Spectrum Spatial Analyst
Extensibility User Guide

Version 2019.1

Contents

Introduction

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.

Overview

Spectrum Spatial Analyst lets you add Angular based components dynamically at run-time to an already deployed and running instance. It is not necessary to compile components or build a custom version of Spectrum Spatial Analyst.

Components are built as standard Angular 4.2.6 components (written in TypeScript) and are included in an Angular 4.2.6 module. An Angular 4.2.6 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.

Static Container

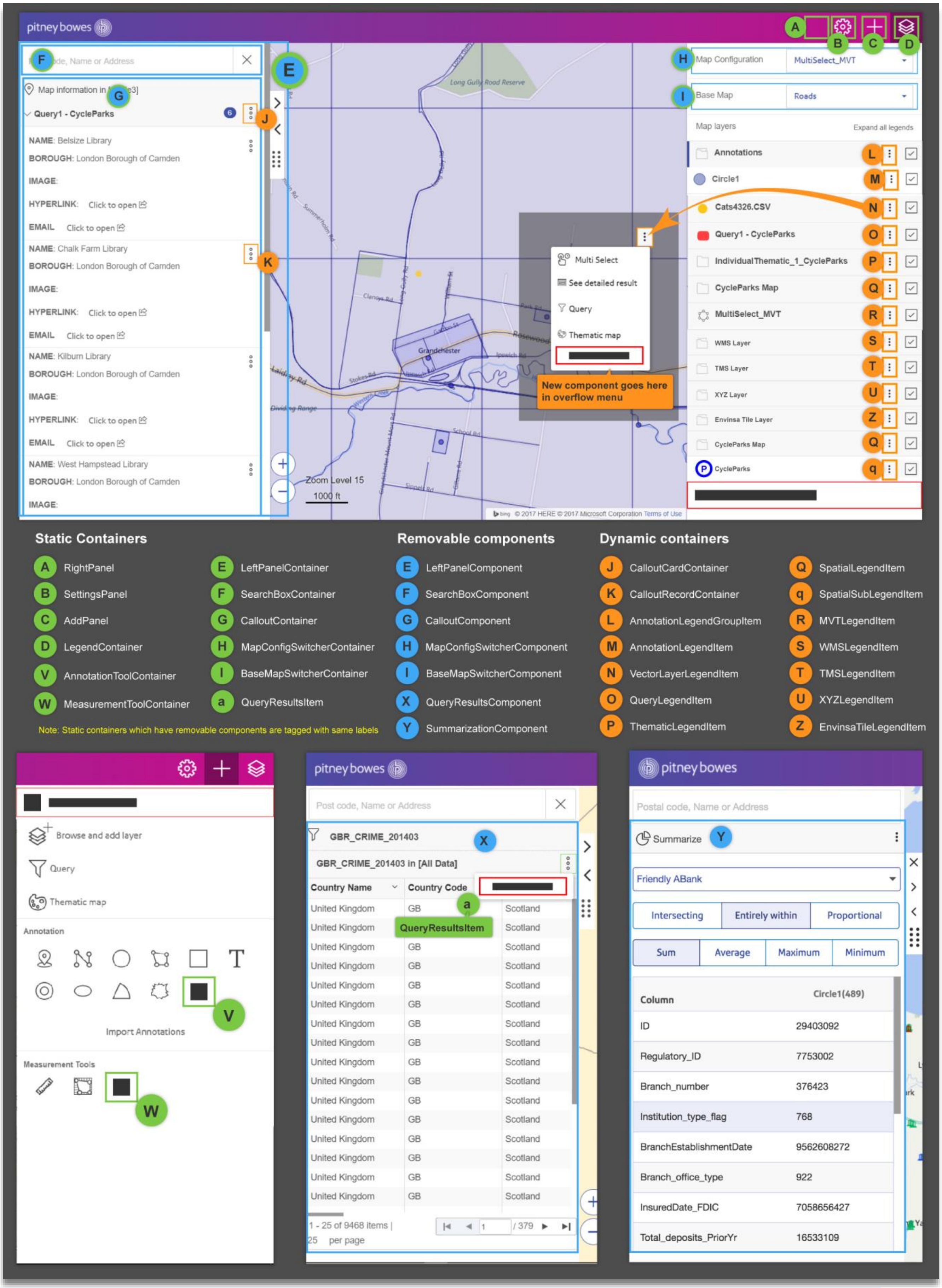
These containers are available for injection in the Spectrum Spatial Analyst Extensibility Platform, such as into the Settings panel, Layer panel, and Right panel. These injection points are available out-of-the-box for developers to add their components. It is irrelevant what type of data or map you host on Spectrum Spatial Analyst from Spectrum Spatial, a Web Mapping Service (WMS), or Vector layer. Static containers are available for injection.

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.



Static Containers

A	RightPanel
B	SettingsPanel
C	AddPanel
D	LegendContainer
V	AnnotationToolContainer
W	MeasurementToolContainer
E	LeftPanelContainer
F	SearchBoxContainer
G	CalloutContainer
H	MapConfigSwitcherContainer
I	BaseMapSwitcherContainer
a	QueryResultsItem

Note: Static containers which have removable components are tagged with same labels

Removable components

E	LeftPanelComponent
F	SearchBoxComponent
G	CalloutComponent
H	MapConfigSwitcherComponent
I	BaseMapSwitcherComponent
X	QueryResultsComponent
Y	SummarizationComponent

Dynamic containers

J	CalloutCardContainer
K	CalloutRecordContainer
L	AnnotationLegendGroupItem
M	AnnotationLegendItem
N	VectorLayerLegendItem
O	QueryLegendItem
P	ThematicLegendItem
Q	SpatialLegendItem
q	SpatialSubLegendItem
R	MVTLegendItem
S	WMSLegendItem
T	TMSLegendItem
U	XYZLegendItem
Z	EnvinsaTileLegendItem

Country Name	Country Code	
United Kingdom	GB	Scotland
United Kingdom		Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland
United Kingdom	GB	Scotland

Column	Circle1(489)
ID	29403092
Regulatory_ID	7753002
Branch_number	376423
Institution_type_flag	768
BranchEstablishmentDate	9562608272
Branch_office_type	922
InsuredDate_FDIC	7058656427
Total_deposits_PriorYr	16533109

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 papaparse (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*.

Hello World Extension Example

To embed an Angular 4 component into Spectrum Spatial Analyst:

1. Create an Angular 4 component; for example, *HelloWorldComponent*.
2. Create an Angular 4 module; for example, *DynamicModule* containing the *HelloWorldComponent*.
3. Place the module and component file into the folder under *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.

Creating and Invoking the Hello World Extension

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. This exercise answers:

1. How to create a custom Angular 4 component
2. How to inject a custom Angular 4 component into the Spectrum Spatial Analyst Extensibility Platform

Prerequisites

- 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)

Steps

1. Create a folder named extensions under:
   <ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\
   Typical installation path looks like:
   C:\Program Files\Pitney Bowes\SpectrumSpatialAnalyst\customerconfigurations\analyst\theme
2. Create a file called *dynamic.component.ts* under the folder:
   <ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme\extensions
3. Paste the below content 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.

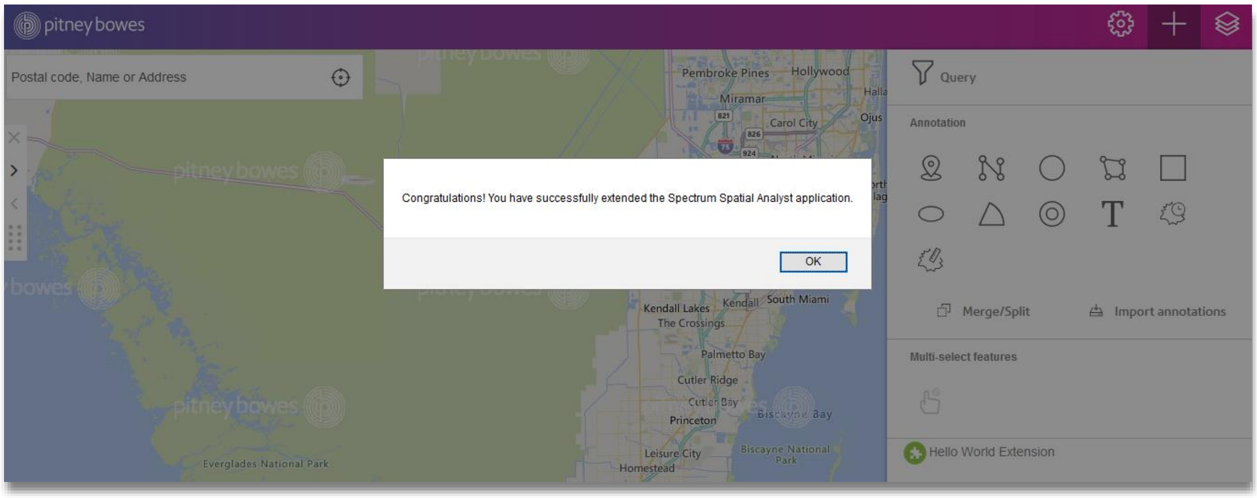
5. Create a file called *dynamic.module.ts* in:
   <ANALYST_INSTALL_PATH>\customerconfigurations\analyst\theme \extensions
6. Paste the following content in that file and save it:

```
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\Pitney Bowes\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 popup with a success message as shown in the following image:


Congratulations! You have successfully extended the Spectrum Spatial Analyst application.

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.

Configuring New Components

Configuration

A model is available to define and represent all the new components that are added. This is held in 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"}
```

```
                                                    ]}
}
```

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

Top Level Nodes

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,

Field Name	Type	Required	Description
			extensions/weather/weather.module.ts#WeatherModule
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

Field Name	Type	Required	Description
			not a third-party component. Specific components, like BaseMapSwitcher and SearchBox, may be removed from the application.
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"
        }
}
```

Components in a Different Module but the Same Container

```
{
	"name": "FINDXYModule",
	"description": "Find a defined x and y with a specific ICON.",
	"modulePath":  "extensions/dynamic.module.ts#DynamicModule",
	"components": [
	{
		"componentName": "FindXY",
		"parentComponentName": "AddPanel"
 }]
 }
{
	"name": "WMTSModule",
	"description": "Find a defined x and y with a specific ICON.",
	"modulePath": "extensions/another.module.ts#AnotherModule",
	"components": [
	{
		"componentName": "AddWMTSLAYER",
		"parentComponentName": "AddPanel"
	}]
}
```

Restricting the Instances of Dynamic Containers where a Component is Added

The Spectrum Spatial Analyst Extensibility Platform supports two types of containers: static and dynamic.

- Static component containers are available out-of-the-box and do not depend on the data or state of the application. The *AddPanel*, *SettingsPanel*, *LayerPanel*, and *LeftPanel* are examples of static components.
- Dynamic component containers depend on the state or data of the system. Examples of dynamic components include the overflow menus shown against different legends, and for map information, the Query legend, Thematic legend, and User Added Vector layer legend.

A container is assigned in the *CustomAnalystModuleConfig.json* file where it also references the name of the parent container.

A dynamic component is handled differently from a static component. If the association of a component is with a dynamic container, then the new component will be visible with all the instances of a dynamic container. For example, a third-party component adds a new menu to the *AnnotationLegendItem* dynamic container to be added against only circle annotations to query data and show a report within the radius of the circle: because a user may create more than one type of annotation, the new menu item would appear for all annotations.

The way to manage this is within the components code by referencing context parameters. Since dynamic containers pass context parameters to their children, then if there is a need to restrict the view of a new component for a given instance, the component can determine under what context it is to be shown. For example, the component can reference the *AnnotationLegendObject* context to see if it is a circle annotation type and decide to hide or show itself. For a detailed explanation and a list of context parameters for all dynamic components, refer to the Context Parameter section.

List of Static Containers

Name	**Description**	**Location**	**CustomAnalystModuleConfig Identifier (CaseSensitiive)**
Add Panel	Top Right panel + icon	Top right corner	AddPanel
Settings Panel	Panel represented by a cogwheel	Top right part of Spectrum Spatial Analyst browser screen	SettingsPanel
Right Panel	Panel containing all sub-panel settings, and add layer. Use this parent to have the component always visible and available at startup.	Top right corner holding all sub-panels	RightPanel
Left Panel	Panel that displays when clicking on the map	Left part of the screen after clicking the map	LeftPanel
Legend Container	Panel represented with Burger Icon	Top right corner	LegendContainer
Search Box Container	Includes a new search box for the Spectrum Spatial Analyst Extensibility Platform	Top Left corner. CSS can be used to the position/change the look and feel. Used especially for the cases when the user wants to replace an existing search of Spectrum Spatial Analyst with a custom one.	SearchBoxContainer
Query Results	Adds the component as a menu item of query results (to push the	Query Results panel in the top left once results are displayed	QueryResultsItem

Name	Description	Location	CustomAnalystModuleConfig Identifier (CaseSensitiive)
	query result to a web service for example)		
Annotation Tools Container	Provides custom annotations for the Spectrum Spatial Analyst Extensibility Platform. Annotation tools enable in Spatial Manager.	Add Panel Annotation Toolset	AnnotationToolsContainer
Map Config Switcher Container	Allows adding the component next to MapConfigSwitcher in the Settings Panel	SettingsPanel MapConfig dropdown	MapConfigSwitcherContainer
Base Map Switcher Container	Allows adding component next to BaseMapSwitcher in Settings Panel	SettingsPanel BaseMap dropdown	BaseMapSwitcherContainer
Layer Panel	Allows adding a component in Layer Panel	Layer Panel	LayerPanel
Measurement Tool Container	Allows a third-party component developer to provide custom measurement tool in Spectrum Spatial Analyst Extensibility Platform	Measurement Tool Container	MeasurementToolContainer

List of Dynamic Containers

Name	Description	Location	CustomAnalystModuleConfig Identifier (CaseSensitiive)
Annotation Legend Item	Line item corresponding to Annotation in Legend	Legend container	AnnotationLegendItem
Query Legend item	Line Item corresponding to the query created in Legend	Legend Container	QueryLegendItem
ThematicLegend Item	Line Item corresponding to the thematic created in Legend	Legend Container	ThematicLegendItem

Name	Description	Location	CustomAnalystModuleConfig Identifier (CaseSensitiive)
Vector Layer Legend Item	Line Item corresponding to the Vector layer added in Legend	Legend Container	VectorLayerLegendItem
TMS Legend Item	Line Item corresponding to the TMS layer in a map project	Legend Container	TMS Legend Item
XYZ Legend Item	Line Item corresponding to XYZ layer legend in map project	Legend Container	XYZLegendItem
WMS Legend Item	Line Item corresponding to WMS layer legend in map project	Legend Container	WMSLegendItem
Spectrum Spatial Group Layer Legend Item	Line Item corresponding to Spectrum Spatial Group layer legend in map project	LegendContainer	SpatialLegendItem
Spectrum Spatial Layer Legend Item	Line Item corresponding to Spectrum spatial layer legend in map project	LegendContainer	SpatialSubLegendItem
Envinsa Tile Legend Item	Line Item corresponding to Envinsa tile layer legend in map project	LegendContainer	EnvinsaTileLegendItem
MVT Layer Legend Item	Line Item corresponding to MVT layer legend in map project	LegendContainer	MVTLegendItem
Callout Card Container	Allows new component to be present at table level menu item of mapclick event	Left panel that comes after map click	CalloutCardContainer
Callout Record Container	Allows new component to be present at each record of a given table	Left panel comes after map click	CalloutRecordContainer

Name	Description	Location	CustomAnalystModuleConfig Identifier (CaseSensitiive)
Annotation Legend Group Item	Adds a new component at the Group level of the Annotation menu list.	Annotation Panel in legend container	AnnotationLegendGroupItem

Advanced Configuration Options

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 onboarding 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.

Embedding a New Library in the Spectrum Spatial Analyst Extensibility Platform

1. Register a library with the Spectrum Spatial Analyst Extensibility Platform by adding an entry in *CustomAnalystModuleConfig.json* file.
   The entry looks similar to:

```
"externalLibraryPath": [{
                                        "libName": "jquery",
                                        "libPath": "https://code.jquery.com/jquery-3.2.1.min.js"
                              },
{
                                        "libName": "gdal",
                                        “mainFilePath”:”index.js”
                                        "libPath": "../controller/theme/extensions/app/gdal"
                              }
]
```

2. *libPath* can be a local and relative path. If the path is relative, the path resolution happens based on the controller URL of Spectrum Spatial Analyst. The above example shows an actual path. You need to mention the controller to resolve libraries (libs) that are located under the theme folder: *customerconfigurations*. Spectrum Spatial Analyst will only ensure backward compatibility and successful upgrades for libraries kept under the extensions folder: *<ANALYST_INSTALL_PATH>/customerconfigurations/analyst/theme/extensions*
3. One can then refer to the embedded library in the component.
4. There is an example corresponding to usage of one such library within the Spectrum Spatial Analyst Extensibility Platform in the code links.
5. In case the library is hosted locally, and there is more than one file in the library, the system needs to know which JavaScript file to refer to for loading all the files. In that case, you need to provide another field as shown below:

```
{
                                        "libName": "testimport",

          "mainFilePath":"../controller/theme/extensions/js/common/testimport.js",
           "libPath": "../../../extensions/js/common"
                              }
```

Specifying the Map Configurations that Show 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 project association to a third-party component is a whitelisting:

- If we mention a component for a map project, then that component is visible only for that map project and not others.
- To make a component available in more than one map project but not in all of them, then whitelist 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 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.

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 dropdown. 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, then it cannot be 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.

Removable Components via Config File

Component Name	Identifier	Remove Only via config file
Base Map Switcher Component	BaseMapSwitcherContainer	Yes

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

Removable Components via 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:

Component Name
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 Parameters

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:

Dynamic Container 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
CalloutCardContainer	calloutObject
CalloutContainer	calloutRecordObject

Component Initialization Parameters

The Spectrum Spatial Analyst extensibility platform envisages cases where more than one instance of newly created angular 4 third-party components need to be onboarded. 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",
        "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.

Components 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.

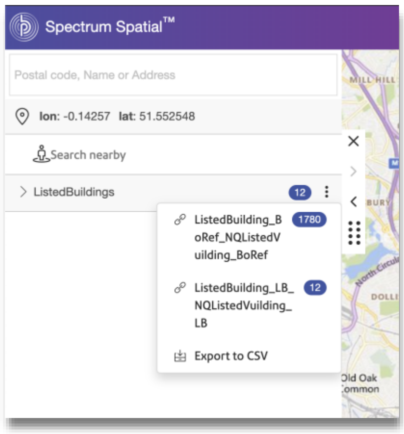
Components with Only Code and Without HTML

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.

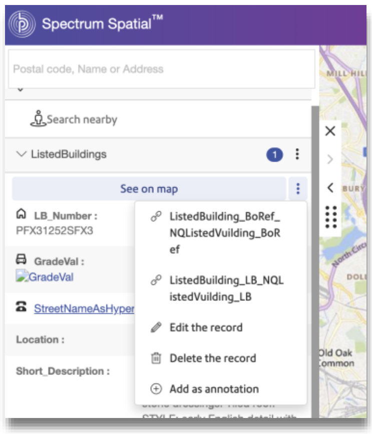
Reordering the Left-Hand Panel Menu

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

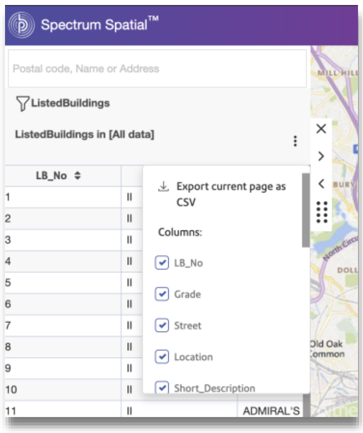
Layer level


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

Record level



Tabular grid's overflow menu



The layer and record levels use the same overflow menu components. Applying the change shown in the below CSS example affects the order of menu items for both 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, refractor 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: `
      <span class="btn btn-link text-left btn-block container-flex">
         </i>CallOutCardContainer
      </span>
   `,
})
```

Wrap the HTML element within an `</li>` element:

```
@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 a custom component to a layer's information 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.

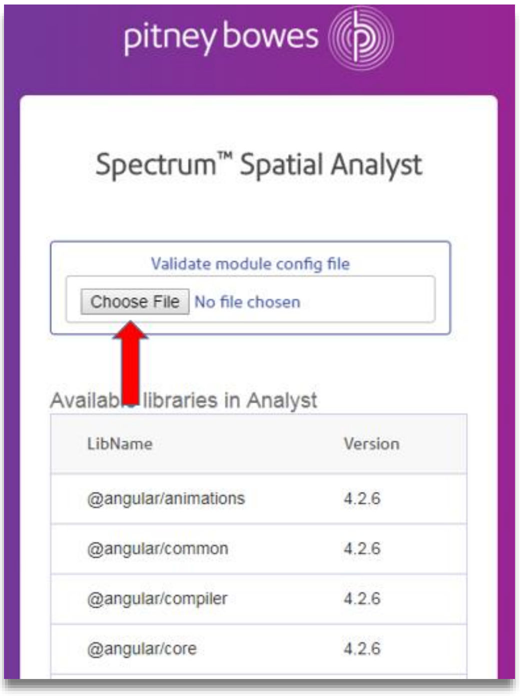
Validating your Components and Available Third-Party Libraries

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.



Availab libraries in Analyst

LibName	Version
@angular/animations	4.2.6
@angular/common	4.2.6
@angular/compiler	4.2.6
@angular/core	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” above.

Branding/Styling of 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 Restrictions for Modules/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.

Referencing Spectrum Spatial Analyst APIs in your Component

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 one has 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 querying 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.

Glossary of Terms

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 Ng Rx 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*.

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

3001 Summer Street
Stamford CT 06926-0700
USA

www.pitneybowes.com