Contents

Introduction and Overview ........................................................................................................... 4
  Introduction ............................................................................................................................. 4
  Overview ............................................................................................................................... 4

Hello World Example .................................................................................................................. 7
  Prerequisites ........................................................................................................................ 8
  Summary ............................................................................................................................... 10

Configuring New Components .................................................................................................. 11
  Configuration ....................................................................................................................... 11
  Containers .......................................................................................................................... 14
  Components in different module but the same container .................................................... 15
  Restricting the instances of dynamic containers where a component is added ................. 16
  List of Static containers ...................................................................................................... 16
  List of Dynamic containers ................................................................................................. 18

Advanced Configuration Options ............................................................................................... 20
  Referencing Third Party Libraries ...................................................................................... 20
  Specifying the map configurations that show components ................................................. 22
  Removing Existing Analyst Components ............................................................................. 22
  MapConfig based Component Removal ............................................................................. 23
  Removable Components via Config file .............................................................................. 23
  Removable Components via Adminconsole ......................................................................... 23
  Component Context Parameters ......................................................................................... 24
  Component Initialization Parameters ................................................................................. 26
  Components that can Run at Startup ................................................................................... 26
  Components with just code and without HTML ................................................................. 27

Validating your Components and Available third party libs in SSA Extensibility Platform ...... 27
  Module Config Validator ...................................................................................................... 27
  Third party libraries and its versions ..................................................................................... 29
  Branding/Styling of Third Party Components ...................................................................... 29
  Path Restrictions for Modules/Components ......................................................................... 29

Referencing the Analyst APIs in your Component ................................................................. 30
  Use of Existing Services, Store Actions, Selectors, Components of SSA Extensibility Platform ....... 30
Introduction and Overview

Introduction

Welcome to the Spectrum Spatial Analyst (SSA\Analyst) Extensibility Guide. From this release onwards, Spectrum Spatial Analyst has introduced a new extensibility platform that allows third parties to customize the product. This framework is based on the Angular 4.2.6 component model that allows new components to be created and added to Spectrum Spatial Analyst to provide a new or different functionality.

This document provides help on how Spectrum Spatial Analyst can be extended, the concepts involved along with examples. You should read this guide in conjunction with the extensibility API documentation available from SSA Admin console > Home page.

Overview

Spectrum Spatial Analyst 12.2 allows angular based components to be added dynamically at run-time to an already deployed and running instance. It is not necessary to compile components or build a custom version of 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, i.e. places where new components can be added. Components can be created to replace certain Analyst components, such as menus and also to be injected dynamically into existing Analyst components. Each of the injection points are represented by a unique identifier. Parent components where new extensions will be injected are called containers. The different types of container are described below-

Static Container

These containers are available all the time for injection in the SSA Extensibility Platform. For example, the settings panel, layer panel and right panel. These injection points are available out of the box to third party developers to inject their third party components into. It is irrelevant what type of data/map you host on SSA be it Spectrum Spatial, WMS or Vector layer, static containers are available for injection.
**Dynamic Container**

Dynamic containers are available based on data the SSA Extensibility Platform hosts. For example, Spectrum Spatial legend item will only be available if a Spectrum Spatial layer is included in the SSA 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 etc (which the child component can use to determine how it behaves and even whether it is rendered).

**Removable Component**

SSA is a composition of multiple individual components such as, the Left panel, Right panel, Map, Legend, Search box. Some of the components can be removed or replaced with a custom component from the SSA Extensibility Platform. Removable components are those components that can be removed for example, searchbox. Analyst allows you to remove components either via Admin console or by using SSA Extensibility Platform config file.
To inject a new component at one of the available extension points, a configuration file called `CustomAnalystModuleConfig.json` is available. This file allows you to configure containers for third party extensions, which components to remove and a wide range of other parameters about how the extension is to be used.

The vast majority of Analyst’s 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 services such as data flows, specifying queries, thematically styling map layers, etc. All these services have APIs which encapsulate a wide variety of third party libraries that are part of 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).

SSA uses a store based architecture for maintaining state and providing inter-component communications. Many services are available to third party developer via store actions and their corresponding selectors. In simple terms, 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 will dispatch an action with the necessary parameters via store dispatch. Use of the store will be an important aspect of many extensions which reference SSA’s extensibility APIs.

**Hello World Example**

Given below are the steps for the third party developer to add a component into Analyst:

1. **Generic steps to embed any component**
   1. Create an Angular 4 component, for example, `WMTSMapComponent`.
   2. Create an Angular 4 module, for example, `WMTSMapModule` containing the `WMTSMapComponent`.
   3. Place the module and component file in folder under `customerconfigurations/analyst/theme` folder.
   4. Create or update a module definition file representing that component.
   5. Validate the module definition file with the Analyst Custom Modules file validator (web page).
   6. Put the module definition file into the custom configuration folder once file is validated.
   7. Now, refresh the browser to see the component embedded in Analyst.

**Steps to embed Hello World**

The following section provides steps to add a new menu item to the Settings menu, which says Hello World when clicked.

Let’s start with sample hello world custom component. By the end of this exercise you should be able to answer the following questions-

1. How to create a custom Angular 4 component?
2. How to inject a created custom Angular 4 component into the SSA Extensibility Platform.
3. What is a parent container?

**Prerequisites**

- Basic knowledge of Angular 2/4
- Ability to code in Typescript
- Basic Understanding of SSA terminologies (Analyst, Adminconsole, customer_configuration, mapconfig etc.)

Time Duration: 15 mins

**Steps**

1. Create a folder named extensions in `<SSA_INSTALL_PATH>\webapps`. Typical installation path looks like below
   “C:\Program Files\Pitney Bowes\SpectrumSpatialAnalyst\Tomcat7\AnalystConnect\webapps”
2. Create a file name dynamic.component.ts under folder `<SSA_INSTALL_PATH>\webapps\extensions`.
3. Paste the below content into the file-

```typescript
import { Component, Input } from '@angular/core';
import { ComponentFactoryResolver } from '@angular/core';
import { ViewContainerRef } from '@angular/core';

@Component({
  selector: 'my-test',
  template: `<div (click)=sayHello() class=""><img class="fillColor" src="../controller/theme/extensions/app/images/icon-extensibility.png" alt="Icon-circle" height="25" width="25">sayHello</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 TestComponent{

  constructor() {
```
```
4. Create an images folder in `<SSA_INSTALL_PATH>\customerconfigurations\analyst\theme\extensions\`
5. Save the following image as `icon-circle.png` in `<SSA_INSTALL_PATH>\customerconfigurations\analyst\theme extensions\images` folder. Please note that theme is an existing folder in the SSA installation.

6. Create a file, `dynamic.module.ts` in `<SSA_INSTALL_PATH>\customerconfigurations\analyst\theme \extensions\`
7. Paste the following content in that file and save it.

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

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

8. Now go to the customer configuration folder in -
   “<SSA_INSTALL_PATH>\customerconfigurations\analyst”, the path looks like -
   “C:\Program Files\Pitney Bowes\SpectrumSpatialAnalyst\customerconfigurations\analyst”
9. Now, create a file called `CustomAnalystModuleConfig.json` under that directory.
10. Paste the following content into the `CustomAnalystModuleConfig.json` file and save it.
11. Go to the browser and launch the SSA URL from the address bar.
12. Go to the Add panel, (+) plus button on top right corner of application.
   You should see your first component appearing with the image pasted above.
13. Click on it and a popup will appear with a “Hello Analyst Extensions” message as shown in
    the following image:

   ![Image of the popup](image_url)

**Summary**

In the above hello world example, in the first few steps an Angular component and Angular module, we also
provided some resources needed for the component like image. Then we created a configuration needed to
inject that component into the SSA Extensibility Platform. Finally that our component was getting injected
correctly.
The parentContainer tag in the `CustomAnalystModuleConfig.json` file is responsible for injecting a component into the correct container. The SSA Extensibility Platform provides multiple such containers to place third party components appropriately at correct position in the screen layout.

Please refer to the section below to understand in detail about containers.

Note: The name of the class in the component file and the name of the component in the configuration file should be identical. 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 will look-

```json
"modules": [{
  "name": "GIQueryModule",
  "description": "Module For GI query",
  "modulePath": "extensions/app/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",
  }
  ]
},
```

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

**Top Level Nodes**

The top two level nodes are listed here-

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>Json Array</td>
<td>Yes</td>
<td>Comprises an array of multiple module definitions as described in definitions section above.</td>
</tr>
<tr>
<td>ComponentsToRemove</td>
<td>Json Array</td>
<td>Yes</td>
<td>Comprises an array of pre-existing components provided out-of-the-box with Analyst that would be removed.</td>
</tr>
</tbody>
</table>

**Module JSON Object**

Each array element inside the Modules node will define a module as follows

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Yes</td>
<td>The name should not be the same as any of the Analyst modules. You can find a list</td>
</tr>
<tr>
<td>Field Name</td>
<td>Type</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>optional</td>
<td>Gives details about the purpose of the module for users looking at the configuration file.</td>
</tr>
<tr>
<td>ModulePath</td>
<td>String</td>
<td>Yes</td>
<td>The location of the module in the file system. This will be in: <code>customerconfigurations/analyst/theme/extensions</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note: The folder containing the module should be in this path for it to be accessible</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note: Each module can have a separate folder. For example, <code>extensions/app/weather/weather.module.ts</code>#WeatherModule</td>
</tr>
<tr>
<td>Components</td>
<td>Map</td>
<td>Yes</td>
<td>This is a key value pair where Key = Name of the component Value = The parent container in Analyst where the component is to be injected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note: The Component Name should match the Module Name that you have declared for creating angular 4 component class and not the selector.</td>
</tr>
<tr>
<td>externalLibraryPath</td>
<td>JSONArray</td>
<td>Optional</td>
<td>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 Analyst Application.</td>
</tr>
<tr>
<td>mapConfigAssociated</td>
<td>Map</td>
<td>Optional</td>
<td>A key value pair where: Key = Analyst map configuration name Value = Array of components that will be visible for that map config.</td>
</tr>
</tbody>
</table>
### ComponentToRemove

Each array element inside the ComponentToRemove node will define the following:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentName</td>
<td>String</td>
<td>Yes</td>
<td>Name of the component to be removed. Note this is an existing Analyst component and not a third party component. Certain components in Analyst are allowed to be removed from the application like BaseMapSwitcher, SearchBox etc.</td>
</tr>
<tr>
<td>fromMapConfig</td>
<td>String</td>
<td>Optional</td>
<td>Removes the component from a given mapconfig. If omitted it will remove component from all the mapconfigs</td>
</tr>
</tbody>
</table>

### Containers

The SSA extensibility platform divides the entire layout of the product into different parts called containers. Containers are parents to the new components created by third party developers. Containers allow a third party developer to place their visual/non visual components in the right place. For example, if a third party developer wants to place a component in the Add Panel then users 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 position of the component in a given container can be controlled by the CSS of a new component.
There can be cases where 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 same parent container will be repeated in each module entry. The following sample illustrates this-

Components in the same module and same container

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

Components in different module but the same container

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

```json
{
    "name": "WMTSModule",
    "description": "Find a defined x and y with a specific ICON.",
    "modulePath": "extensions/app/dynamic.module.ts#DynamicModule",
    "components": [ 
    {
        "componentName": "AddWMTSLAYER",
        "parentComponentName": "AddPanel"
    } ]
}
```
Restricting the instances of dynamic containers where a component is added

The SSA Extensibility Platform supports two types of containers: static and dynamic.

Static component containers are the containers that are available out of the box and do not depend on the data/state of the application to be shown. The AddPanel, SettingsPanel, LayerPanel and LeftPanel are examples.

Dynamic component containers are the ones that depend on the state/data of the system. For example, this includes the overflow menus shown against different legends and for map information, Query Legend, Thematic legend Item, User Added Vector layer legend item. In both cases you will reference the name of the parent container when assigning it in the CustomAnalystModuleConfig.json File.

However, there is one subtle difference between components associated with dynamic containers and static containers. 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, let’s say a third party component adds a new menu to the “AnnotationLegendItem” dynamic container and the intent is for this to be added against only circle annotations (perhaps to query data and show a report within the radius of the circle, as 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Location</th>
<th>CustomAnalystModuleConfig Identifier (CaseSensitive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Panel</td>
<td>Top Right panel + icon</td>
<td>Top right corner</td>
<td>AddPanel</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
<td>Location</td>
<td>Parent Component</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Settings Panel</td>
<td>Panel represented by cog wheel</td>
<td>Top right part of Analyst browser screen</td>
<td>SettingsPanel</td>
</tr>
<tr>
<td>Right Panel</td>
<td>Panel Containing all the subpanel setting, add layer. Use this parent if you want your component to be visible always and needed at startup.</td>
<td>Top right corner holding all sub panels</td>
<td>RightPanel</td>
</tr>
<tr>
<td>Left Panel</td>
<td>Panel that comes on click of the map</td>
<td>Left part of the screen once map is clicked</td>
<td>LeftPanel</td>
</tr>
<tr>
<td>Legend Container</td>
<td>Panel represented with Burger Icon</td>
<td>Top right corner</td>
<td>LegendContainer</td>
</tr>
<tr>
<td>Search Box Container</td>
<td>Allow a new search box to be added to SSA Extensibility Platform</td>
<td>Top Left corner. CSS can be used to position/change the look and feel. Used especially for the cases when the user wants to replace an existing search of Analyst with a custom one.</td>
<td>SearchBoxContainer</td>
</tr>
<tr>
<td>Query Results</td>
<td>Allows adding of component as menu item of query results. For example, a user want to push query result to some webservice.</td>
<td>Query Results panel in top left once results are displayed</td>
<td>QueryResultsItem</td>
</tr>
<tr>
<td>Annotation Tools Container</td>
<td>Allows third party component developer to provide custom annotations in SSA Extensibility Platform. Annotation tools must be enabled from adminconsole for this to be used.</td>
<td>Add Panel Annotation Toolset</td>
<td>AnnotationToolsContainer</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Location</td>
<td>CustomAnalystModuleConfig Identifier (CaseSensitive)</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>Annotation Legend Item</td>
<td>Line item corresponding to Annotation in Legend</td>
<td>Legend container</td>
<td>AnnotationLegendItem</td>
</tr>
<tr>
<td>Query Legend item</td>
<td>Line Item corresponding to the query created in Legend</td>
<td>Legend Container</td>
<td>QueryLegendItem</td>
</tr>
<tr>
<td>ThematicLegendItem</td>
<td>Line Item corresponding to the thematic created in Legend</td>
<td>Legend Container</td>
<td>ThematicLegendItem</td>
</tr>
<tr>
<td>Vector Layer Legend Item</td>
<td>Line Item corresponding to the Vector layer added in Legend</td>
<td>Legend Container</td>
<td>VectorLayerLegendItem</td>
</tr>
<tr>
<td>TMS Legend Item</td>
<td>Line Item Corresponding to the TMS layer in a mapconfig</td>
<td>Legend Container</td>
<td>TMS Legend Item</td>
</tr>
<tr>
<td>Legend Item</td>
<td>Line Item</td>
<td>Legend Container</td>
<td>Item Type</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>XYZ Legend Item</td>
<td>Line Item corresponding to XYZ layer legend in mapconfig</td>
<td>Legend Container</td>
<td>XYZLegendItem</td>
</tr>
<tr>
<td>WMS Legend Item</td>
<td>Line Item corresponding to WMS layer legend in mapconfig</td>
<td>Legend Container</td>
<td>WMSLegendItem</td>
</tr>
<tr>
<td>Spectrum Spatial Group Layer Legend Item</td>
<td>Line Item corresponding to Spectrum spatial Group layer legend in mapconfig</td>
<td>LegendContainer</td>
<td>SpatialLegendItem</td>
</tr>
<tr>
<td>Spectrum Spatial Layer Legend Item</td>
<td>Line Item corresponding to Spectrum spatial layer legend in mapconfig</td>
<td>LegendContainer</td>
<td>SpatialSubLegendItem</td>
</tr>
<tr>
<td>Envinsa Tile Legend Item</td>
<td>Line Item corresponding to Envinsa tile layer legend in mapconfig</td>
<td>LegendContainer</td>
<td>EnvinsaTileLegendItem</td>
</tr>
<tr>
<td>MVT Layer Legend Item</td>
<td>Line Item corresponding to MVT layer legend in mapconfig</td>
<td>LegendContainer</td>
<td>MVTLegendItem</td>
</tr>
<tr>
<td>Callout Card Container</td>
<td>Allows new component to be present at table level menu item of mapclick event</td>
<td>Left panel that comes after map click</td>
<td>CalloutCardContainer</td>
</tr>
<tr>
<td>Callout Record Container</td>
<td>Allows new component to be present at each record of a given table</td>
<td>Left panel comes after map click</td>
<td>CalloutRecordContainer</td>
</tr>
<tr>
<td>Annotation Legend Group Item</td>
<td>Allows new component to be added at Group level of Annotation menu items.</td>
<td>Annotation Panel in legend container</td>
<td>AnnotationLegendGroupItem</td>
</tr>
</tbody>
</table>
Advanced Configuration Options

Referencing Third Party Libraries

The SSA 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 SSA Extensibility Platform facilitates onboarding such libraries with ease. In order to onboard/use new libraries in the component, follow steps given below. Libraries can be references from the file system of the Analyst server or can be referenced from a hosting site/CDN. The Mechanism for registering the library is the same in both the cases. There are certain restrictions that the SSA 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.
3. Making sure to check for license terms, vulnerability and certification of new libs in the SSA Extensibility Platform is the responsibility of the third party component developer.
4. If someone intentionally violates point 1 and more than one version of the same library is added to SSA 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 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 again in the other modules.

Process of embedding a new libs in SSA Extensibility Platform is below

1. Register a library with the SSA Extensibility Platform by adding an entry in CustomAnalystModuleConfig.json file.
2. The entry will look similar to the one below:

   ```json
   "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"
   }]
   ```
3. *libPath* can be local relative path. In case if path is relative that path resolution happens based on controller url of analyst. Please refer to above example for actual path, you need to mention controller to resolve libs under theme folder of customerconfigurations. Please note that Analyst will only ensure backward compatibility and successful upgrades for libs being put up under extensions folder of `<SSA_INSTALL_PATH>/customerconfigurations/analyst/theme/extensions`.

4. One can then refer the embedded library in the component.

5. There is an example corresponding to usage of one such library within the SSA Extensibility Platform in the code links.

6. 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 in order to load all files. In that case you need to provide another field as given below:

```json
{
    "libName": "testimport",
    "mainFilePath": ".../controller/theme/extensions/js/common/testimport.js",
    "libPath": "../../../extensions/js/common"
}
```
Specifying the map configurations that show components

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

Map configuration association to a third party component is kind of a whitelisting. If we mention a component for a given map configuration then that component will be visible only for that map configuration and not others. In cases where the third-party developer wants to show a component in more than one map configuration (but not in all of them) then users needs to whitelist in all of the map configurations. If component needs to be available in all the mapconfigs we don’t have to provide any entry in mapconfigAssociated tag. For example-

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

Removing Existing Analyst Components

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

The list of components that can be completely removed from the SSA extensibility platform are given below. It depends on customer needs if they want to introduce new component or just remove it.

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

In order to remove a component, a third party developer needs to mention the component in CustomAnalystModuleConfig.json file as shown below.

**Note** - If a component is removed, for example, leftpanel then it cannot be a parent container of any third party component.
MapConfig based Component Removal

There may be certain use-cases where third party developer wants to remove certain component in certain conditions only. In this case, all users needs to create a map configuration and configure the component to be removed in the CustomAnalystModuleConfig.json file. A typical entry in CustomAnalystModuleConfig.json for such case will look like below.

"componentsToRemove":
 {
  "componentName":"BaseMapSwitcherComponent",
  "fromMapConfig":"Drive Time"
}

There may be a case when same component has to be removed from more than one map configuration. In this case the entry for a removing the component has to be repeated for each map configuration.

Removable Components via Config file

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Identifier</th>
<th>Remove Only via config file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Map Switcher Component</td>
<td>BaseMapSwitcherContainer</td>
<td>Yes</td>
</tr>
<tr>
<td>Map Config Switcher Component</td>
<td>MapConfigSwitcherContainer</td>
<td>Yes</td>
</tr>
<tr>
<td>Left Panel Component</td>
<td>LeftPanelContainer</td>
<td>Yes</td>
</tr>
<tr>
<td>Query Results Component</td>
<td>QueryResultsComponent</td>
<td>Yes</td>
</tr>
<tr>
<td>Callout Container Component</td>
<td>CalloutContainerComponent</td>
<td>No (Via Adminconsole as well)</td>
</tr>
<tr>
<td>Search Box Component</td>
<td>SearchBoxContainer</td>
<td>Yes</td>
</tr>
<tr>
<td>Summarization Results Component</td>
<td>SummarizationComponent</td>
<td>No (Via Adminconsole as well)</td>
</tr>
<tr>
<td>Legend Container Component</td>
<td>LegendContainerComponent</td>
<td>Yes</td>
</tr>
<tr>
<td>Summarization Component</td>
<td>SummarizationComponent</td>
<td>No (Via Adminconsole as well)</td>
</tr>
</tbody>
</table>

Removable Components via Adminconsole

SSA also supports removal of component via adminconsole. Depending on the use-case you can opt for removing some common components via adminconsole. The list of components is shown below.

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Identifier</th>
<th>Remove Only via Adminconsole</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td></td>
<td>Yes (Via Adminconsole as well)</td>
</tr>
</tbody>
</table>
The SSA extensibility platform provides support for passing in context parameters from dynamic container components to its child components, including child components created by third party developers. Third party 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 third party developer creates a custom component to query within the circle annotation. The SSA Extensibility Platform passes in all the information pertaining to circle annotation to the custom component like radius, XY location, name of annotation etc.

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

In order to access this context data a third party 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 have a specific name for context parameters for example annotationLegendItem context parameter name annotationLegendObject. The following table provides the names of all the context data parameters that are available for different dynamic components:

<table>
<thead>
<tr>
<th>Dynamic Container Name</th>
<th>Context parameter Name (For example, data. annotationLegendObject)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnnotationLegendItem</td>
<td>annotationLegendObject</td>
</tr>
<tr>
<td>EnvinsaTileLegendItem</td>
<td>legendGroupObject</td>
</tr>
<tr>
<td>MVTLegendItem</td>
<td>legendGroupObject</td>
</tr>
<tr>
<td>AnnotationLegendGroupItem</td>
<td>annotationGroupObject</td>
</tr>
<tr>
<td>Legend Item</td>
<td>Object Name</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>QueryLegendItem</td>
<td>queryLegendObject</td>
</tr>
<tr>
<td>SpatialLegendItem</td>
<td>legendGroupObject</td>
</tr>
<tr>
<td>SpatialSubLegendItem</td>
<td>legendObject</td>
</tr>
<tr>
<td>ThematicLegendItem</td>
<td>legendGroupObject</td>
</tr>
<tr>
<td>TMSLegendItem</td>
<td>legendGroupObject</td>
</tr>
<tr>
<td>VectorLayerLegendItem</td>
<td>vectorLayerLegendObject</td>
</tr>
<tr>
<td>WMSLegendItem</td>
<td>legendGroupObject</td>
</tr>
<tr>
<td>XYZLegendItem</td>
<td>legendGroupObject</td>
</tr>
<tr>
<td>CalloutCardContainer</td>
<td>calloutObject</td>
</tr>
<tr>
<td>CalloutContainer</td>
<td>calloutRecordObject</td>
</tr>
</tbody>
</table>
Component Initialization Parameters

The SSA 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 SSA extensibility platform supports such parameter sharing via init parameters among multiple component instances. In order to support getting access to the init parameter the third party developer needs to take the below steps;

1. Users need to declare an input field called data in its component.
2. Users need to add an entry corresponding to component in CustomAnalystModuleConfig.json. For example -

   ```json
   "modules": [{
     "name": "GIQueryModule",
     "description": "Module For GI query",
     "modulePath": "../../../extensions/app/dynamic.module.ts#DynamicModule",
     "components": [{componentName: "TestComponent", "parentComponentName": "SettingsPanel"},
                     {"InitParameters": {
                     "apiKey": "abcdef"
                     }]
     }
   }]
   ```

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

Init parameters supports all types of data that javascript supports, at the same time it does not put any restrictions on the size of the parameters supplied.

Components that can Run at Startup

The SSA extensibility platform supports running components that are required during startup time (when a user first browses to Analyst). 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. So, if a third party developer wanted to make their component available at the time of startup, they can declare the RightPanel as its parent so that component comes into existence at startup itself.
Components with just code and without HTML

The SSA 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 in nature. Now, if a third party developer wants to create a component without HTML it can keep the template blank and the SSA Extensibility Platform will call the component appropriately at the time of instantiating its parent container. For example, a third party developer creates a component that gets weather data from remote API and passes this on to some other component for further processing. In this case let’s assume the component developer makes it a child of the AddPanel container. So when a user clicks on the AddPanel in Analyst, the third party component will be called and in turn it will make a call to get the weather data.

Validating your Components and Available third party libs in SSA Extensibility Platform

Module Config Validator

Since the module configuration can become complex if many modules are added to it, the SSA extensibility platform provides a Module config validator that will allow you to validate the CustomAnalystModuleConfig.json file. The tool is simple to use, you just need to browse to it as follows:

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

2. Now, the developer will choose and browse to a manually authored CustomAnalystModuleConfig.json file on the file system. The tool will validate the file. It will also show any errors to the user.

3. Once validation is complete the developer can place the CustomAnalystModuleConfig.json file into the customer_configuration folder of the Analyst installation.

The ModuleConfig validator will do semantic as well as syntactic validation of the CustomAnalystModuleConfig.json. It will give all the errors corresponding to missing braces, un-equal parenthesis, not giving mandatory fields etc. Apart from this, it will also validate the location of modules as well as external js files provided in the CustomAnalystModuleConfig.json file. It is recommended that third party developers ensure that the CustomAnalystModuleConfig.json file is valid before deploying it to Analyst in the customer_configuration 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 its versions

Module config validator page also provides list of libraries that are part of SSA Extensibility Platform and its corresponding version. It is important to go over the list if you want to onboard a new library in the SSA Extensibility Platform. If a library is available in the SSA 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 SSA via tech support. If library in 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 SSA extensibility platform provides a comprehensive branding/styling as part of the brand CSS facility to customize the look and feel of Analyst. A number of different brands can be created and referenced in map configuration via the Admin Console. A third party developer reference all of the branding classes in the CSS to adjust the look and feel of their components as per the branding guidelines of Analyst. For example, let’s assume a third party developer created a component, which has a button. Now, users wants to align the color or button and shape as per the default branding of Analyst. They can use the branding classes that the core Analyst uses to style its components and get the look and feel exactly the same as the SSA Extensibility Platform. In our extension examples we provided lots of SSA Extensibility Platform examples on how to achieve this.

Path Restrictions for Modules/Components

This section shares information regarding the location of custom components. It is recommended to put the components in the extensions folder in the `<SSA_INSTALL_PATH>/Customerconfigurations/analyst/theme/`. It will help the SSA installer in backup and restore activity during upgrades. It will also ensure that during installation and upgrades your extensions are not lost. Paths mentioned in the CustomAnalystModuleConfig.json are static.
Referencing the Analyst APIs in your Component

Use of Existing Services, Store Actions, Selectors, Components of SSA Extensibility Platform

Store

Ngrx store is a base architecture for the SSA Extensibility Platform. It is the primary mechanism to consume the resources of the SSA 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 parameter in the component’s constructor. We have provided various examples with the 12.2 release to showcase how it can be used.

The real power of the SSA Extensibility Platform comes from re-using many services, components and third-party libraries that come out of the box with 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 own component. There are also a multiple set of utilities exposed via the SSA Extensibility Platform. A complete list of these can be found in the API docs shipped with SSA installation.

Please note that, whilst the API docs list all of the functionality in Analyst’s components, third party developers are advised to only use the public APIs/components of the Analyst extensibility platform. This is necessary to maintain backward compatibility. In case developer uses private APIs then backward compatibility is not guaranteed. Private APIs are subject to change without prior notice.

Openlayers

Most third party components are likely to need to interact with the Openlayers library that is available out of the box with 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 direct with Openlayers, in most circumstances it is recommended that this is done via store actions. The use of store actions in particularly will ensure that state is maintained consistently between say the map and the legend panel where actions on one affect the other.
Definitions used in this document

**Component**
This is an Angular 4 component written in typescript (TS) and provided to the SSA Extensibility Platform to be embedded at run time.

**SSA Extensibility Platform**
The architecture of Analyst that allows third party components to be dynamically added at run-time and the set of core services that Analyst will provide for re-use by embedded components.

**SystemJS**
Main module that Analyst uses to boot-strap and enable run time embedding of third party components in Analyst.

**CustomModulesDefinition.json**
Configuration file configuring the definition of third party components

**Module**
Smallest unit of third party code that the SSA Extensibility Platform can embed. A module is Angular 4 module can comprise of one or multiple components mentioned in point 1.

**Store**
The Ng rx store on which the SSA Extensibility Platforms front end architecture is based. Most of the services that a third party developer can use are available via Store Actions and their corresponding selectors. In simple terms 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 will dispatch an action with the necessary parameters such as layerUrl/extents/center etc. via store.dispatch

Useful links

https://angular.io/tutorial/toh-pt1

https://angular.io/api

https://v2.angular.io/docs/js/latest/cookbook/ts-to-js.html

https://github.com/angular/quickstart

https://code.visualstudio.com/download