# Maximo JSON API-CRUD

~~V1 - Updated: 11/30/2015~~
~~V2 - Updated: 12/08/2015~~
V3 - Updated: 12/17/2015

## Contents

## *Introduction*

This document covers creating, updating and deleting Maximo data using the JSON API. You should review the Maximo JSON API Overview document prior to using this document.

In the APIMETA for MXASSET, the creationFactory provided a URL to support the creating/updating/deleting resources.

```
"creationFactory": [
   {
      "name": "default",
      "href": "http://host:port/maximo/oslc/os/mxasset"
   }
],
```

We will start by using this URL to create assets

## *Creating Resources (HTTP POST)*

There are different ways to use/test the API when creating/updating/deleting data.  With just a browser, you can add a REST Client plug-in that allows you to POST messages with HTTP Headers and a Body.

### Object Structures

For those familiar with using the MIF for other integration scenarios, the object structure is the core component that interacts with the Maximo Business Objects (MBOs).  Out of the box object structures, like MXASSET, include a processing class that enables successful creation and updating of assets.  If you configure new object structures, you may need to provide a processing class or automation script in order to successfully create or update the MBOs configured in the object structure.

## Creating an Asset

Using the URL provided in the creationFactory for MXASSET, you can create an asset, with minimal data, by providing this JSON data

```
http://host:port/maximo/oslc/os/mxasset?lean=1



{

"assetnum": "TEST200",

"status": "NOT READY",

"description":"Test Asset 200",

"siteid": "BEDFORD"

}
```

With the above JSON, you need to provide:

- the lean parameter if this is your initial request for a session, the api expects an OSLC namespace and you will likely get an error related to missing data for a field that is part of the primary key (siteid would be common).
- the HTTP header Content-Type with a value of application/json - this is required for all POST requests

The response will contain the following headers:

- Status Code: 201 Created
- Content-Language: en-US
- Content-Length: 0
- Date: Wed, 25 Nov 2015 15:39:24 GMT
- Etag: 1655563450
- Location: http://host:port/maximo/oslc/os/mxasset/_VEVTVDIwMC9CRURGT1JE
- X-Frame-Options: SAMEORIGIN
- x-powered-by: Servlet/3.0

The Location header will contain the link to the resource created which can be used to query or update the resource after creation. If you query the created asset you will see many more fields with values as the business object (MBOs) provides default values for many fields.

### HEADER:properties

If you want the response to include asset data in the body, you can provide the header named 'properties' with a value of *. This will return all the data for newly created asset (as if you queried for it). You can also choose to return selected fields in the response body by setting the properties header to a comma separated list of fields, such as assetnum, changedate, description. Using the properties header gives you access to all the data that was defaulted (by the MBOs) during the creation of the asset.

### Child Object data

The example below shows the creation of an Asset and an Assetmeter which is a child object in the MXASSET object structure:

http://*host:port*/maximo/oslc/os/mxasset?lean=1

```
{

"assetnum": "TEST200",

"assetmeter":[
{
  "metername":"TEMP-F",

  "linearassetmeterid":""
}
],

"status": "NOT READY",

"description":"Test Asset 200",

"siteid": "BEDFORD"

}
```

## *Updating Resources*

Updating a resource requires that you have the link to the resource which you can obtain with a query for the collection or from the location header when you created the resource.

### HTTP POST with x-method-override

To perform a resource update using a POST, you must provide the x-method-override header with a value of PATCH. Without this header, the processing will attempt to create the resource and will get an error due to the record already existing.

This URL identifies the specific asset resource:

```
http://host:port/maximo/oslc/os/mxasset/_VEVTVDYwMC9CRURGT1JE?lean=1
```

This JSON data for the body of the request will change the description of the asset:

```
{
"description":"New Description",
}
```

**PLEASE READ:**

The above update processes as a 'replacement' of the resource. One impact of this is for resources, like MXASSET, that have a parent object (Asset) and child objects (assetmeter,assetspect etc), you need to provide data for all of the child objects. Any objects that you do not provide their key value will be DELETED.

In the example above the asset that was updated with a new description would have had all of its related meters deleted since those were not provided in the request. This is the default behavior for updating. An alternative is to use an additional header called patchtype

## HTTP POST with patchtype

If you want to update a resource but do not want/not able to provide data for all the child objects, you can do an HTTP POST and provide the x-method-override header and the patchtype header to indicate that you want to patch, not replace.

Using the same URL and JSON data in the example above, if you perform a POST and provide the x-method-override property with a value of PATCH and the header patchtype with a value of MERGE, then the asset description will be updated but none of the child objects (assetmeter, assetspec etc) would be deleted. For those who have worked with the traditional MIF integration components, this is comparable to processing an inbound message with an Action of AddChange.

### Managing Child Object data

When you want to create/update child objects for a resource, in addition to providing x-method-override and patchtype headers, you can also provide actions for the individual child objects as part of the payload data.  Using actions will allow you to update an asset and let you create an assetmeter, update an existing assetmeter and delete an existing assetmeter when the _action is provided for each assetmeter (in a single transaction).

Using a URL for a specific asset

http://localhost:port/maximo/oslc/os/mxasset/_MTAwMS9CRURGT1JE?lean=1

The JSON body below would update the description of the asset, update an existing assetmeter object with metername PRESSURE, delete metername TEMP-F and add a new assetmeter, TEMP-C.

```
{
"description":"new description",
"assetmeter": [
   {
     "metername": "PRESSURE",
     "linearassetmeterid": 0,
     "newreading":"106",
     "_action":"Change"
   },
   {
     "metername": "TEMP-F",
     "linearassetmeterid": 0,
     "_action":"Delete"
   },
   {
     "metername": "TEMP-C",
     "linearassetmeterid": 0,
     "newreading":"29",
     "_action":"Add"
   }

]
}
```

## *Deleting Resources*

To delete a resource, use HTTP POST with the x-method-override header with a value of DELETE.

This URL identifies the specific asset resource to delete:

http://host:port/maximo/oslc/os/mxasset/_VEVTVDYwMC9CRURGT1JE

Within the MBOs, the delete processes through the business logic and there may be reasons why a delete is prevented. When this occurs an error will be returned noting that the resource cannot be deleted - example below:

```
{
  "Error": {
    "message": "BMXAA0106E - Cannot delete Asset TEST500 because it is referenced in the WORKORDER table.",
    "statusCode": "400",
    "reasonCode": "BMXAA0106E",
    "extendedError": {
      "moreInfo": {
        "href": "http://localhost:port/maximo/oslc/error/messages/BMXAA0106E"
      }
    }
  }
}
```

## *Rowstamp*

Maximo applications support the use of the rowstamp on a row of data to validate that the record being updated/deleted is 'current', meaning it has not been changed since it was retrieved. When it has been changed a user in an application see an 'updated by another user' message and be forced to retrieve the record again prior to updating it.

The API supports this same feature by providing an attribute called _rowstamp (with the rowstamp value) as part of the resource data when queried:

```
"_rowstamp": "26982",
```

When an update is requested and the _rowstamp (with its value) is provided, the same check will be done as described above. If the rowstamp of the Maximo record has changed then the API will return an error. If no _rowstamp is provided, then the update (or delete) would execute on the current record in the database with no check.

A _rowstamp is supported for each record (all levels) in the resource data (i.e. for asset, assetmeter, assetspec etc)

## *Attachments*

The API supports the creation and deleting of attachments that are associated to resources. For example, you created an asset and now you want to attach a PDF file that describes the maintenance procedures for that asset. There is no support for updating an attachment, you would need to delete the current version and create a new version.

Note: see the related API Query document regarding the API's support for querying attachments associated to a resource.

For a resource, such as MXASSET, to support attachments:

- the Maximo attachments feature has to enabled
- the mxasset object structure must be configured with the DOCLINKS MBO as a child to the ASSET object.

When you query a single asset (using mxasset) you will get a doclinks URL that you use to the associate an attachment to the asset. The URL would look like this in the asset JSON data:

```
"doclinks":
{
   "href": "http://localhost:port/maximo/oslc/os/mxasset/_MTAwMi9CRURGT1JE/doclinks"
},
```

Note: in the current version of the API you can create an attachment for a resource(asset) only after the resource exists in Maximo. You cannot create the attachment at the time of creating the resource.

An attachment is made up of two components, the attachment file and the related metadata of the attachment. You create an attachment using HTTP POST with binary content or base64 binary content. There is no support for multi-part messages.

When creating an attachment for a resource there is a limited set of metadata that can be provided (along with the file) using HTTP Headers:

| Header | Value | Description |
|---|---|---|
| slug | File Name | The name of the attachment file |
| encslug | File Name | If the attachment file name has non-ascii characters it can be provided in the header base64 encoded. It is suggested that you always base64 encode your file name using this property if you believe you might have a mix of non-ascii characters |
| Content-Type | "text/plain" | Based on the type of attachment - text/plain supports a .txt file |
| x-document-meta | Attachments | Tied to the DOCTYPES domain that defines the supported attachment types |
| x-document-description | Description | The description of the document |
| x-document-encdescription | Description | If the description has non-ascii characters it can be provided in the header base64 encoded. It is suggested that you always base64 encode your description using this property if you believe you might have a mix of non-ascii characters |
| custom-encoding | "base64" | This header facilitates testing using a browser client such as RESTClient (for FF). Allows you to paste in a base64 encoded image into the Body of the tool (otherwise you need to test with programmatic tool). You can use public tools to base64 encode your image file |

You post to the URL (shown above) with the attachment, when successful the response will have a header named Location and it will contain a URL to the attachment file:

http://localhost:port/maximo/oslc/os/mxasset/_MTAwMS9CRURGT1JE/doclinks/80

Using that URL in the browser will return the file. Using that URL in the RESTClient tool will return a header named Link which will have a URL to the metadata for this attachment. URL looks like this:

http://localhost:port/maximo/oslc/os/mxasset/_MTAwMS9CRURGT1JE/doclinks/meta/80

These URLs are also available when querying the resource (see the related API Query document which has a section on Attachments).

When a resource attachment is created through the API, you should be able to go to the related application (Asset) and View Attachments to verify the attachment was successfully associated to the asset.

}

Deleting an Attachment

Using HTTP POST with the URL of the attachment:

http://localhost:port/maximo/oslc/os/mxasset/_MTAwMS9CRURGT1JE/doclinks/80

and providing the x-method-override Header with a value of DELETE will remove the attachment from the Asset resource.

## *Bulk Processing*

The API supports the processing of multiple resources in a single transactions. Using the x-method-override header with a value of BULK, directs the processing to process multiple resources provided in the message body within a JSON array [ ]. The example below shows 3 assets being created using the mxasset collection url (Bulk processing only supported using collection url)

http://localhost:port/maximo/oslc/os/mxasset

```
[
 {
  "_data":{
   "assetnum": "test-5",
   "siteid": "BEDFORD",
   "description": "TS test 5"}
 },
 {
   "_data":{
   "assetnum": "test-6",
   "siteid": "BEDFORD",
   "description": "TS test 6" }
 }
]
```

Each resource has an element called _data (reserved name) in which the data for the resource is provided. Unless there is a syntax type of error in your JSON data, you will always get a response code of 200. However, you need to process the response to determine which resources were updated successfully or not. If the second asset failed due to an invalid site the response body would look like this:

```
[
 {
   "_responsemeta": {
     "ETag": "1992365271",
     "status": "201",
     "Location":
"http://localhost:port/maximo/oslc/os/mxasset/_VEVTVC0zNS9CRURGT1JE"
   }
 },
 {
   "_responsedata": {
     "Error": {
       "message": "BMXAA4153E - [BEDFORDXXYY is not a valid site. Enter a valid Site
value as defined in the Organization Application.]",
       "statusCode": "400",
       "reasonCode": "BMXAA4153E",
       "extendedError": {
         "moreInfo": {
           "href": "http://localhost:port/maximo/oslc/error/messages/BMXAA4153E"
         }
       }
     }
   },
   "_responsemeta": {
     "status": "400"
```

```
      }
    },
    {
      "_responsemeta": {
        "ETag": "1992365297",
        "status": "201",
        "Location":
"http://localhost:port/maximo/oslc/os/mxasset/_VEVTVC0zNy9CRURGT1JE"
      }
    }
]
```

The first and third assets were successful and returned a 201 with the URI to the asset resource. The second asset returned a 400 with an error message.

NOTE: The processing performs a Commit for each resource.

## Multiple Operations with BULK

The above examples showed the creation of multiple assets using the BULK processing. You can also use BULK to perform a mix of create, update and delete of asset resources in a single transaction.

To support this, in addition to _data which is used to provide the json data for a resource in a BULK transaction, you can also provide meta data using _meta (another reserved name). The meta data that can be provided are:

- method
- uri
- patchtype

The is the equivalent of the x-method-override header discussed earlier in this document. When POSTing to create a new resource, there is need to provide a method. To perform an update you provide the value PATCH and for a delete you provide the value DELETE.

The uri is the resource uri when processing an Update or Delete. This is required when updating or deleting a resource.

The patchtype allows the support of MERGE when processing an update (as described earlier in this document).

Below is an example JSON data that will

- Update an asset with a 'New Description'
- Create a new asset (test-100)
- Delete an asset

```
[
{
  "_data":{
     "description": "New Description"},
   "_meta":{
   "uri":"http://localhost:port/maximo/oslc/os/mxasset/_VEVTVC0yL0JFREZPUkQ-",
   "method": "PATCH",
   "patchtype":"MERGE"}
  },
 {
  "_data":{
   "assetnum": "test-100",
   "siteid": "BEDFORD",
   "description": "New Asset 100"}
 },
 {
   "_meta":{
   "uri":"http://localhost:port/maximo/oslc/os/mxasset/_VEVTVC00L0JFREZPUkQ-",
   "method": "DELETE" }
 }
]
```

The first asset 'meta' data includes the URI to identify the along with headers identifying that it is a Patch (an update) with a type of Merge.

The second asset provides no 'meta' data since it is a Create and no meta data is applicable.

The third assets provides only the 'meta' data to identify the asset to be deleted.

As with the Creation example earlier in this section, the response code will be a 200 but you must examine the response information in the response JSON body to determine if processing of each asset was successful or not.

## *Duplicate Transaction Processing*

When a client invokes the API to create/update/delete a resource there is the possibility that the resource is updated in Maximo but the client fails to get a response back confirming the update, due to a technical issue such as dropped connection.  This could lead to the client resending the transaction and possibly duplicating the transaction processing.

The client can avoid this situation by providing a header named 'transactionid' and providing a unique ID value with each transaction.  If the client hits an error and resends a transaction, the API framework will check if the ID was previously processed, and if so will return a 409 response without updating the resource. The client must be coded to process accordingly when a 409 response is returned.

The IDs are maintained in the database and an escalation named, OSLCTXNCLEANUP, will delete entries using an Action of the same name.  The parameter in the Action identifies the number of days an ID will be maintained before it is deleted.  The default parameter value is 5 days.  You should configure this based on how your client processing is generating/managing ID values used for API transactions.

## *Invoking Actions using the API*

In Maximo version 7.6.0.3 additional capability was provided to support invocation of actions on a resource.  Maximo has a limited set of 'out of the box' available to invoke using the API and there is now capability to define custom actions using an automation script. See the related tech note for more information:

*http://www.ibm.com/support/docview.wss?uid=swg21972876*

Please send any corrections or suggestions to Tom Sarasin at tsarasin@us.ibm.com