C4C-SBS-Extensions Refactoring POC functions Approach

Contents

[Abstract 1](#_Toc110520691)

[Code configurability 1](#_Toc110520692)

[Function code refactor approach 1](#_Toc110520693)

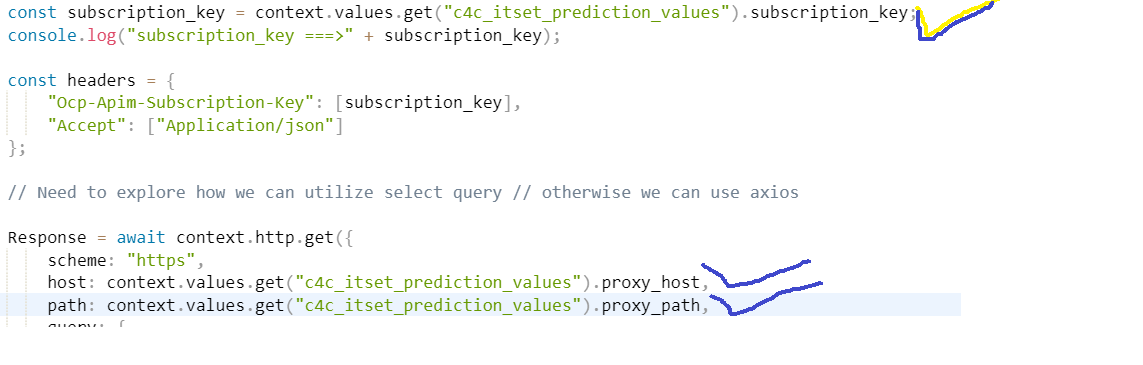
# Abstract

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 individual functions will be created specific to the POC’s. This document describes how the different functions can be standardized and make it specific to a feature. 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 a configuration value store in Atlas MongoDB app services. Similarly, the configuration that can be stored in database also fetched and used. 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

**Feature wise update function for every POC.**

* This will be the only function which takes care of get the ticket record from c4c APIM endpoint and execute the rule for the data availed and patch the delta changes back to c4c for a specific column depending upon a POC functionality.
* We need basically an update function for a POC, and its name must be c4c\_<feature name>\_update. Where features will be like itsetstatus, ispstatus etc.
* Use configuration values as well as database configuration options to get environmental specific variables/constants to use inside code. The subscription, base URL, authorizationKey database cluster name, query etc configurable items.
* Include meaningful logging message and log it for the request data comes in.
* Fire the context.http.get call to APIM get endpoint using ticket\_root\_entityIdIn value and for further JSON iteration use the **c4cResponse.d.results[0].**
* Form the input data from the **c4cResponse.d.results[0]** object like below which can be dynamic as below.

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]);

});

}

* Execute the rule engine against the input and respond with rule match found status.
* Execute the patch method using context.http.patch depending on the data to be patched for a POC when the rule match found.
* Once the patch operation succeeded make sure log the status based on the status of patch operation completed

|  |  |  |  |
| --- | --- | --- | --- |
| **Function Name** | **Parameters** | **Route** | **Psuedocode** |
| c4c\_claim\_type\_update | request, response | /c4c\_claim\_type\_update | 1. This will be the only function which takes care of get the ticket record from c4c APIM endpoint and execute the rule for the data availed and patch the delta changes back to c4c for a specific column depending upon a POC functionality.  2. Use configuration values as well as database configuration options to get environmental specific variables/constants to use inside code. The subscription, base URL, authorizationKey database cluster name, query etc configurable items.  3. Include meaningful logging message and log it for the request data comes in.  4. Fire the context.http.get call to APIM get endpoint using ticket\_root\_entityIdIn value and for further JSON iteration use the c4cResponse.d.results[0].  5. Form the input data from the c4cResponse.d.results[0] object like below which can be dynamic as below.  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]);  });  }  7. Execute the rule engine against the input and respond with rule match found status.  8. Execute the patch method using context.http.patch depending on the data to be patched for a POC when the rule match found.  9. Once the patch operation succeeded make sure log the status based on the status of patch operation completed |
| c4c\_market\_type\_update | Request, Response | /c4c\_market\_type\_update | 1. Validate the market\_type which is already set to 102 discontinue the entire operation, log the information about the status and skip the function execution.  2. This will be the only function which takes care of get the ticket record from c4c APIM endpoint and execute the rule for the data availed and patch the delta changes back to c4c for a specific column depending upon a POC functionality.  3. Use configuration values as well as database configuration options to get environmental specific variables/constants to use inside code. The subscription, base URL, authorizationKey database cluster name, query etc configurable items.  4. Include meaningful logging message and log it for the request data comes in.  5. Fire the context.http.get call to APIM get endpoint using ticket\_root\_entityIdIn value and for further JSON iteration use the c4cResponse.d.results[0].  6. Form the input data from the c4cResponse.d.results[0] object like below which can be dynamic as below.  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]);  });  }  7. Execute the rule engine against the input and respond with rule match found status.  8. Execute the patch method using context.http.patch depending on the data to be patched for a POC when the rule match found.  9. Once the patch operation succeeded make sure log the status based on the status of patch operation completed |
| c4c\_itsetstatus \_update | Request, Response | /c4c\_itsetstatus \_update | 1. Validate the, it status which is already set to Z6, discontinue the entire operation and log the information about the status and skip the function execution.  2. This will be the only function which takes care of get the ticket record from c4c APIM endpoint and execute the rule for the data availed and patch the delta changes back to c4c for a specific column depending upon a POC functionality.  3. Use configuration values as well as database configuration options to get environmental specific variables/constants to use inside code. The subscription, base URL, authorizationKey database cluster name, query etc configurable items.  4. Include meaningful logging message and log it for the request data comes in.  5. Fire the context.http.get call to APIM get endpoint using ticket\_root\_entityIdIn value and for further JSON iteration use the c4cResponse.d.results[0].  6. Form the input data from the c4cResponse.d.results[0] object like below which can be dynamic as below.  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]);  });  }  7. Execute the rule engine against the input and respond with rule match found status.  8. Execute the patch method using context.http.patch depending on the data to be patched for a POC when the rule match found.  9. Once the patch operation succeeded make sure log the status based on the status of patch operation completed |
| More POC’s to be defined | ---DO----- | Define routes | Define implementation of function. |

Finally after end-to-end testing ensure all the functions are working fine and tidy up unwanted functions and keep only required POC update functions and configuration values.