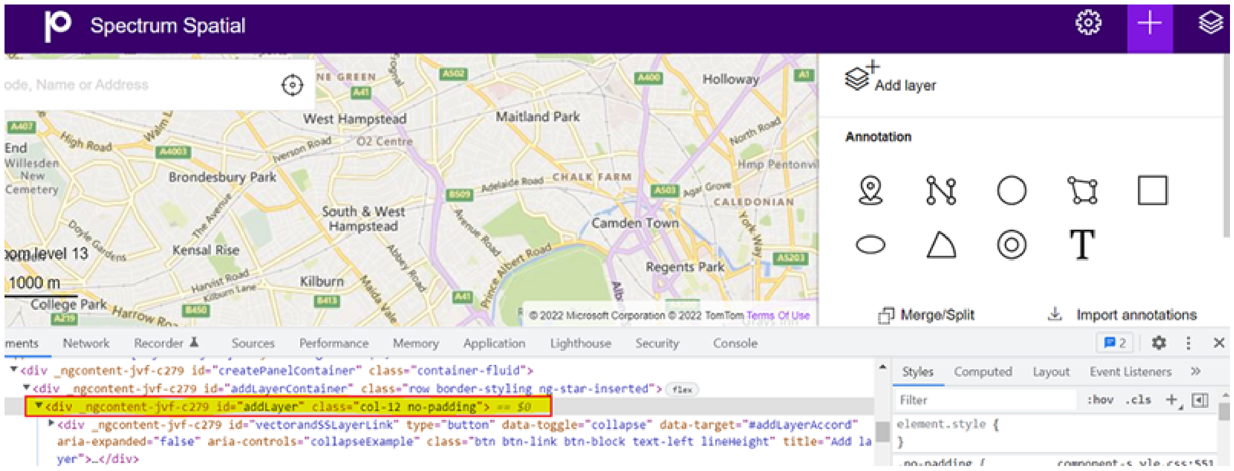
The ‘order’ attribute specified for an element determines the order in which it appears. Elements without a specified order display at the top in the panel.

You must replace the <CUSTOM_COMPONENT_ELEMENT_NAME1> placeholder with the corresponding custom component’s selector name. The order applies to all projects using the corresponding brand. We recommend creating a new branding file instead of editing the default brand.css file.

Example

In the following example, we will demonstrate how to insert an extensible component below the 'Add Layer' link on the Add panel.

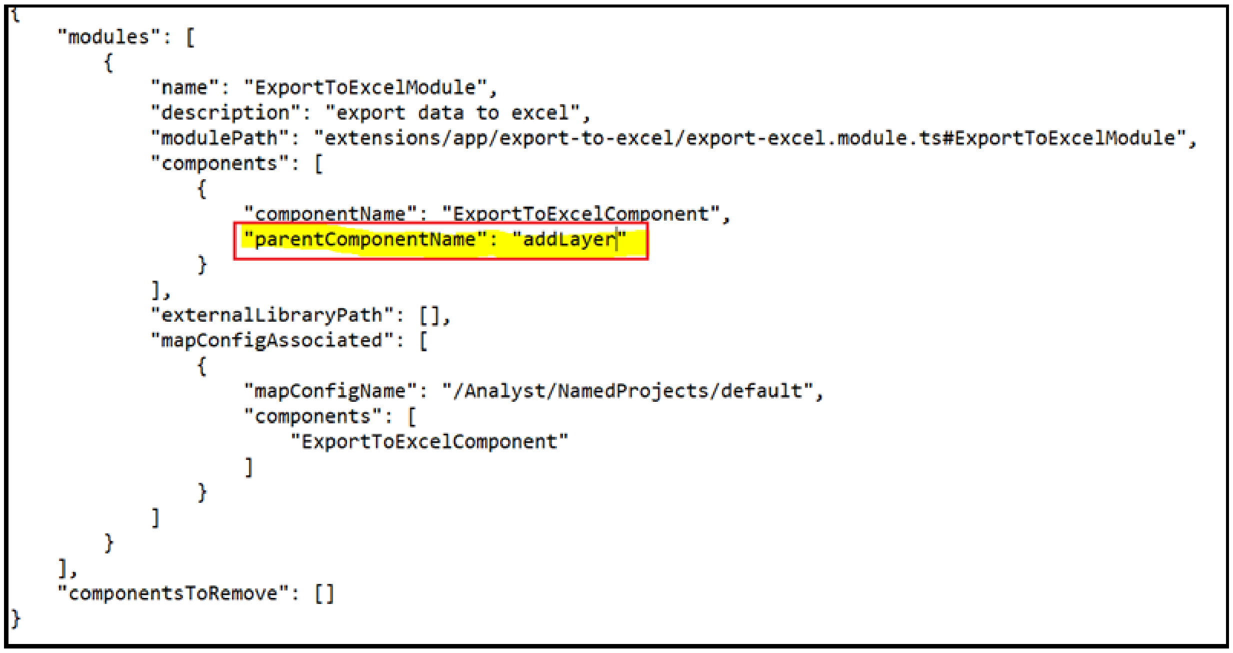
1. Start with inspecting the ‘Add Layer’ link in the browser's developer tools by right clicking it. For example, if you are using **Chrome DevTools**, when the DevTools opens, copy the id as shown below. In this example, the id is addLayer.



```
▼<div _ngcontent-jvf-c279 id="createPanelContainer" class="container-fluid">
  ▼<div _ngcontent-jvf-c279 id="addLayerContainer" class="row border-styling ng-star-inserted"> flex
    ▼<div _ngcontent-jvf-c279 id="addLayer" class="col-12 no-padding"> == $0
      ▶<div _ngcontent-jvf-c279 id="vectorandSSLayerLink" type="button" data-toggle="collapse" data-target="#addLayerAccord"
       aria-expanded="false" aria-controls="collapseExample" class="btn btn-link btn-block text-left lineHeight" title="Add la
       yer">…</div>
```

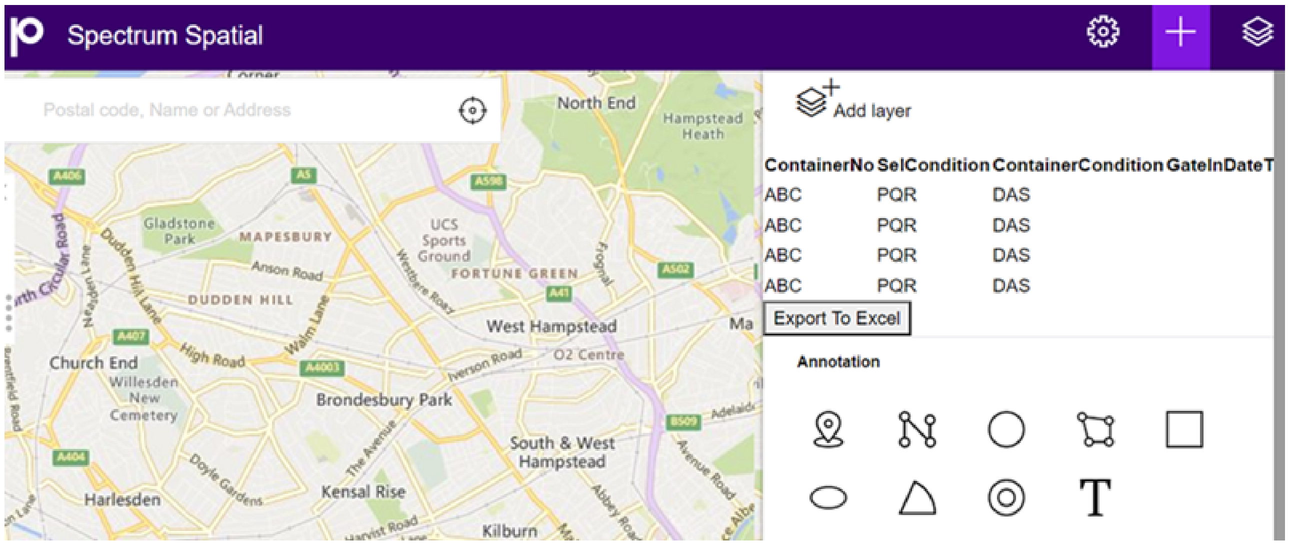
In the screenshot, we can see there are other ids like addLayerContainer for the elements which are parent to ‘Add Layer’ link. We can use those id as well. So, we can copy either addLayer or addLayerContainer from the dev tool. Let’s assume we use the id as addLayer for this sample.

2. Paste the id that we copied in Step 1 as parentComponentName inside the `CustomAnalystModuleConfig.json` file as highlighted in yellow.



```
{
    "modules": [
        {
            "name": "ExportToExcelModule",
            "description": "export data to excel",
            "modulePath": "extensions/app/export-to-excel/export-excel.module.ts#ExportToExcelModule",
            "components": [
                {
                    "componentName": "ExportToExcelComponent",
                    "parentComponentName": "addLayer"
                }
            ],
            "externalLibraryPath": [],
            "mapConfigAssociated": [
                {
                    "mapConfigName": "/Analyst/NamedProjects/default",
                    "components": [
                        "ExportToExcelComponent"
                    ]
                }
            ]
        }
    ],
    "componentsToRemove": []
}
```

3. Open the SSA application and then open the Add panel. We will see the extensible component getting injected below the ‘Add Layer’ link as shown below.



ContainerNo	SelCondition	ContainerCondition	GateInDateT
ABC	PQR	DAS	
ABC	PQR	DAS	
ABC	PQR	DAS	
ABC	PQR	DAS	

Removing Existing Spectrum Spatial Analyst Components

Current users of Spectrum Spatial Analyst have use-cases where they need to replace entire components of Spectrum Spatial Analyst with custom components. One use-case is the address search box that Spectrum Spatial Analyst provides. Another use-case is when a client wants a different base map switching capability instead of a drop-down. The Spectrum Spatial Analyst Extensibility Platform supports the replacement of components in two stages. In the first stage, a component developer removes the existing component from the Spectrum Spatial Analyst Extensibility Platform and in the second stage, they introduce a new typescript based angular component in its place.

Below is a list of components that can be removed from the Spectrum Spatial Analyst extensibility platform. It depends on your needs if you want to introduce a new component or remove it.

Note: If a parent component is removed, then its child component is also removed automatically.

To remove a component, a developer needs to mention the component in `CustomAnalystModuleConfig.json` file shown below.

```
"   componentsToRemove":[
{"componentName":"BaseMapSwitcherComponent"}]
```

Note: When removing a component, such as the left panel, ensure it is not a parent container of any third-party component.

Map Project-based Component Removal

There may be certain cases when a developer wants to remove components in certain conditions only. In this case, all users need to create a map project and configure the component to remove in the `CustomAnalystModuleConfig.json` file. A typical entry in CustomAnalystModuleConfig.json would look like this:

```
"componentsToRemove":[
 {"componentName":"BaseMapSwitcherComponent",
 "fromMapConfig":"Drive Time"}]
```

When removing the same component from more than one map project, the entry for the component repeats for each map project.

Removing Components using Config File

Component Name	Identifier	Remove Only via config file
Base Map Switcher Component	BaseMapSwitcherContainer	Yes
Map Config Switcher Component	MapConfigSwitcherContainer	Yes
Left Panel Component	LeftPanelContainer	Yes
Query Results Component	QueryResultsComponent	Yes
Callout Container Component	CalloutContainerComponent	No (Via Adminconsole as well)
Search Box Component	SearchBoxContainer	Yes

Component Name	Identifier	Remove Only via config file
Summarization Results Component	SummarizationComponent	No (Via Adminconsole as well)
Legend Container Component	LegendContainerComponent	Yes
Summarization Component	SummarizationComponent	No (Via Adminconsole as well)

Removing Components using Functionality Profile Settings

Spectrum Spatial Analyst also supports the removal of components via a functionality profile. Depending on the use case, you can choose to remove some common components via functionality profile. The list of components is:

- Query
- Annotations
- Summarize Data in Annotations
- Measuring Tools
- Annotations Tools
- Annotation KML Import/Export
- Print
- End-User Thematics
- Add Layer
- Summarization Component
- Editing in Tables

Component Context Parameter

The Spectrum Spatial Analyst extensibility platform provides support for passing in context parameters from dynamic container components to its child components, including child components created by developers. Developers can use the data as per their needs to adjust the logic of the components they create. For example, when a user draws a circle annotation, a developer creates a custom component to query within the circle annotation. The Spectrum Spatial Analyst Extensibility Platform passes in all the information about circle annotation to the custom component like radius, XY location, name of annotation and so on.

Context data may be useful for passing the information to external systems or it can be used to make the component rendering exclusive for an instance of the dynamic container. For example, if there are more than one circle annotation and developer wants to show the component for the first circle annotation only then he can use the annotation name from the context parameter to restrict the view of the new component in its template.

To access this context data, a component developer needs to create an input field with name data: any in its own created typescript component. Inside this data field, each of the dynamic containers has a specific name for context parameters; for example, the annotationLegendItem context parameter name is annotationLegendObject. The following table provides the names of all the context data parameters that are available for different dynamic components:

Component Name	Context parameter Name (For example, data. annotationLegendObject)
AnnotationLegendItem	annoationLegendObject
EnvinsaTileLegendItem	legendGroupObject
MVTLegendItem	legendGroupObject
AnnotationLegendGroupItem	annoationGroupObject
QueryLegendItem	queryLegendObject
SpatialLegendItem	legendGroupObject
SpatialSubLegendItem	legendObject
ThematicLegendItem	legendGroupObject
TMSLegendItem	legendGroupObject
VectorLayerLegendItem	vectorLayerLegendObject
WMSLegendItem	legendGroupObject
XYZLegendItem	legendGroupObject

Component Name	Context parameter Name (For example, data. annotationLegendObject)
CalloutCardContainer	calloutObject
CalloutContainer	calloutRecordObject

Component Initialization Parameter

The Spectrum Spatial Analyst extensibility platform envisages cases where more than one instance of newly created angular 4 third-party components need to be on-boarded. There can be cases where multiple instances of the new component may need to share the same set of information. For example, a developer may create a new component that shows Google Street View that is shown in multiple instances and needs to share the API key for Google between them. The Spectrum Spatial Analyst extensibility platform supports parameter sharing using the init parameters among multiple component instances. To get access to the init parameter, the developer needs to:

1. Declare an input field called data in its component.
2. Add an entry corresponding to the component in `CustomAnalystModuleConfig.json`. For example:

```
 "modules": [{
 "name": "GIQueryModule",
 Spectrum Spatial Analyst Extensibility Guide v.2019.1 26
 "description": "Module For GI query",
"modulePath": "../../../extensions/dynamic.module.ts#DynamicModule",

 "components":[{componentName:"TestComponent","parentComponentName":
"SettingsPanel",
 "initParameters": {
 "apiKey": “abcdef”
 }
 }}]
 }]
```

3. Once this is declared in CustomAnalystModuldeConfig.json one can access the key like data.initParameters.apiKey in the component instance.

The init parameter supports the data types that JavaScript supports. It does not put restrictions on the size of the parameters supplied.

Component that Can Run at Startup

The Spectrum Spatial Analyst extensibility platform supports running components that are required during startup time (when a user first opens the Spectrum Spatial Analyst application in the browser). This can be achieved if the component is injected into a parent that comes into existence during startup. The RightPanel is one such parent container. To make a component available at startup, declare the RightPanel as its parent. The component then comes into existence at startup.

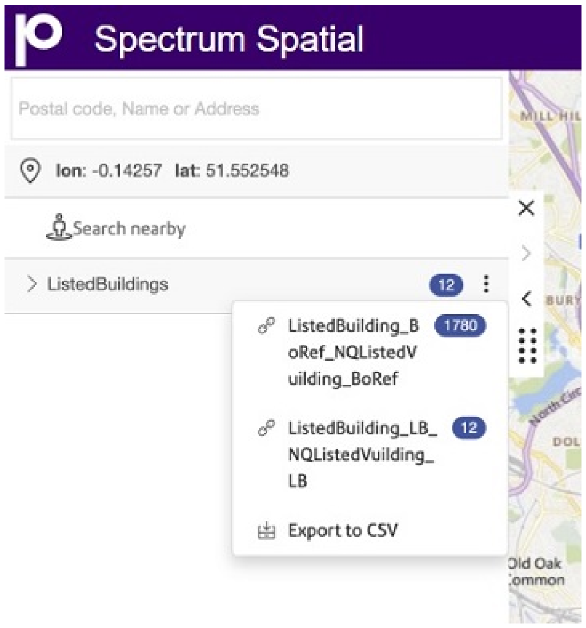
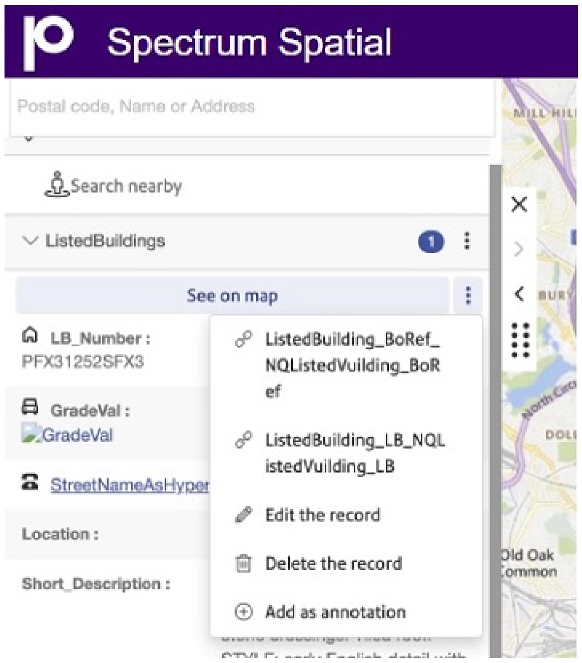
Component Without HTML

The Spectrum Spatial Analyst extensibility platform supports components having pure business logic and no visual elements. As such all Angular components support capability to embed HTML in them but it is optional. To create a component without HTML, keep the template blank, and the Spectrum Spatial Analyst Extensibility Platform calls the component at the time of instantiating its parent container. For example, a developer creates a component that gets weather data from a remote API and passes this on to some other component for further processing. Let's assume the component developer makes it a child of the AddPanel container. When a user clicks on the AddPanel in Spectrum Spatial Analyst, the third-party component calls the child and it then makes a call to get the weather data.

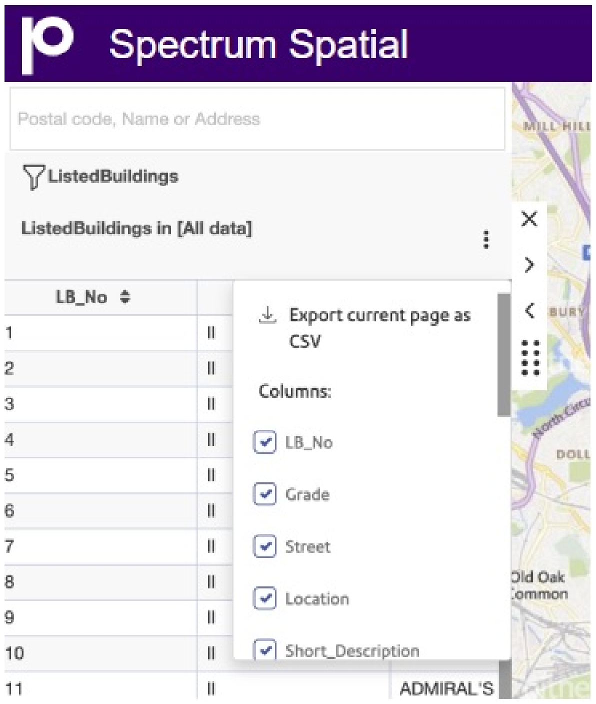
Reordering Left Hand Panel Menu

You can reorder the menu items in the left-hand panel at three levels.

- Layer level
- Record level
- Tabular grid's overflow menu

Layer level	
Record level	

Spectrum Spatial
Postal code, Name or Address
lon: -0.14257 lat: 51.552548
Search nearby
ListedBuildings 12
ListedBuilding_BoRef_NQListedVuilding_BoRef 1780
ListedBuilding_LB_NQListedVuilding_LB 12
Export to CSV

Tabular grid's overflow menu	

The layer and record levels use the same overflow menu components. The following CSS example re-orders menu items for the layer and record levels.

```
    .open>#overflowMenu{
    display: flex;
    flex-direction: column;
    }
    .exportAsCsv{
    order: 10;
    }
    .addAsAnnotation{
    order: 20;
    }
    .showOnMap{
    order: 30;
    }
    .editRecord{
    order: 40;
    }
    .deleteRecord{
    order: 50;
    }
    .dataBindTitle{
    order: 60;
    }
    .linkouts{
    order: 70;
    }
    /* Please specify selector of custom component to specify its order*/

    #overflowMenu>CUSTOM-ELEMENT-NAME{
    order: 30;
    }
```

This CSS example re-orders menu items in the tabular grid's overflow menu:

```
.open>#gridOverflowMenuContent{
   display: flex;
   flex-direction: column;
   }
   .linkoutQryGrd{
   order: 10;
   }
   .exportCurrentPageQryGrd{
   order: 20;
   }
   .exportAllDataQryGrd{
   order: 30;
   }
   .columnsQryGrid{
   order: 40;
   }
   .columnNamesQryGrid{
   order: 50;
   }
```

To achieve this, refactor the overflow menu component. If a custom component uses injection points, such as QueryResultsItem, CalloutCardContainer, or CalloutContainer, then adapt the component code as follows:

```
@Component({
 selector: 'my-CallOutCardContainer',
 template: `
 <pan class="btn btn-link text-left btn-block container-flex">
 </i>CallOutCardContainer
 </span>
 `,
 })
 @Component({
 selector: 'my-CallOutCardContainer',
 template: `
 <li class="ellipsesDropdown">
 <span class="btn btn-link text-left btn-block container-flex">
  <i _ngcontent-c46="" class="nc-icon-outline ui-1_trash-simple margin-r
 overflow-imgMargin">
  </i>CallOutCardContainer
 </span>
 </li>
`,
})
```

Example - Adding Custom Component in Overflow Menu

This example adds a custom component at the second position in a layer's information overflow menu.

1. Configure an extensible component at the CalloutContainer injection point.
2. Uncomment the following CSS in the brand.css file.

```
.open>#overflowMenu{
 display: flex;
 flex-direction: column;
 }
 .exportAsCsv{
 order: 10;
 }
 .addAsAnnotation{
 order: 20;
 }
 .showOnMap{
 order: 30;
 }
 .editRecord{
 order: 40;
 }
 .deleteRecord{
 order: 50;
 }
 .dataBindTitle{
 order: 60;
 }
 .linkouts{
 order: 70;

 }
 /* Please replace CUSTOM-ELEMENT-NAME with the selector of custom
component to specify its order*/
 #overflowMenu>CUSTOM-ELEMENT-NAME{
 order: 30;
 }
```

3. Replace the CUSTOM-ELEMENT-NAME with the selector of the custom component. For example, `my-CallOutContainer` is the selector in the following component:

```
@Component({
 selector: 'my-CallOutContainer',
 template: `
 <li class="ellipsesDropdown">
  <span class="btn btn-link text-left btn-block container-flex">
  <i _ngcontent-c46="" class="nc-icon-outline ui-1_trash-simple margin-r
 overflow-imgMargin">
  </i>CallOutCardContainer
  </span>
 </li>
 `,
 })
```

The entry in the brand.css file for my-CallOutContainer will be something like this:

```
/* Please replace CUSTOM-ELEMENT-NAME with the selector of custom
component to specify its order*/
   #overflowMenu>CUSTOM-ELEMENT-NAME{
   order: 30;
   }
```

After replacing the CUSTOM-ELEMENT-NAME to my-CallOutContainer, the entry looks like this:

```
/* Please replace CUSTOM-ELEMENT-NAME with the selector of custom
component to specify its order*/
   #overflowMenu>my-CallOutContainer'{
   order: 11;
   }
```

Also update the order property from 30 to 11, so that it is visible after the user selects the Export to CSV option in Spectrum Spatial Analyst.

7 - Validating Your Components and Third-party Libraries

This section discusses how to validate your components and available third-party libraries in the Spectrum Spatial Analyst Extensibility Platform.

In this section

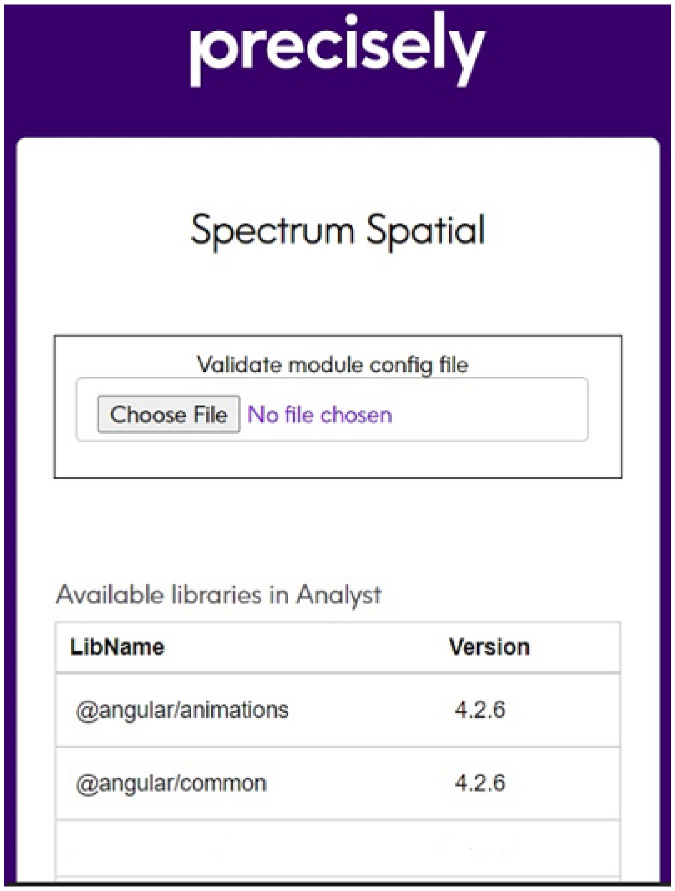
Overview

This section discusses how to validate your components and available third-party libraries in the Spectrum Spatial Analyst Extensibility platform.

Module Config Validator

Since the module configuration can become complex if many modules are added to it, the Spectrum Spatial Analyst extensibility platform provides a module config validator that can be used to validate the `CustomAnalystModuleConfig.json` file. The tool is simple to use, browse to it as follows:

1. Browse to `http://<InstallationURL>:8010/connect/analyst/mobile/#/customModuleValidation`. The following screen displays.



Available libraries in Analyst

LibName	Version
@angular/animations	4.2.6
@angular/common	4.2.6

2. Choose and browse to a manually authored `CustomAnalystModuleConfig.json` file on the file system. The tool will validate the file. It also shows any errors to the user
3. Once validation is complete, place the `CustomAnalystModuleConfig.json` file into the customerconfigurations folder of the Spectrum Spatial Analyst installation.

The ModuleConfig validator performs semantic and syntactic validation of the `CustomAnalystModuleConfig.json`, and returns any errors for missing braces, un-equal parenthesis, missing mandatory fields, and so on. It also validates the location of modules and external JavaScript files in the `CustomAnalystModuleConfig.json` file. We recommend that you check the `CustomAnalystModuleConfig.json` file is valid before deploying it to Spectrum Spatial Analyst in the customerconfigurations folder. Please refer to the tables for configuring the `CustomAnalystModuleConfig.json` to check for mandatory and optional fields in the `CustomAnalystModuleConfig.json` file.

Third-party Libraries and Versions

The module config validator page also provides a list of libraries that are part of the Spectrum Spatial Analyst Extensibility Platform and its corresponding version. It is important to go over the list if you want to onboard a new library in the Spectrum Spatial Analyst Extensibility Platform. If a library is available in the Spectrum Spatial Analyst Extensibility Platform, you can use it directly in your component.

Note: You cannot include two versions of the same library. If you need some feature that is available in a later version of a library, you should request this in the next version of Spectrum Spatial Analyst via tech support. If the library is not available in the list, you are free to onboard it as described in “Referencing Third-party Libraries” section.

Branding Third-party Components

The Spectrum Spatial Analyst extensibility platform provides comprehensive branding/styling as part of the brand CSS facility to customize the look and feel of Spectrum Spatial Analyst. Different brands can be created and referenced in map projects via the map project settings. A component developer can reference all of the branding classes in the CSS to adjust the look and feel of their components as per the branding guidelines of Spectrum Spatial Analyst.

Path Restriction for Module or Components

It is recommended to put the components in the extensions folder in the <ANALYST_INSTALL_PATH>/customerconfigurations/analyst/theme/extensions. It helps the Spectrum Spatial Analyst installer in backing up and restoring these extensions during upgrades. It ensures that during installation and upgrades, your extensions are not lost. Paths mentioned in the `CustomAnalystModuleConfig.json` are static.

8 - Referencing Spectrum Spatial Analyst APIs in your Component

In this section

Use of Existing Services Store Actions, Selectors, and Components

Use of Existing Services, Store Actions, Selectors, and Components.

This section describes how to use the existing services, store actions, selectors, and components of the Spectrum Spatial Analyst Extensibility Platform.

NgRx Store

NgRx store is the base architecture for the Spectrum Spatial Analyst Extensibility Platform. It is the primary mechanism to consume the resources of the Spectrum Spatial Analyst Extensibility Platform in third-party components via the use of store actions and selectors.

For example, if you want to listen to the map click event in your custom component you will use a selector to achieve this. Similarly, if you want to add a new layer to the map from your custom component you will dispatch an action for this. For the store to be used in a component you need to add it as a parameter in the component's constructor. We have provided various examples to showcase how it can be used.

The real power of the Spectrum Spatial Analyst Extensibility Platform comes from re-using many services, components and third-party libraries that come out of the box with Spectrum Spatial Analyst. For example, you can utilize the extensive set of services the platform exposes for the query features by SQL, at an XY or within a user drawn region, rather than coding this into your component. There are also multiple sets of utilities exposed via the Spectrum Spatial Analyst Extensibility Platform. A complete list of these can be found in the API docs shipped with Spectrum Spatial Analyst installation.

Though API docs list all of the functionality in Spectrum Spatial Analyst components, developers are advised to only use the public APIs or components of the Spectrum Spatial Analyst extensibility platform. This is necessary to maintain backward compatibility. Backward compatibility is not guaranteed when using private API, which is subject to change without notice.

Openlayers

Most third-party components are likely to need to interact with the Openlayers library that is available out of the box with Spectrum Spatial Analyst. This may be to capture user interaction (such as map clicks, drawing, or feature selection) or to add layers and features to the map or to move/zoom the map. While it is entirely feasible to reference the map object and to interface directly with Openlayers, in most circumstances. We recommend doing this via store actions. The use of store actions ensures that the state is consistently maintained between the map and the legend panel, where actions on one affect the other.

Useful Links

- https://angular.io/tutorial
- https://angular.io/api
- https://v2.angular.io/docs/js/latest/cookbook
- https://github.com/angular/quickstart
- https://code.visualstudio.com/download

9 - Gloassary of Terms

In this section

List of Terms

Component

Component This is an Angular 4 component written in typescript (TS) and provided to the Spectrum Spatial Analyst Extensibility Platform to embed at run time.

Spectrum Spatial Analyst Extensibility Platform

The architecture of Spectrum Spatial Analyst that allows third-party components are dynamically added at run-time and the set of core services that Spectrum Spatial Analyst provides for re-use by embedded components.

SystemJS

This is the main module that Spectrum Spatial Analyst uses to boot-strap and enable run-time embedding of third-party components in Spectrum Spatial Analyst.

CustomModulesDefinition.json

Configuration file configuring the definition of third-party components.

Module

The smallest unit of third-party code that the Spectrum Spatial Analyst Extensibility Platform can embed. A module is an Angular 4 module of one or more components mentioned in point 1.

Store

Spectrum Spatial Analyst Extensibility Platform's front end architecture is based on the NgRx store. Most of the services that a developer can use are available via Store actions and their corresponding selectors. The Store is a bridge between the caller and executor. For example, if a third-party developer wants to draw a layer on a map, then they dispatch an action with the necessary parameters such as layerUrl/extents/center and so on via store.dispatch.

A - Appendix

In this section

List of changes in Action payloads and Action names

Below is the list of actions whose payload have changed after upgrading the NgRx Store.

Add Layer

All actions updated in the module are listed below.

Action Name	Updated payload
AddSSNamedLayerAction	type added { namedLayers: NamedLayerData[], repositoryPaths: string[] }

Annotations

No change in action creation except removing the `new` operator.

Databind

All actions updated in the module are listed below.

Action Name	Updated payload
DataBindExecuteQueryAction	type added {url:string}
DataBindExecuteQueryFailureAction	input/payload removed
RemoveDataBindAction	input/payload removed
GetAllDataBindListAction	input/payload removed
UpdateDataBindListAction	type added {databindList: DataBindList}
DatabindResultAction	Action added. Populates results from a databind join. Previously only present as an action type string.

Find My Nearest

All actions updated in the module are listed below.

Action Name	Updated payload
ClearFmnStateAction	Removed input/payload

Action Name	Updated payload
UpdateFmnStateAction	Added FmnState type

Layer Information

All actions updated in the module are listed below.

Action Name	Updated payload
ClearFeatureInfoAction	removed input/payload
ResetCalloutFeaturesAction	removed input/payload

Legend Module

All actions updated in the module are listed below.

Action Name	Updated payload
UpdateLegendGroupAction	payload changed from any to LegendGroup

Login

All actions updated in the module are listed below.

Action Name	Updated payload

GetEditPermissionTableListAction	**Before upgrade**:
	this.store.dispatch(new GetEditPermissionTableListAction(tablesList));
	After upgrade:
	this.store.dispatch(GetEditPermissionTableListAction());

StoreFeatureEditTemplateMappingsAction	**Before upgrade**:
	this.store.dispatch(new StoreFeatureEditTemplateMappingsAction(payload));
	After upgrade:
	this.store.dispatch(StoreFeatureEditTemplateMappingsAction({ payload: EditTemplateMappingObject }));

FeatureUpdateAction	**Before upgrade**: this.store.dispatch(new FeatureUpdateAction(payload)); **After upgrade**:

Action Name	Updated payload
	this.store.dispatch(FeatureUpdateAction({ payload: FeatureEditableTableObject}));

Map

All actions updated in the module are listed below.

Action Name	Updated payload
RemoveFeatureFromMapAction	type added {layerName: string, prefix: string, type?: string}
AddFeatureToMapAction	type added { feature: GeometryObject, id: number | string, featureName: string, shouldFeatureBeZoomed: boolean }
HighlightFeaturesAction	type added { featuresToHighlight?: string[], annotationsDrawn?: string[], operation?: string }
CloseOverlayAction	input/payload removed
MoveCopyRightInformationAction	input/payload removed
UpdateLegendAction	type added SpatialLegendItem | AnnotationObject | VectorLegendInput
UpdateLabelParamsAction	type added { name: string, type: string, updatedLabelSources: LabelSource[] }
UpdateLayersParamsAction	type added { name: string, type: string, color?: string, layerList?: string[], params?:Unknown macro: { queryResultMapJson}}
RegisterViewEventAction	type added {event: string, callback: Function }
ChangeLayerOpacityAction	type added {layerName: string, opacity: number}
ToggleLayerVisibilityAction	type added { layerName: string, visible: boolean, type?: string, repositoryPath?: string }
AddLayerAction	type added MapLayer | VectorLayerOptions | VectorLayerVariant | RangeThematicOptions
DeleteLayerAction	type added { name: string, type?: string }
UpdateLayersStateActionOnDelete	input type changed to string
TeleportMapInPrintContainerAction	input/payload removed
RefreshLayerAction	type added { layerName?: string, layerRepositoryPath:string }
UpdateMapSizeAction	input/payload removed

Action Name	Updated payload
UpdateGoogleMapSizeAction	type changed to { gMapHeight: number, gMapWidth: number }
ClearMapStateAction	input/payload removed
MapClickedAction	type changed to MapBrowserEvent
ClearMultiSelectFeatureGeometryAndData	input/payload removed
ClearMeasurementLayerAction	input/payload removed
ClearLegendLayersAction	input/payload removed

MapProject/MapConfig

All actions related to MapConfig module have been moved to MapProject module. You need to update all Import references related to MapConfig action.

Action Name	Updated payload
SaveProjectAction	type added { payload?: MapProjectUIState }
OpenMapProjectSettingsAction	type added { sName: string };
DataUpload	UpdateDataCompleteAction type added { payload: UploadDataDetails }
ResetUploadDataAction	input/payload removed

Marker Overlay

All actions updated in the module are listed below.

Action Name	Updated payload
ShowFmnMarkersAction	MarkerInformationFmn type added
StartFeatureHoverAction	New Action to turn on Feature Hover stream (now disabled by default)
StopFeatureHoverAction	New Action to turn of Feature Hover stream

Print Module

All actions updated in the module are listed below.

Action Name	Updated payload
EnablePrintModeAction	type of payload changed from { status: true } to (payload: EnableDisablePrintModePayload = new EnableDisablePrintModePayload(true))

EnableDisablePrintModePayload	class EnableDisablePrintModePayload {
	
	status: boolean = false;
	
	constructor(enable: boolean)
	
	{ this.status = enable; }
	
	}

DisablePrintModeAction	type changed from public payload = { status: false }; to (payload: EnableDisablePrintModePayload = new EnableDisablePrintModePayload(false))
LoadPrintTemplateConfigAction	type of payload has been changed from { file: string, type: string }) to LoadPrintTemplateConfig which has been defined as :- interface LoadPrintTemplateConfig { file: string; type: string; }
CustomeAnalystModuleConfig	no changes

Query

All actions updated in the Query module are listed below.

Action Name	Updated payload

SelectQueryAction	type added { queryName: string } : this.store.dispatch(SelectQueryAction({ queryName: 'MyQuery1'}));
	
	Before upgrade:
	
	this.store.dispatch(new SelectQueryAction(queryName));
	
	After upgrade:
	
	this.store.dispatch(SelectQueryAction({ queryName:legendGroup.name }));

DeleteQueryAction	type added { queryName: string }
	
	Before upgrade:
	
	this.store.dispatch(new DeleteQueryAction(queryName));
	
	After upgrade:

Action Name	Updated payload
	this.store.dispatch(DeleteQueryAction({ queryName: queryName }));

LoadPredefinedFiltersSuccessAction	type added { predefinedFilters:string : string[] }
	Before upgrade:
	this.store.dispatch(new LoadPredefinedFiltersSuccessAction(predefinedFilters));
	After upgrade:
	this.store.dispatch(LoadPredefinedFiltersSuccessAction({ predefinedFilters: filters }));

LoadVectorFiltersSuccessAction	type added { predefinedFilters: string[] }
	Before upgrade:
	this.store.dispatch(new LoadVectorFiltersSuccessAction(predefinedFilters));
	After upgrade:
	this.store.dispatch(LoadVectorFiltersSuccessAction({ predefinedFilters: filters }));

LoadAdminDefinedQueriesAction	type added { adminQueries: string[] }
	Before upgrade:
	this.store.dispatch(new LoadAdminDefinedQueriesAction(adminQueries));
	After upgrade:
	this.store.dispatch(LoadAdminDefinedQueriesAction({ adminQueries: adminQueries }));

SelectQueryFilterAction	type added { filterName: string }
	Before upgrade:
	this.store.dispatch(new SelectQueryFilterAction(filterName));
	After upgrade:
	this.store.dispatch(SelectQueryFilterAction({ filterName: filterName }));

ClearQueryStateAction	removed input/payload **Before upgrade:** this.store.dispatch(ClearQueryStateAction({}));

Action Name	Updated payload
	After upgrade: this.store.dispatch(ClearQueryStateAction());

Resource Selector

All actions updated in the module are listed below.

Action Name	Updated payload
UpdateSelectedResourcePath	**Before upgrade** : this.store.dispatch(new UpdateSelectedResourcePath(path)); **After upgrade** : this.store.dispatch(UpdateSelectedResourcePath({resourcePath: path}));
ToggleResourceSelectorVisibility	**Before upgrade** : this.store.dispatch(new ToggleResourceSelectorVisibility(true)); **After upgrade** : this.store.dispatch(ToggleResourceSelectorVisibility({toggleVisiblity: true}));
ToggleResourceSelectorLoading	**Before upgrade** : this.store.dispatch(new ToggleResourceSelectorLoading(true)); **After upgrade** : this.store.dispatch(ToggleResourceSelectorLoading({toggleLoading: true}));
UpdateResourceListData	**Before upgrade** : this.store.dispatch(new UpdateResourceListData(data[])); **After upgrade** : this.store.dispatch(UpdateResourceListData({listViewData: data[]}));;

Spatial Named Resources

All actions updated in the module are listed below.

Action Name	Updated payload
SetNamedMapDescriptionAction	type changed from object to DescribeNamedMapResponse

Action Name	Updated payload
SetNamedLayersDescriptionsAction	types added LayerDescription | DescriptionError)
SelectVectorTableAction	VectorTable type added to optional input
SelectVectorColumnAction	optional tableId?: string added
SetNamedTableDescriptionsAction	type changed from object to {tableRef: string, data: TableMetadata}
GetNamedTablesDescriptionAction	type changed from object to {[name: string]: TableMetadata}

Thematic

All actions updated in the module are listed below.

Action Name	Updated payload
GetThemeListAction	type added {tableRef: string; column: ThematicColumn;}
CompleteThemeListAction	type added string[]
GetGeometriesListAction	type added {tableRef: string;}
CreateIndividualThematicAction	has been deprecated see CreateThematicAction or RequestThematicLegendAction
RequestThematicLegendAction	added. Used to request a legend to be generated
ClearThematicStateAction	input/payload removed
ClearThematicLegendAction	input/payload removed
CreateThematicLegendAction	has been deprecated see CreateThematicAction or RequestThematicLegendAction
CreateThematicAction	Used to create a generic Thematic with all known details
CompleteThematicLegendAction	type added ThematicLegendData
AddThematicLayerAction	type changed to ThematicLayer
GetThematicLegendsAction	(ThematicLayer) included in type

User Authorization

All actions updated in the module are listed below.

Action Name	Updated payload
ClearUserAuthorizationStateAction	input/payload removed
UpdateNamedLayersListInAuthStateAction	type added { layerList: string[] }

Action Name	Updated payload
SetWritableFoldersForUserAction	type added { folders: string[] }
UpdateTileServiceProfilesForUserAction	type added { tileList: string[] }
UpdateWMSProfilesForUserAction	type added { wmsProfiles: string[] }
UpdateNamedTablesListInAuthStateAction	type added { namedTableList: string[] }
UpdateNamedMapsListForUserAction	type added { namedMapList: string[] }
UpdateNamedTilesListForUserAction	type added { namedTileList: string[] }
DescribeWMSProfileAction	type added { wmsName: string }
DescribeTileServiceProfileAction	type added { tileServiceName: string }

Left Panel Open-Layer map

All actions updated in the module are listed below.

Action Name	Updated payload
ClearMultiSelectFeaturesDataInitiatedAction	**Before upgrade**: this.store.dispatch(new ClearMultiSelectFeaturesDataInitiatedAction(payload)); **After upgrade**: this.store.dispatch(ClearMultiSelectFeaturesDataInitiatedAction({inputData: payload}));
ClearMultiSelectFeaturesDataCompletedAction	**Before upgrade**: this.store.dispatch(new ClearMultiSelectFeaturesDataCompletedAction(payload)); **After upgrade**: this.store.dispatch(ClearMultiSelectFeaturesDataCompletedAction({inputData: payload}));

Extension points

List of Parent Component Name

The following is a list of parentComponentName values in the Spectrum Spatial Analyst. You can add a child component inside them to extend and customize a component.

No.	parentComponentName Value	Description
1.	ADD_PANEL	Use this parent component to add a component inside the Add panel on the right of SSA main page.
2.	ANNOTATION_LEGEND_ITEM	Use this parent component to add component inside overflow menu for annotation item in legend panel.
3.	ANNOTATION_LEGEND_GROUP_ITEM	Use this parent component to add component inside the AnnotationLegendGroupItem.
4.	ANNOTATION_TOOL_CONTAINER	This will add a component inside the Annotation tool container.
5.	MEASUREMENT_TOOL_CONTAINER	This will add a component inside the MeasurementToolContainer
6.	BASE_MAP_SWITCHER_CONTAINER	This will add a component inside the BaseMapSwitcherContainer
7.	CALLOUT_CARD_CONTAINER	This will add a component inside the CalloutCardContainer
8.	CALLOUT_CONTAINER	This will add a component inside the Callout container.
9.	LAYER_PANEL	This will add a component inside the LayerPanel.
10.	LEGEND_CONTAINER	This will add a component inside the LegendContainer.
11.	LEFT_PANEL_CONTAINER	This will add a component inside the LeftPanelContainer.
12.	MAP_CONFIG_SWITCHER_CONTAINER	This will add a component inside the MapConfigSwitcherContainer.
13.	MVT_LEGEND_ITEM	This will add a component inside the MVTLegendItem.
14.	QUERY_LEGEND_ITEM	This will add a component inside the QueryLegendItem.
15.	QUERY_RESULTS_ITEM	This will add a component inside the QueryResultsItem.
16.	RIGHT_PANEL	This will add a component inside the RightPanel.
17.	RIGHT_PANEL_TOOLBAR	This will add a component inside the RightPanelToolbar.

No.	parentComponentName Value	Description
18.	SEARCH_BOX_CONTAINER	This will add a component inside the SearchBoxContainer.
19.	SETTINGS_PANEL	This will add a component inside the SettingsPanel.
20.	SPATIAL_LEGEND_ITEM	This will add a component inside the SpatialLegendItem.
21.	SPATIAL_SUB_LEGEND_ITEM	This will add a component inside the SpatialSubLegendItem.
22.	THEMATIC_LEGEND_ITEM	This will add a component inside the ThematicLegendItem.
23.	TMS_LEGEND_ITEM	This will add a component inside the TMSLegendItem.
24.	VECTOR_LAYER_LEGEND_ITEM	This will add a component inside the VectorLayerLegendItem.
25.	WMS_LEGEND_ITEM	This will add a component inside the WMSLegendItem.
26.	XYZ_LEGEND_ITEM	This will add a component inside the XYZLegendItem.
27.	CALLOUT_CONTAINER_COMPONENT	This will add a component inside the CalloutContainerComponent.
28.	LEGEND_CONTAINER_COMPONENT	This will add a component inside the LegendContainerComponent.
29.	DATA_CREATION_COMPONENT	This will add a component inside the DataCreationComponent.

Spectrum Spatial Analyst have been supporting these components from the previous releases and will continue to support them along with the new injection points.

Note: There are more injection points in addition to those listed here. The Spectrum Spatial Analyst team will continue to add more injection points in the future.