C4C-SBS-Extensions Refactoring POC functions Approach

Contents

[Abstract 1](#_Toc110520691)

[Code configurability 1](#_Toc110520692)

[Function code refactor approach 1](#_Toc110520693)

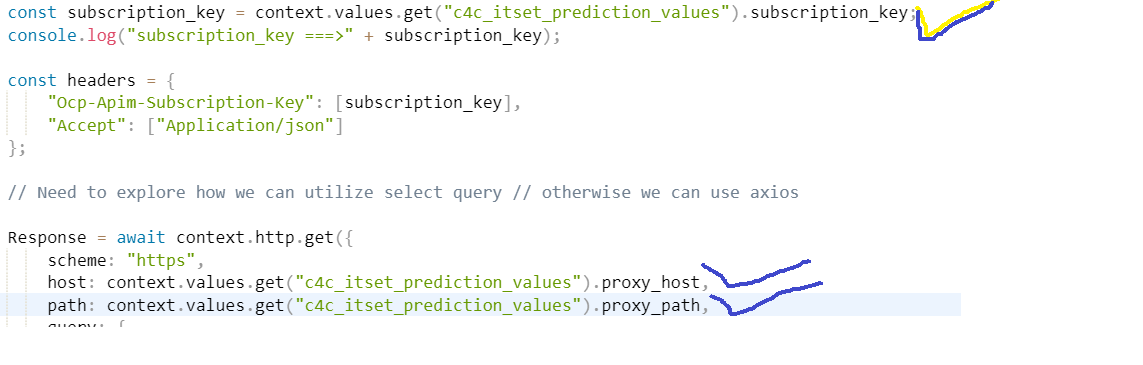
# Abstract

This document covers the key aspects to refactor the POC’s under C4C-SBS-Extensions Atlas MongoDB app services. Major issues with the current code are for all three different POC’s we have specific functions created for update, get, patch and execute rule function. Whilst refactor the code the overwhelming number of functions should be able to reduce, and the function can make very generic by-passing right arguments and configurations. This document describes how the different functions can be standardized and make it generic. It also covers the configurability aspects to fetch values from a configuration value store available in MongoDB Atlas app services.

# Code configurability

The code must be configurable. It should not contain any hardcoded values. All environment specific values must be stored in configuration value store in Atlas MongoDB app services. The secrets also must be stored and used inside the code, but it can be fetched from environmental specific secrets.

Few examples are shown below.



# Function code refactor approach

1. **Changes to get method.**

* The get method will be renamed to c4c\_customer\_serviceticket\_get
* Need to include parameters ticket\_root\_entityIdIn and queryParamater.
* Inside query section of get call directly use the queryParameter.
* Include more logging for the request data comes in.
* Use configuration values and secrets to get values of subscription, baseUrl, authorizationKey etc.
* Change the return statement to **return c4cResponse.d.results[0];**

1. **Changes to patch method.**

* The patch method will be renamed to **c4c\_customer\_serviceticket\_patch**
* Need to include parameters ticket\_root\_entityIdIn, patchBody, message, patchResponse = {}
* Remove the axios library usage and implement the context.http.patch.
* Use configuration values and secrets to get values of subscription, baseUrl, authorizationKey
* Form the message parameter values and append to the statusMessage constant. After that assign the statusMessage to the patchResponse.Body response.

1. **Changes to executerule method.**

* The get method will be renamed to c4c\_execute\_rule
* Need to include parameters request, response and query, projections.
* Form the input data from the request object like below which is dynamic.

let inputData = {};

if (data && data.attributes && data.attributes.length > 0) {

data.attributes.map((element) => {

if (element.name === 'CountryText') {

const ServiceRequestUsedAddress = request['ServiceRequestUsedAddress'];

console.log("ServiceRequestUsedAddress", JSON.stringify(ServiceRequestUsedAddress));

inputData[element.name] = ServiceRequestUsedAddress[element.name] ? ServiceRequestUsedAddress[element.name] : '';

} else {

inputData[element.name] = request[element.name] ? request[element.name] : '';

}

console.log("inputData", element.name, request[element.name]);

});

}

* Include necessary conditional logic to execute the rule engine with right set of parameters/input json.
* Execute the rule engine against the respond with execution status.

1. **Changes to update method.**

* We need basically for every POC an update function and its name must be c4c\_<feature name>\_update. Where features will be like itsetstatus, ispstatus etc.
* Use configuration values and secrets to get values of subscription, base URL, authorizationKey etc.
* Do the condition check specific to the POC, which is essential for which column we need to make the patch call.
* Form the patch\_Body, message, ticket\_root\_entityIdIn along with empty response object to execute the path method which is c4c\_customer\_serviceticket\_patch.

Finally tidy up unwanted functions and keep only required functions and configuration values.