# Store Services

**Lambda Function**: BSROStoreFunction

**Project:** BSROStoreFunction

**DynamoDB Tables**: Store, StoreHours, AreaRegionDivision, GeoData

Store table is daily feed,

AreaRegionDivision is also feed where data will be moved in store feed.

GeoData table data will be updated 14 days once. If it’s not update some API will not return any value.

Data for Store hours will be moved daily basis from oracle to DynamoDB using utility.

**APIs:**

## *1. /ws2/ipinfo/ip*

In the background we are using a 3rd party call (h<ttp://api.quova.com/gppdirectory/v1/ipinfo/)> for getting the zip based on user IP address. Our web service is just a wrapper call.

There is no database communication for this services

## *2. /ws2/store/info*

This service is used to get the store details like store basic information, store holiday hour & store closing for the given store along with area, region and division details.

**Parameter:** storeType (Path param)

* By passing storeType, store details will be fetched from storesByStoreType-index in the Store table.
* By passing storeType, store hours details will be fetched from storeHoursByStoreType-index in the StoreHours table.
* Each store in the store table will have area id. By passing the area id hierarchy (area, region and division) details for each store will be fetched from AreaRegionDivision table. Area id is hash key in AreaRegionDivision table.

**Parameter:** storeNumber (Query param)

* For particular storeNumber store details will be fetched from Store table.
* For particular storeNumber store hours details will be fetched from StoreHours table.
* Each store in the store table will have areaId. By passing the area id hierarchy (area, region and division) details for each store will be fetched from AreaRegionDivision table.

## *3. /ws2/store/locator/geo/{latitude}/{longitude}/{siteName}*

This Service is used to get the available stores near the given latitude & longitude geo inputs.

**Parameter:**

* lat - Geo location
* lng - Geo location
* siteName - FCAC or TP or HT or WWT
* storeCount - The number of stores need to be listed in response, storeCount is passed as a query param.

The list of stores will be fetched for the particular siteName and between latitude (sum of latitude passed and 0.74 \* miles / 50.0 and difference of latitude and 0.74 \* miles / 50.0) and longitude (sum of longitude passed and 0.74 \* miles / 50.0 and difference of longitude and 0.74 \* miles / 50.0) that passed from storesBySiteName-index in Store table.

Here miles is 250. (i.e.) radius of 250.

Store hours details will be fetched from storesHoursBySiteName-index in the StoreHours table by passing the siteName and stored in the map based on storenumber as key.

For the list of stores distance is calculated as

Float radians = (float) (180/3.14159265);

Float miles = null;

if((lat1==lng1) && (lat2==lng2))

return 0.00f;

Float lt1 = lat1/radians;

Float lg1 = lng1/radians;

Float lt2 = lat2/radians;

Float lg2 = lng2/radians;

miles = (float) (3959\*Math.acos(Math.sin(lt1)\*Math.sin(lt2)+Math.cos(lt1)\*Math.cos(lt2)\*Math.cos(lg2-lg1)));

return miles;

The list of stores is sorted by distance that calculated. For store hours details for each store will be fetched from map that has storeHours detail.

## *4. /ws2/store/locator/zip/{zip}/{siteName}*

This Service is used to get the available stores near the given zip code.

**Parameter:**

* zip - zip is passed as query param. lat and lng will be retrieve for the zip by Bing request.
* siteName - FCAC or TP or HT or WWT.
* storeCount - The number of stores need to be listed in response, storeCount is passed as a query param.

Latitude and longitude is fetched for the zipCode from GeoData table in DynamoDB. If there is no data for particular zipcode passed then in the background we are using a 3rd party call (http://dev.virtualearth.net/REST/v1/Locations) to get latitude and longitude and we are saving these data in GeoData table.

**Note:** GeoData table will update latitude and longitude 14 days once. Internally it will call the same 3rd party service (http://dev.virtualearth.net/REST/v1/Locations).

The stores list will be fetched for the particular siteName and between latitude (sum of latitude passed and 0.74 \* miles / 50.0 and difference of latitude and 0.74 \* miles / 50.0) and longitude (sum of longitude passed and 0.74 \* miles / 50.0 and difference of longitude and 0.74 \* miles / 50.0) that passed from storesBySiteName-index in Store table.

Here miles is 250. (i.e.) radius of 250.

Store hours details will be fetched from storesHoursBySiteName-index in the StoreHours table by passing the siteName and stored in the map based on storeNumber as key.

For the list of stores distance is calculated as

Float radians = (float) (180/3.14159265);

Float miles = null;

if((lat1==lng1) && (lat2==lng2))

return 0.00f;

Float lt1 = lat1/radians;

Float lg1 = lng1/radians;

Float lt2 = lat2/radians;

Float lg2 = lng2/radians;

miles = (float) (3959\*Math.acos(Math.sin(lt1)\*Math.sin(lt2)+Math.cos(lt1)\*Math.cos(lt2)\*Math.cos(lg2-lg1)));

return miles;

The list of stores is sorted by distance that calculated. For store hours details for each store will be fetched from map that stores storeHours detail.

## *5. /ws2/store/locator/address/{siteName}*

**Parameter**:

* city -
* state - lat and lng will be retrieve for the address by Bing request based on this city and state.
* zip - lat and lng will be retrieve for the address by Bing request based on this zip code.
* siteName - FCAC or TP or HT or WWT.
* storeCount - The number stores need to be listed in response, storeCount is passed as a query param.

Neither City and State or zip code will be passed as parameter.

**Parameter:** City and State

Latitude and longitude is fetched for the city-state from GeoData table in DynamoDB. If there is no data for particular city-state passed then in the background we are using a 3rd party call (http://dev.virtualearth.net/REST/v1/Locations) to get latitude and longitude and we are saving these data in GeoData table.

**Parameter:** zipcode

Latitude and longitude is fetched for the zipcode from GeoData table in DynamoDB. If there is no data for particular zipcode passed then in the background we are using a 3rd party call (http://dev.virtualearth.net/REST/v1/Locations) to get latitude and longitude and we are saving these data in GeoData table.

Note: GeoData table will update latitude and longitude 14 days once. Internally it will call the same 3rd party service (http://dev.virtualearth.net/REST/v1/Locations) for city-state and zipcode.

The stores list will be fetched for the particular siteName and between latitude (sum of latitude passed and 0.74 \* miles / 50.0 and difference of latitude and 0.74 \* miles / 50.0) and longitude (sum of longitude passed and 0.74 \* miles / 50.0 and difference of longitude and 0.74 \* miles / 50.0) that passed from storesBySiteName-index in Store table.

Here miles is 250. (i.e.) radius of 250.

Store hours details will be fetched from storesHoursBySiteName-index in the StoreHours table by passing the siteName and stored in the map based on storeNumber as key.

For the list of stores distance is calculated as

Float radians = (float) (180/3.14159265);

Float miles = null;

if((lat1==lng1) && (lat2==lng2))

return 0.00f;

Float lt1 = lat1/radians;

Float lg1 = lng1/radians;

Float lt2 = lat2/radians;

Float lg2 = lng2/radians;

miles = (float) (3959\*Math.acos(Math.sin(lt1)\*Math.sin(lt2)+Math.cos(lt1)\*Math.cos(lt2)\*Math.cos(lg2-lg1)));

return miles;

The list of stores is sorted by distance that calculated. For store hours details for each store will be fetched from map that stores storeHours detail.

## *6. /ws2/store/services/{storeType}*

This service is used to get the store details like store basic information, store holiday hour & store closing for the given store along with area, region and division details.

**Parameter:** storeType (Path param)

Using storeType store details (storeNumber, activeFlag, onlineAppointmentActiveFlag, tirePricingActiveFlag, and eCommActiveFlag) will be fetched from storesByStoreType-index in the Store table.

**Parameter**: storeNumber (Query param)

For particular storeNumber store details (storeNumber, activeFlag, onlineAppointmentActiveFlag, tirePricingActiveFlag, and eCommActiveFlag) will be fetched from Store table.

## *7. /ws2/locator/get-state/{siteName}*

The service is used to retrieve the list of states for the particular site.

**Parameter:**

* siteName - FCAC or TP or HT or WWT.

By passing siteName, state codes for all stores will be fetched from storesBySiteName-index in the Store table.

State name for each state code is maintained in state.properties file. State name will be taken from this property file.

*Note: In code we are maintaining the state.properties file. If any modification has to be done in states, code change is required.*

## *8. /ws2/locator/get-city/{state}/{siteName}*

The service is used to retrieve the list of cities for the particular state.

**Parameter:**

* siteName - FCAC or TP or HT or WWT.
* state - State code.

By passing siteName and state code, cities for stores will be fetched from storesByState-index in the Store table.

# Tire Vehicle Widget Services

**Lambda Function**: BSROTireVehicleWidgetFunction

**Project:** BSROTireVehicleWidgetFunction

**DynamoDB Tables**: Fitment, FitmentYear

Both Fitment and FitmentYear table were Mitchel feed.

We are maintaining Model Year in separate table FitmentYear because if we going to fetch modelYear from Fitment table then we have to achieve it through Scan. Scan will take much time comparing to Query.

For year-index modelYear is hash key, to retrieve make by modelYear year-index is used.

For ym-index ym is hash key, to retrieve model by year and make ym-index is used ym attribute is combination of year+"-"+make (in lowercase)

For ymm-index ymm is hash key, to retrieve submodule by year, make and model ymm-index is used ymm attribute is combination of year+"-"+make+"-"+model (in lowercase)

For fitmentByAcesVehicleId-index acesVehicleId is hash key, to retrieve all the records for particular acesVehicleId, fitmentByAcesVehicleId-index is used.

**APIs:**

## *1. /ws2/widget/by-vehicle*

This service is used to get vehicles model Year. ModelYear will be fetched from FitmentYear table where distinct year is maintained.

## 2. */ws2/widget/by-vehicle/{year}/*

This service is used to get vehicle make id and names by year.

**Parameter**: year

By passing year all makes for particular year is fetched from year-index.

## *3. /ws2/widget/by-vehicle/{year}/{makeId}*

This service is used to get vehicle model id and names by year and make.

**Parameter**: year and makeId

By passing year, makeId as year+"-"+make (in lowercase) all model for particular year and make is fetched from ym-index.

## *4. /ws2/widget/by-vehicle/{year}/{makeId}/{modelId}*

This service is used to get vehicle sub model id and names by year, make and model.

**Parameter**: year, makeId and modelId

By passing year, makeId as year+"-"+make+"-"+model (in lowercase) all sub model for particular year, make and model is fetched from ymm-index.

# Tire Quote Services

**Lambda Function**: BSROTireQuoteFunction

**Project:** BSROTireQuoteFunction

**DynamoDB Tables**: TireQuote, Fitment

quoteId will be hash key for TireQuote.

**APIs:**

## *1. /ws2/tires/quote/create*

This service is used to create a tire quote. QuoteId is generated by concatenating millisecond + random number Between 1 to 10.

**Parameter:**

* storeNumber - store number for which the tire quote is created
* articleNumber - tire article number
* quantity - total number of tires
* acesVehicleId - ACES vehicle Id
* firstName - customer first name
* lastName - customer last name
* emptyCart - Boolean
* tpms - Boolean
* siteName - FCAC/TP/HTP/WW

Here the data will be saved as payload (JSON Structure)

## 2. */ws2/tires/quote/get*

This service is used to get tire quote for quoteId passed.

**Parameter**:

* quoteId – is generated by concatenating millisecond + random number Between 1 and 10.

By passing year all makes for particular year is fetched from year-index.

## *3. /ws2/tires/quote/update*

The service is used to update tire quote with cart details for the selected tire.

**Parameter**:

* quoteId – is generated by concatenating millisecond + random number Between 1 and 10.

Here the data will be saved as payload (JSON Structure)

# Alignment Services

**Lambda Function**: BSROAlignmentPricingFunction

**Project:** BSROAlignmentPricingFunction

**DynamoDB Tables**: Fitment, Store, AlignmentPricing, AlignmentPricingQuote

**APIs:**

## *1. /ws2/vehicle/tire/repair-alignment-pricing/get*

This service is used to create a tire quote. QuoteId is generated by concatenating millisecond + random number Between 1 to 10.

**Parameter:**

* storeNumber - To get alignment pricing for store number
* siteName - FCAC/TP/HTP/WW
* acesVehicleId - Vehicle Id (Not Mandatory)

If acesVehicleId parameter is passed, year, make, model, sub model for particular acesVehicleId is fetched from fitmentByAcesVehicleId-index of Fitment table.

If alignmentExclusion attribute in Fitment table is 1 then ‘No Alignment Options Found’ error is thrown, alignmentExclusion attribute is inserted at the time of Mitchell feed by referring AlignmentExclusion table.

Particular store details (storeNumber, storeType, areaId, zip) were queried from Store table.

Based on the areaId in store details Alignment pricing is filtered from AlignmentPricing table. The data in AlignmentPricing which has same areaId and where areaId null is selected.

## 2. */ws2/vehicle/tire/repair-alignment-quote/get*

This service is used to get tire quote for quoteId passed.

**Parameter**:

* quoteId – is generated by concatenating millisecond + random number Between 1 and 10.

Alignment pricing quote details for particular quoteId is fetched by passing the quoteId from AlignmentPricingQuote table.

## *3. /ws2/vehicle/tire/repair-alignment-quote/save*

The service is used to update tire quote with cart details for the selected tire.

**Parameter**:

* storeNumber - store number for which the tire quote is created
* articleNumber - tire article number which the boss prefers
* altype - alignment type
* alpricingId - alignment pricing id
* acesVehicleId - ACES vehicle Id
* firstName - customer first name (Not Mandatory)
* lastName - customer last name (Not Mandatory)
* siteName - site where request received from (FCAC/TP/HTP/WW
* emailId - customer mail id ) (Not Mandatory)

quoteId – is generated by concatenating millisecond + random number Between 1 and 10.

acesVehicleId is mandatory year, make, model, sub model for particular acesVehicleId is fetched from fitmentByAcesVehicleId-index of Fitment table. If there is no vehicle is found error ‘INVALID ACESVEHICLEID’ is thrown.

Particular store details (storeNumber, storeType, areaId, zip) were queried from Store table.

Based on the areaId in store details Alignment pricing is filtered from AlignmentPricing table. The data in AlignmentPricing which has same areaId and where areaId null is selected.

These information will be saved in AlignmentPricingQuote with new quoteId generated. Based on altype pricingName, article, price is stored.

# Battery Vehicle Services

**Lambda Function**: BSROBatteryVehicleFunction

**Project:** BSROBatteryVehicleFunction

**DynamoDB Tables**: InterstateAutomobileYear, InterstateAutomobile

Both InterstateAutomobile and InterstateAutomobileYear table are feed.

We are maintaining Model Year in separate table InterstateAutomobileYear because if we going to fetch modelYear from InterstateAutomobile table then we have to achieve it through Scan. Scan will take much time comparing to Query.

For year-index modelYear is hash key, to retrieve make by modelYear year-index is used.

For ym-index ym is hash key, to retrieve model by year and make ym-index is used ym attribute is combination of year+"-"+make (in lowercase)

For ymm-index ymm is hash key, to retrieve submodel by year, make and model ymm-index is used ymm attribute is combination of year+"-"+make+"-"+model (in lowercase)

**APIs:**

## *1. /ws2/vehicle/battery/options/year-make-model-engine/years*

This service is used to get batteries model Year. ModelYear will be fetched from InterstateAutomobileYear table where distinct year is maintained.

## 2. */ws2/vehicle/battery/options/year-make-model-engine/makes*

This service is used to get battery make by year.

**Parameter**: year

By passing year all model for particular year is fetched from ym-index.

## *3. /ws2/vehicle/battery/options/year-make-model-engine/models*

This service is used to get battery model by year and make.

**Parameter**: year and make

By passing year and make as year+"-"+make (in lowercase) all model for year-make is fetched from ym-index.

## *4. /ws2/vehicle/battery/options/year-make-model-engine/engine-sizes*

This service is used to get battery submodel by year, make and model.

**Parameter**: year, make and model

By passing year, make and model as year+"-"+make+”-”+model (in lowercase) all model for year-make-model is fetched from ymm-index.

# Battery Pricing Services

**Lambda Function**: BSROBatteryPricingFunction

**Project:** BSROBatteryPricingFunction

**DynamoDB Tables**: InterstateAutomobile, InterstateBatteryPricing, BatteryLife

**APIs:**

## *1. /ws2/vehicle/battery/get/search-results*

This service is used to get battery result for the year, make, model and submodel.

**Parameter**: year, make, model and submodel

By passing year, make and model as year+"-"+make+”-”+model+”-“+submodel (in lowercase) all battery details for year-make-model-submodel is fetched from ymmm-index. From battery result, unique productCode is taken.

From InterstateBatteryPricing table for particular products, pricing details have been retrieved. Battery result and pricing details have been mapped together.

The above battery pricing result will have product type. Battery results will grouped based on the product type. Random battery for each product type is selected. If there is no battery for that product type that product type will not be displayed.

## 2. */ws2/vehicle/battery/get/battery-life*

This service is used to get battery life for the zip code passed.

**Parameter**:

* zip – first 3 digits of the zip is used.

By using first 3 digit of zip, the batteryLifeAverage from BatteryLife table is fetched. By using batteryLifeAverage lifeYear, lifeMonth and lifeTotMonth is calculated.

## *3. /ws2/vehicle/battery/get/battery-quote*

The service is used to get battery quote for the particular quoteId.

**Parameter**:

* quoteId – is generated by concatenating millisecond + random number Between 1 and 10.

## *4. /ws2/vehicle/battery/save/battery-quote*

The service is used to create battery quote.

**Parameter**:

* siteName - web site name (FCAC/TP/HTP/WW)
* storeNumber - store number for which the battery quote is created
* productCode - battery productcode to save quote
* zip - zip code for which the battery quote is created
* year - vehicle year
* make - vehicle make
* model - vehicle model
* engine - vehicle engine

quoteId – is generated by concatenating millisecond + random number Between 1 and 10.

By passing product code product details were fetched from InterstateBatteryPricing table.

Based on the pricing result battery quote is stored as a JSON.