SSR Radar Health

Technical Design Document

July 22, 2022

Project Summary

The purpose of this task is to provide insight into the status of the radars that are providing radar tracks to AVERT C2. In addition to indicating the status of the radars, we will also provide limited control of the radars to change the status. To accomplish these goals, we will add a new access point integration to the MIS API. The radars will be considered access points in AVERT C2 as this will give us the ability to indicate the location of the radars, the status of the radars, and provide existing UI/UX for issuing commands to the radars.

Software Design Considerations

The radar objects provided by the SSR MIS API have an “antennaRpm” property that indicates how fast the radars are spinning. Typically, when a specific radar is not reporting tracks, it is because the radar is no longer spinning for some reason. We will use the antennaRpm property and a configurable threshold to represent the health of the radar by changing its map icon color to green if it is healthy or red if it is unhealthy.

It should be noted that in this integration, we are assuming that the SSR MIS API is providing accurate and live data. We will not be adding handling for missing data or inaccurate reporting to this integration.

User Interface Design

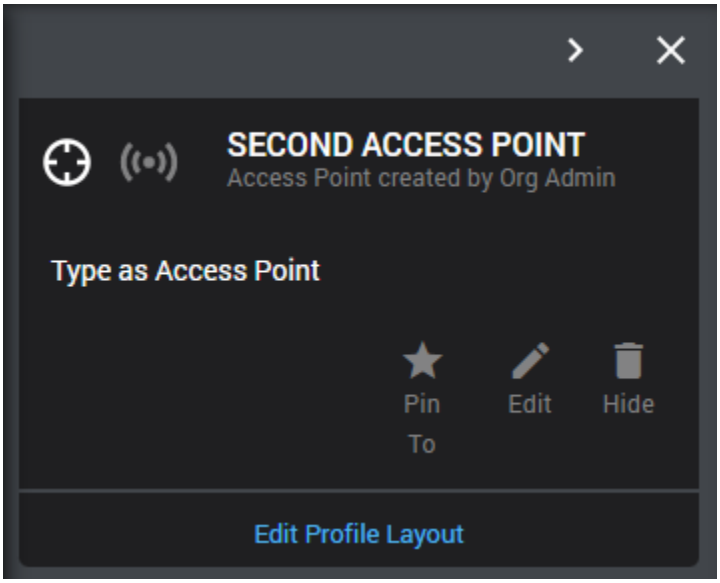
Radars will be added to AVERT C2 as access points. This means they will leverage all existing access point functionality, including map icons, profiles, control, and manual map/facility placement by the user.

Map Icons

The two map icons we will use for the radars are the green and red sensor icons. The green icon will represent a healthy radar and the red icon will represent an unhealthy radar.

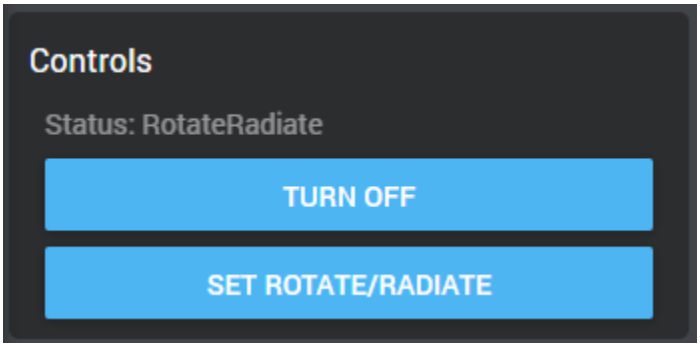
Profile

The profile for the radars will be the existing access point profile. One thing to note is that we want to indicate that these access points are radars. This means there will likely be additional UI work that needs to be done to allow for different access point types than “Access Point”.



Control

The radars will use the existing control widget in the access point profile to handle issuing commands to the radars. The two configured commands will be to turn the radar off and to set the radar’s status to RotateRadiate.



Placement

The radars will utilize the existing placement workflow for access points.

User Interface Code Specification

Map Icon

To change the map icon based on the antennaRpm property, we'll need to add a mapIconTemplate property to the access point feed. This property should be a properly formatted JSONata template that will use the green icon if the antennaRpm property is above a given threshold or the red icon if the property is below the given threshold.

Ex. "'Sensor_' & (properties.antennaRpm < 10 ? 'red' : 'green')"

Control

The controls should be driven from the access point features property. This functionality was completed in version 2022 R2, however portions of it were hard-coded and need to be changed. Rather than the features' command property containing a string indicating the command, it will need to be changed to the full integration-app URL. For example, if the existing feature has a command of "open", it will need to be changed to "/integration-app/api/externalSystem/ssr-radar/resource/open" assuming the external system ID is "ssr-radar".

Once this change has been made, changes to the AccessControlWidget in orion-components will need to be made to simply issue a POST to the command URL using the restClient in client-app-core.

Service Code Specification

Radar Data

The three main components to the radar data integration are the edge interface, the transform, and the ecosystem changes necessary to support it.

Edge Interface

For the edge interface, we can use the existing rest-polling-client in integration app. The endpoint that we will need to poll is "/api/radars?expand=control" where the base url is the SSR MIS API the customer is using. This edge interface supports adding headers to the request, so we will utilize this to pass through the authentication header. The polling interval should be set to 10 minutes by default, but we'll be able to adjust this after deployment if necessary.

Transform

We will need to create a new transform (ssr-radars-transformer) to handle the body of the response from the edge interface. The radars can be found in the items array in the payload that the transformer receives.

Since this is our first access point integration using the edge/transform workflow, we will need to add support in the transform-process-helper. This should be as easy as adding "external_accessPoint" to the entity types object.

The properties we are interested in for the radar access point can be found in the table below.

Access Point Property	SSR Radar Property
sourceId	id
name	radarFullName
location	position
status	radarState
antennaRpm	antennaRpm

In addition to these properties, we will also need to add features to the radar access point properties. The only two features we are interested in are one to turn the radar off and one to set its status to RotateRadiate. Below is an example of what these might look like, but they could require changes depending on the radar control implementation.

```
[
      {
            "command": "/integration-app/api/externalSystem/ssr-
radar/resource/updateRadarStatus?radar=1&status=Off",
            "label": "Turn Off"
      },
      {
            "command": "/integration-app/api/externalSystem/ssr-
radar/resource/updateRadarStatus?radar=1&status=RotateRadiate",
            "label": "Set Rotate/Radiate"
      }
]
```

Ecosystem Changes

In ecosystem, we'll need to add support for external access points to the externalSystemProcessor. The entity type should match what was used in the transform-process-helper in integration-app. For consistency, "external_accessPoint" should be used in both places.

When an access point is received, we'll need to figure out if we need to add it as a new access point or update an existing one. To do this, we can use the getBySourceId method in the accessPointModel. If an access point is returned from that method, we know we're updating an existing one. If nothing is returned, we know we need to add a new one.

Radar Control

For the control portion of this integration, we will be adding a new postResource method to the existing ssr-radar-api-manager. The resourceType to support these controls should be “updateRadarStatus”. The method should handle two query string parameters: radar and status. The radar parameter can be used as the SSR radar id property and the status parameter will be the target status for the call.

The SSR MIS API endpoint that should be used is a POST to /radars/{id}/control/state where a radarState property is passed in the JSON body of the request. In this endpoint, the id will be the radar parameter passed to the postResource method. The value of the radarState property will be the status parameter passed to the postResource method.

In order to use the existing ssr-radar-api-manager with this new external system, we will need to add support for specifying an API manager in the external system configuration. To do this, we will update the switch statement in the external-system-api-manager-factory to use the apiManager property of the external system configuration if it is present. Otherwise, it should continue to use the externalSystemId variable. This means in our new external system we will need to add an apiManager property and set its value to “ssr-radar”.

Example request to integration app:

```
GET /integration-app/api/externalSystem/ssr-
radar/resource/updateRadarStatus?radar=1&status=RotateRadiate
```

Example request to SSR MIS API:

```
POST /radars/1/control/state
{
    “radarState”: “RotateRadiate”
}
```

In addition to these changes in the ssr-radar-api-manager, we will need to make changes to the other access point control interface(s) to support the new feature command structure. At the time of writing, the only other access point control interface is the udc-accesscontrol-api-manager. If you need more details for how to make this change, please reach out for help.

Database Changes

External System

A new external system will need to be added to the sys_externalSystem table in the ecosystem database to support the new edge and transform interfaces. This external system will need to provide the configuration necessary for the edge interface (rest-polling-client) and the new transform process (ssr-radars-transformer). It will also need to specify the apiManager property as “ssr-radar”. The feedId property should be set to the feedId of the corresponding feed type.

Feed Type

A new feed type will need to be added to the sys_feedTypes table in the ecosystem database for the access points. The mapIconTemplate property will need to be set to the JSONata template used to determine which map icon is shown based on the determined health of the radar. The feedId property that is set for this feed type will need to be set as the feedId property on the corresponding external system.

Testing Requirements

We will be able to test part of this integration using the public demo SSR MIS API. The connection information for the demo system can be found in the Notes section of this document. We will be able to use this demo system to test the edge interface, transform process, and UI changes. We will *not* be able to test the control portion of this integration with the demo system because our credentials do not have permission to control the radars. The control portion of this integration will need to be tested on-site.

Notes

SSR MIS API Demo System

URL: https://mis.ssreng.com/api
Username: guest
Password: guest-of-ssr