# Maximo Mobile Learning and Tips

# Introduction

This document will explore some of the development discoveries and tips for Maximo Mobile customisations.

# Maximo-Datasources

This section will cover some details of adding new fields and relationships to a maximo-datasource and datasource manipulation.

## Relationships

While adding a custom field to a maximo-datasource is simple, bringing a field from a relationship can be made with different forms.

### Sub-attributes

You can show the relationship field from the primary datasource by adding an attribute with the relationship name and sub-attributes with the related table fields.

<attribute name="serviceaddress" id="vqm4x">

<attribute name="streetaddress" id="m\_n7b"/>

<attribute name="addressline2" id="xjpby"/>

<attribute name="addressline3" id="wz84r"/>

<attribute name="city" id="zp\_ra"/>

<attribute name="regiondistrict" id="nkgv2"/>

<attribute name="stateprovince" id="y5vxe"/>

<attribute name="postalcode" id="zp8w6"/>

<attribute name="country" id="ve579"/>

<attribute name="formattedaddress--formattedaddress" searchable="true" id="exe\_8"/>

<attribute name="latitudey" id="nzbrw"/>

<attribute name="longitudex" id="ex9\_v"/>

<attribute name="description--svcaddressdesc" id="nbbxe"/>

<attribute name="saddresscode--saddresscode" id="y6835"/>

</attribute>

In the response JSON for this datasource, it will create a new object containing the attributes. For example:

{

wonum: "12234",

[...],

serviceaddress: {

streetaddress: "ZZZZZ",

[...]

}

}

This relationship must be a 1:1 relationship with the parent table.

### In line relationship

Another way to bring the fields from a relationship is using the inline mode.

<attribute name="serviceaddress.formattedaddress--formattedaddress" searchable="true" id="g7dgk"/>

<attribute name="serviceaddress.latitudey" id="g2xj2"/>

<attribute name="serviceaddress.longitudex" id="vjbxj"/>

In the same way, it must be a 1:1 relationship and will also produce a response with an object containing the fields from the related table.

### Field alias

We can use the --[name] in front of the attribute name to rename or to give an alias to that attribute and avoid conflicts with other parent attributes.

<attribute name="description" />

<attribute name="serviceaddress" id="vqm4x">

<attribute name="description--svcaddressdesc" id="nbbxe"/>

</attribute>

The example above shows how to use this feature to avoid conflicts with the parent description field.

The JSON output will also change. Instead of accessing this description attribute inside the serviceaddress object, it will be added as svcaddressdesc into the main JSON body.

{

wonum: "12234",

svcaddressdesc: "Service address description", // field added because of the alias name

[...],

serviceaddress: [

{

streetaddress: "ZZZZZ",

[...]

}

]

}

### Multi line relationship

The relationships that return more than one record will be added as an array on the JSON response and can be created as the below example.

<attribute name="rel.failurelist{failurelist,failurecode.description,failurecode.failurecode}" id="yx4jk"/>

<attribute name="rel.assetmeter{metername,active,rollover,lastreading,readingtype,lastreadingdate,measureunitid,meter.measureunit.description--unitdescription,sequence, pointnum}" id="zp6y9"/>

## Saved Queries and Where Clauses

The saved queries are created in the object structure used in a datasource and can be combined with a where clause.

<maximo-datasource-override id="todaywoassignedDS" saved-query="ASSIGNEDWOLIST" where="status!=&quot;DELAY&quot;" mobile-qbe-filter="{{'status\_maxvalue': '!=COMP,CAN,CLOSE,WAPPR'},{'status':'!=DELAY'}}" offline-immediate-download="true" default="true"/>

Note that the where clause will be used only on the Resource Application Roles and not on a device. To do so, we can also use the mobile-qbe-filter parameter, which will handle the filter as an object.

To create an and clause with multiple conditions, we can create as many objects as we need with the first JSON field as an existing field from the datasource. If the field is not in the schema, we must add it.

mobile-qbe-filter="{{'status\_maxvalue': '!=COMP,CAN,CLOSE,WAPPR'},{'status':'!=DELAY'}}"

## Synonym domains

Do not use a different saved query to create new data sources using the mxapisynonymdomain object structure. Otherwise, it will remove the standard query from the database generation.

Instead, you can change the MOBILEDOMAIN query, add your domain to the where clause, and override the datasource with a new where clause, a new saved query, or JavaScript.

Here is an example overriding the existing datasource with maximo-datasource-override and using a different saved query and a JavaScript Controller class. We can also use the where and mobile-qbe-filter, but not with the Synonym Domain, as it can contain other synonyms used across the application.

<maximo-datasource id="synonymdomainDS" lookup-data="true" object-structure="mxapisynonymdomain" offline-immediate-download="true" saved-query="MOBILEDOMAIN">

<schema id="bn686">

<attribute id="xp2e9" name="value"/>

<attribute id="g58pj" name="maxvalue" searchable="true"/>

<attribute id="wwwzq" name="description"/>

<attribute id="m2ebr" name="domainid" searchable="true"/>

<attribute id="aakrv" name="valueid" searchable="true" unique-id="true"/>

<attribute id="r27qk" name="siteid" searchable="true"/>

<attribute id="wz5gx" name="orgid" searchable="true"/>

<attribute id="avz96" name="defaults" searchable="true"/>

</schema>

<maximo-datasource-override controller="ReasonCodeDataController" id="reasoncodeDS" offline-immediate-download="true" saved-query="QUERY" selection-mode="single"/>

</maximo-datasource>

And the Controller Class:

class ReasonCodeDataController {

onDatasourceInitialized(ds, owner, app) {

this.datasource = ds;

this.owner = owner;

this.app = app;

}

async onAfterLoadData() {

this.datasource.setQBE('domainid', '=', 'RSNCODE');

await this.datasource.searchQBE();

}

}

export default ReasonCodeDataController;

We can also use the AppCustomizations.js to filter the datasource, which I prefer to use. Here is an example in AppCustomizations.js.

onBeforeLoadData(datasource, query) {

if (datasource.name === 'dsFailureList') {

let qbe = datasource.parseQBE({ orgid: `=${this.page.state.workorder.orgid}` });

query.qbe = { ...query.qbe, ...qbe };

}

if (datasource.name === 'synonymdomainData' && query.qbe?.domainid[0]?.value === 'LOGTYPE') {

// set default query if not being filtered by another search

if (!query.qbe?.value) {

let qbe = datasource.setQBE('value', 'in', ['CLIENTNOTE','UPDATE','WORK']);

query.qbe = { ...query.qbe, value: qbe };

}

}

}

Using AppCustomization.js, the onBeforeLoadData method will intercept all data sources before they start loading. So, we can change the query that will be used to fetch the data.

In the first condition, datasource.name === 'dsFailureList', we have a different data source, and we can intercept and change the query using a pre-existing function from the datasource class.

let qbe = datasource.parseQBE({ orgid: `=${this.page.state.workorder.orgid}` });

query.qbe = { ...query.qbe, ...qbe };

The parseQBE function will accept a JSON object.

INPUT: {siteid: "BEDFORD", assettype:"=PUMP", assethealth: ">20"}

For the second condition, datasource.name === ‘synonymdomainData' && query.qbe?.domainid[0]?.value === ‘LOGTYPE', we are intercepting a specific domain LOGTYPE, as the synonym domain datasource can contain several domains. The use of qbe?.domainid[0]? guarantees that any other datasource that doesn’t include a qbe query or this qbe query doesn’t have a domainid value won’t break the execution. Thus, we can change the data source and add a new condition to the original query.

// set default query if not being filtered by another search

if (!query.qbe?.value) {

let qbe = datasource.setQBE('value', 'in', ['CLIENTNOTE','UPDATE','WORK']);

query.qbe = { ...query.qbe, value: qbe };

}

In this specific case, to not always override the LOGTYPE domain, we check wheater we have a filter in the value field.

## JavaScript datasource manipulation

As mentioned above, we can use JavaScript to manipulate the data source data, and examples of it are using the method onBeforeLoadData or creating a custom Controller.

Another option is to manipulate the data after it is loaded. To do so, we use the method onAfterLoadData. Here is an example of using it with custom data sources, but it can be used with any.

Data source:

<maximo-datasource id="myalndomainsds" offline-immediate-download="true" lookup-data="true" object-structure="mxapialndomain" page-size="100" selection-mode="single">

<schema id="p3apw">

<attribute name="value" searchable="true" id="ggzba"/>

<attribute name="valueid" unique-id="true" id="mnxqw"/>

<attribute name="description" id="jqr6a"/>

<attribute name="domainid" searchable="true" id="jg4rn"/>

<attribute name="siteid" id="zbyr3"/>

<attribute name="orgid" id="ryqp4"/>

</schema>

<maximo-datasource-override id="resolutioncategoryds" offline-immediate-download="true" where="domainid=&quot;RESOLUTIOCAT&quot;" selection-mode="single"/>

<maximo-datasource-override id="restorecodeds" offline-immediate-download="true" where="domainid=&quot;RESTORECODE&quot;" selection-mode="single"/>

<maximo-datasource-override id="restorecauseds" offline-immediate-download="true" where="domainid=&quot;RESTORECAUSE&quot;" selection-mode="single"/>

</maximo-datasource>

JavaScript:

async onAfterLoadData(datasource, item, query) {

if (datasource.name === 'resolutioncategoryds') {

datasource.setQBE('domainid', '=', 'RESOLUTIOCAT');

await datasource.searchQBE();

} else if (datasource.name === 'restorecodeds') {

datasource.setQBE('domainid', '=', 'RESTORECODE');

await datasource.searchQBE();

} else if (datasource.name === 'restorecauseds') {

datasource.setQBE('domainid', '=', 'RESTORECAUSE');

await datasource.searchQBE();

}

}

Note that even with the where clause in the data source, the data won’t be filtered in the mobile device. So we can use the mobile-qbe-filter or JavaScript.