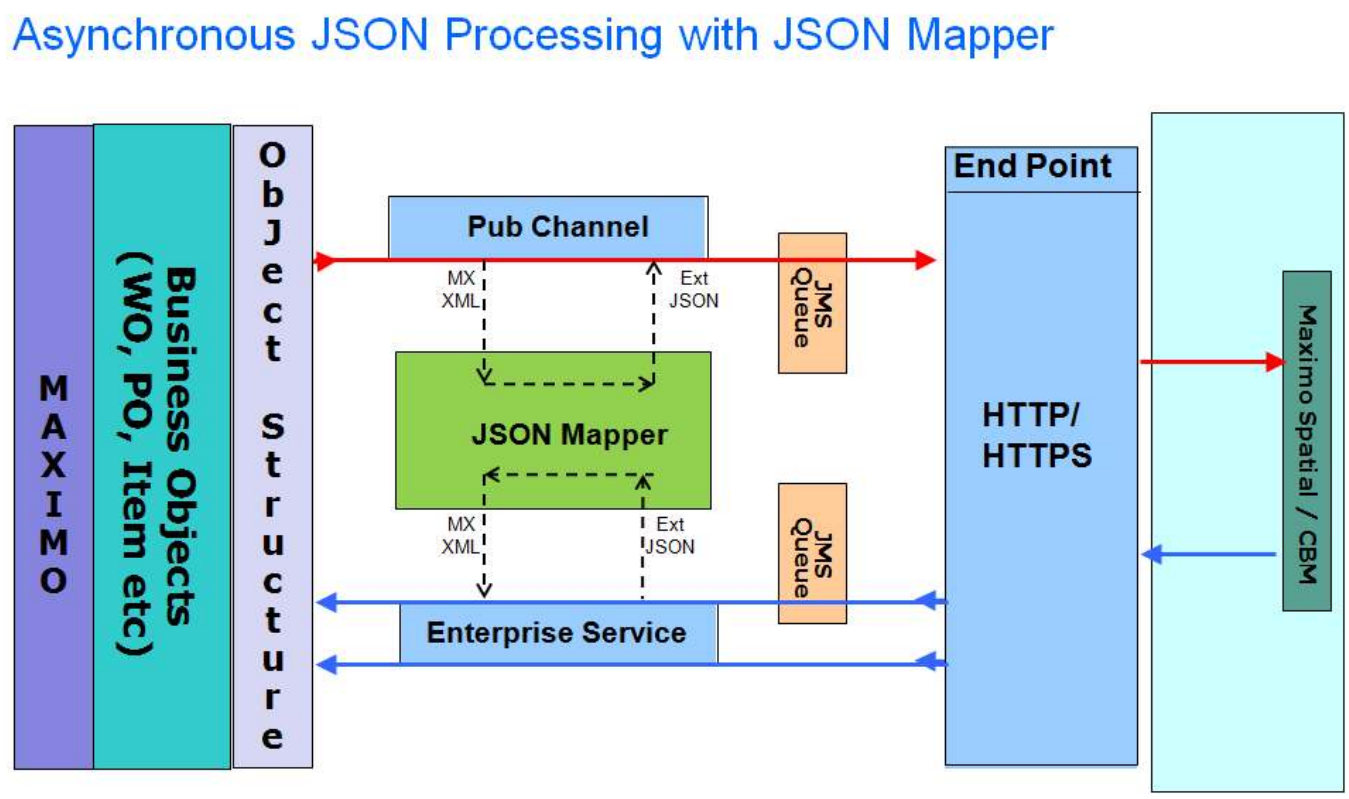
JSON Mapping

General Approach

The MIF will provide a pluggable 'mapper' at the Publish Channel and Enterprise Service layers that will allow an implementer to convert JSON to Maximo XML and Maximo XML to JSON. The mapper will be deployed using the External Exit in either the Publish Channel or Enterprise Service, which allow the support of asynchronous processing via the JMS queues. The JSON mapper will be comparable to the existing use of Java/Scripts/XSL that allow customization processing on an integration message (inbound or outbound).



In order to provide this capability, there will be enhancements provided within the Integration Framework:

- JMS queues will support JSON data (in addition to XML) - both types of messages can be stored in the same JMS queue.
- Message Reprocessing will support JSON data

- Message Tracking will support JSON data
- View/Delete Queue utilities will support JSON data
- JMS CRON tasks selectors will operate on JSON Data
- JMS support for 'Text format' will support JSON data
- A new JSON Mapping application will be provided
- Publish Channels and Enterprise Services will support the implementation of a JSON Map

JMS Queues and related processing

The JMS queues will support the storing and subsequent processing of an integration message in JSON format for either a Publish Channel or Enterprise Service. The loading of inbound JSON data into a queue will be supported using either HTTP/HTTPS or a direct update via JMS (no support for loading of JSON data using Interface Tables, Web Services or from a file). An individual JMS queue will support messages in both XML and JSON format. The MDB and JMS CRON tasks will support the use of Selectors against queues that have JSON data.

The View Queue and Delete Queue that allows viewing or deleting of data in the queues will support messages in a JSON format.

A Queue defined to support 'Text Format/Text Message Encoding' will support JSON data.

Message Tracking will be supported for Channels and Services that have a JSON Map implemented. When data is configured to be saved it will be in JSON format. The Search and External Message IDs are supported using the following format:

$.features.attributes.OBJECTID (note: values are case sensitive)

where features and attributes are JSON objects.

Message Reprocessing will be supported for Channels and Services (when a JSON Map implemented) that have messages that go in error. The Internal and External message view will support both XML and JSON formatted data.

Enterprise Service (ES)

A JSON Map can be used with an Enterprise Service (inbound process) to accept a message in a JSON format and convert it to 'Maximo XML' in order for the message to be processed into Maximo. The use

of the Map can be implemented when the ES is invoked in an asynchronous model where the message is persisted to an inbound JMS queue or when ES is invoked in a synchronous model (no queue).

The invocation of the JSON Map occurs when the Processing Class (external exit) of the ES is executed.

When an Enterprise Service supports a JSON Map, the Processing Class (external exit) must be com.ibm.tivoli.maximo.fdmbo.JSONMapperExit, or a class that extends this class. This processing class is needed for both the request side and the response side (when response is used in a synchronous approach).

The JSON Map identifies the Enterprise Service based on the Map Name assigned:

1. A JSON Map is defined using a naming convention that identifies what component it belongs to. The name format would be
   - ExternalSytem.EnterpriseService.IN

     ex: EXTSYS1.MXASSETInterface.IN
   - ExternalSytem.EnterpriseService.RESPONSE

     ex: EXTSYS1.MXITEMInterface.RESPONSE

   When a message is processed through the Enterprise Service the framework will invoke the registered JSON Map as part of the request (IN) or RESPONSE.

When calling an Enterprise Service with JSON, the MIME type of application/json must be provided in the Content-Type header. Records added directly into the JMS queue require a JMS Header of mimetype with a value of 'application/json'

Publish Channel (PC)

A JSON Map can be used with an Publish Channel (outbound process) to convert an outbound message from 'Maximo XML' format to a JSON format. A map can be applied to a Channel pushing data out (from an event or Data Export) and when a Channel is fired on a 'recursion' basis when the event is triggered from an inbound integration message.

The invocation of the JSON Map occurs when the Processing Class (external exit) of the PC is executed.

When a Publish Channel supports a JSON Map, the Processing Class (external exit) must be com.ibm.tivoli.maximo.fdmbo.JSONMapperExit, or a class that extends this class.

The JSON Map for a PC is defined using a naming convention that identifies what component it belongs to. The name format would be

- ExternalSytem.PublishChannel.OUT

  ex: EXTSYS1.MXASSETInterface.OUT

- ExternalSytem.EnterpriseService.RESPONSE

  ex: EXTSYS1.MXITEMInterface.RESPONSE

  When a PC event is fired from the processing of an inbound integration message (recursion), the JSON map named is identified with a 'RESPONSE' in the name since the recursion transaction is a response to inbound integration transaction.

  Note: a customization (java/scripting) using the Event Filter Class is needed to enable recursion processing

PC Messages in a JSON format that are placed into the outbound queue can be processed from the queue using an HTTP-based End Point, JMS-based End Point or XML File end point. The XML file End Point will write a file in JSON format (not XML).

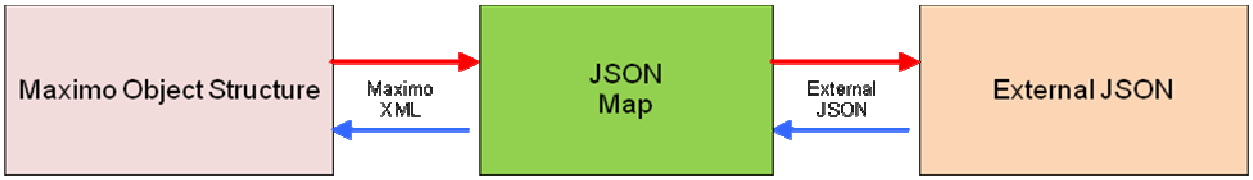
Invocation Channel

No support for a JSON Map with an Invocation Channel.

JSON Mapper Application

A new integration application, JSON Mapping, will be provided for the creation and configuration of a JSON Map that can be enabled for use with integration message processing (as mentioned above). This application supports the creation, testing and deletion of JSON Maps.

The JSON Mapping application will allow an integrator to select an integration object structure and supply a sample JSON snippet from which a 'mapping' will be created to map JSON data to Maximo XML (and vice versa).



The Mapping application can leverage a provided URL or Integration HTTP End Point to retrieve the JSON data as part of the configuration. The sample JSON may also be manually entered. Once the JSON is supplied, the integrator can begin mapping JSON data to the objects and attributes of the selected object structure.

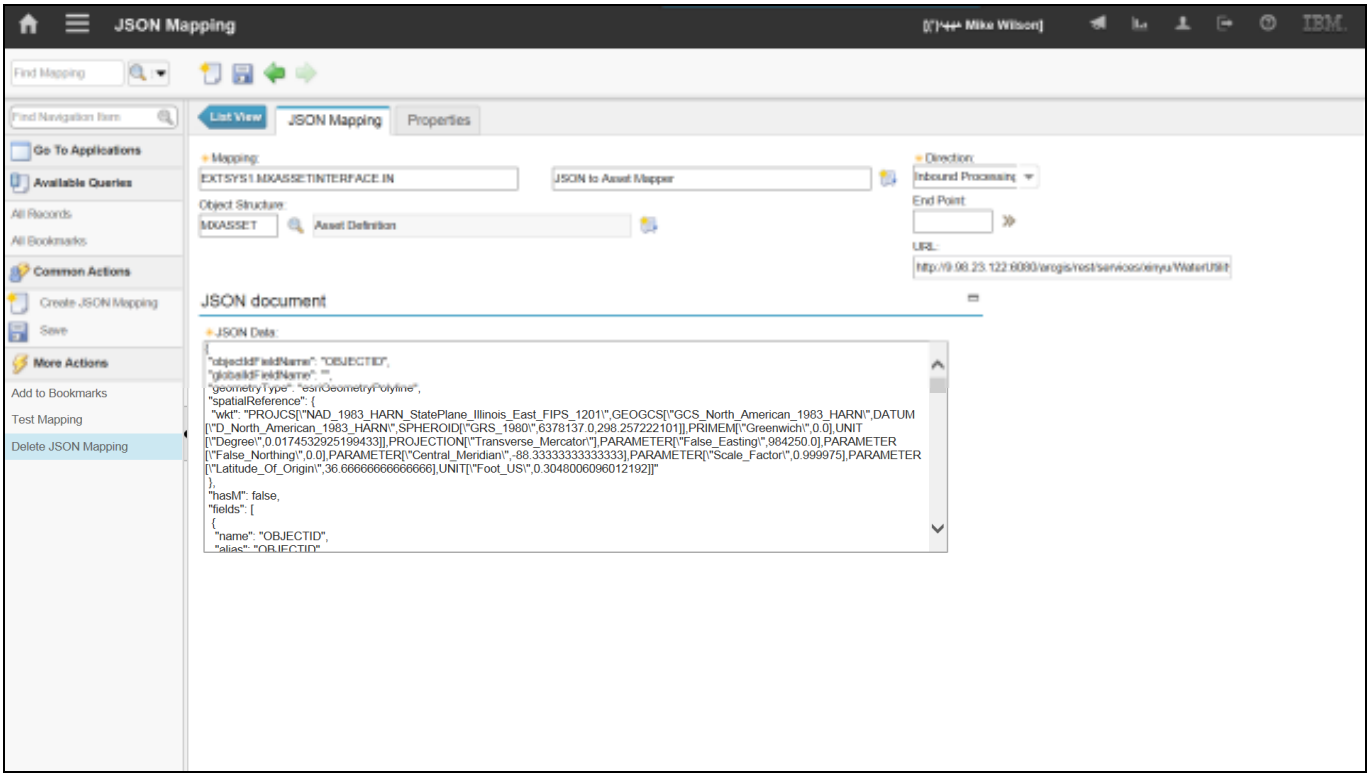
The application provides a 'Test' action to allow an integrator to provide JSON or XML data into the 'mapper' and view the output JSON or XML data that was produced using the Map.

Creating a JSON Map

When creating a JSON Map for use with a Publish Channel or Enterprise Service you can link the map to your channel or service by following a defined naming convention for the map (see the earlier sections above for the naming convention details.

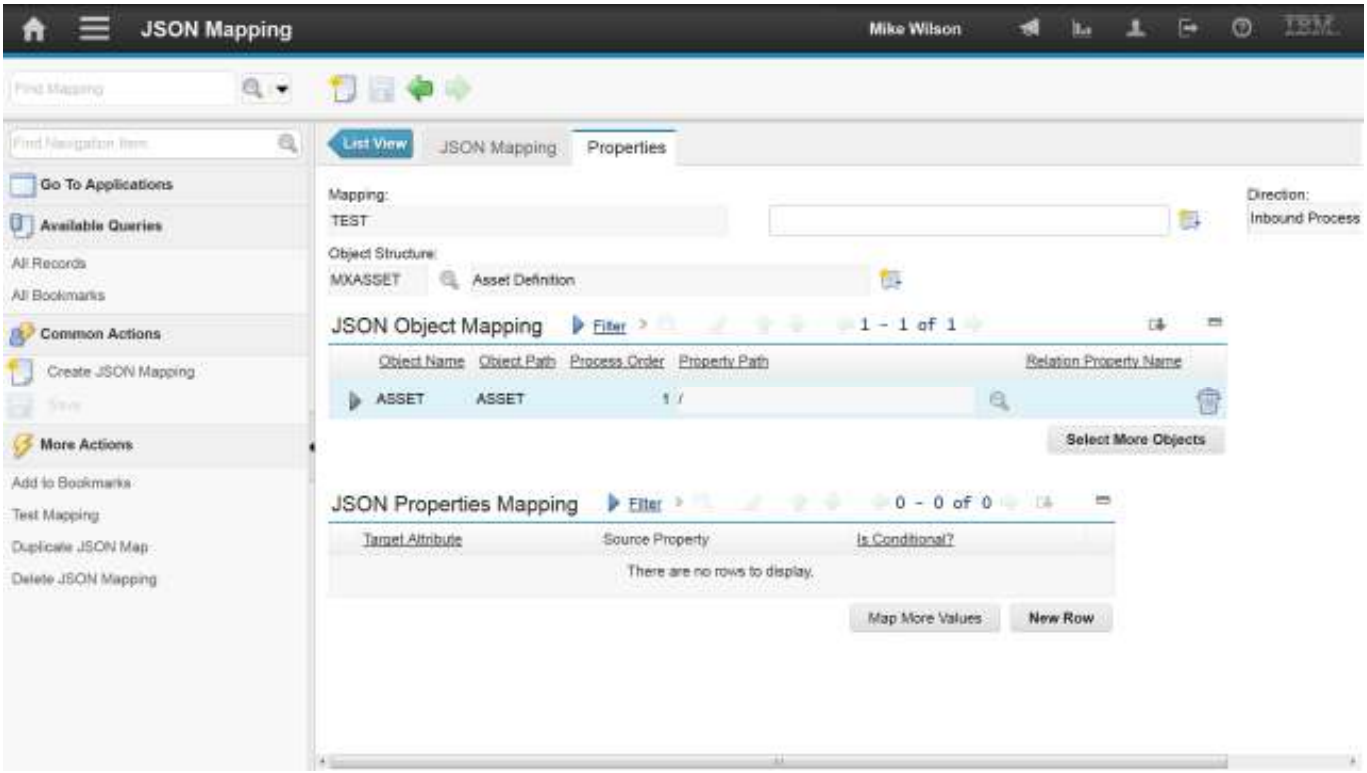
In addition to the Name and Description, you select Direction, either Inbound for an Enterprise Service or outbound for a Publish Channel. You are required to select an Object Structure (type INTEGRATION or OSLC) and then you can enter either an existing Integration End Point (handler type of HTTP) or a URL. The End Point or URL should be able to access your sample JSON and will automatically populate the JSON data section. You may also cut and paste sample JSON data directly into the JSON Data field

Note: when cutting and pasting JSON data, depending upon the source, (from a browser window, an email etc), you may carry extra/hidden characters which might give validation errors. It is suggested that you paste your JSON into a Text editor such as Notepad and then copy and paste from there into the Maximo application.



Once the Map is saved, you can then proceed to the Properties tab of the JSON application and begin mapping your JSON data to the objects and attributes that are contained within the object structure you selected.

The Properties tab provides two tables where the parent (top) table supports the mapping of objects and the child (lower) table is for the mapping of the attributes for those objects. The lower table content is driven by the 'in-focus' row (objects) in the upper table.



The mapping of the JSON data to the object structure requires an understanding of JSON syntax, as well as, the data content provided in the sample JSON within the JSON Mapping tab.

By default the root object in the object structure (ASSET in the screen shot above) will be initially mapped to the root of the JSON data, identified by a single slash (/).

Below is a sample JSON file and it is made up of JSON Objects, Arrays and Simple Types. Objects are identified with a { } and Arrays are identified with [ ] . Simple Types may exist within either.

SAMPLE JSON

```
{
    "clientId": "ABC Co",
    "typeId": "PUMP",
```

```
 "deviceId": "X1237",
 "deviceInfo":
 {
    "serialNumber": "AEI12AW387",
    "manufacturer": "XYZ Co",
    "model": "H-100",
    "deviceClass": "Centrifugal",
    "description": "Centrifugal Pump 100GPM/60FTHD",
    "descriptiveLocation": "BR450"
 },
"metadata":
{
"asset": "111321",
"siteid": "BEDFORD"
},
  "monitor":
     {
        "measuretype": [
        {
           "id": "TEMP-F ",
           "desc": "Temperature Fahrenheit"
        },
       {
           "id": "TEMP-C",
           "desc": "Temperature Celsius"
        }
       ],
     },
     "status":
     {
        "alert":
        {
```

```
            "enabled": false,
            "timestamp": "2015-10-13T18:39:34.580Z"
          }
        },
}
```


In the example above, there are four JSON objects under the root level:

- deviceinfo { }
- metatdata { }
- monitor { }
- status { }

In addition there are simple types at the root level as well: clientID, typeId, deviceId.

Within the status object there is an object named 'alert' and within the monitor object there is an object name 'measuretype' that is an Array [ ].

A hierarchical view of the objects in this JSON data would look like this:

/

deviceinfo { }

metatdata { }

monitor { }

measuretype{ }

measuretype{ }

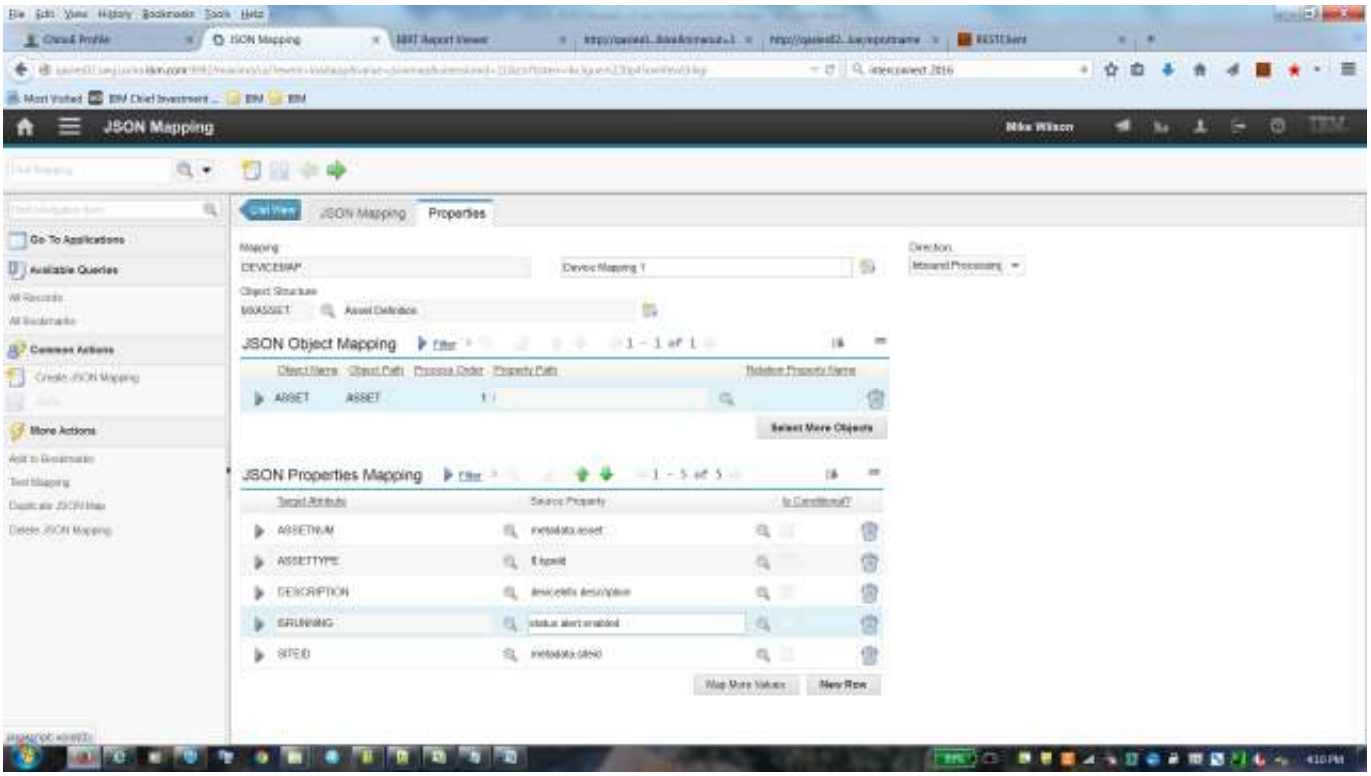
status { }

alert{ }

The object and properties mapping has a limited set of validation to allow the implementer full flexibility to map the JSON data to Maximo XML (as defined by the object structure). With this in mind, it is the responsibility of the implementer to perform a 'logical' mapping based on the data structure of the JSON data to create valid XML message that can be processed into Maximo.

Mapping from the root level

If it is logical to map all the data to just the Asset object within the MXASSET object structure then your mapping can be done at the 'root' level as shown below. A dot (.) notation is used to traverse down a hierarchy of JSON objects (ex: status.alert.enabled).



In the mapping above, there was no mapping to monitor.measuretype data since that is an array. If an array was mapped there could have been multiple values attempted to be mapped to a single asset. This would have resulted in multiple assets in the XML structure created from the JSON data.

In addition to setting a property value to populate a Target Attribute, you can also use Javascript (following standard JavaScript syntax) to provide a value or provide a literal value such as "ABC". You can concatenate two source properties separated by a dash using this format: *$.field1+"-"+$.field2*

Mapping from multiple levels of the JSON data

In order to map from a level of JSON data to an object structure, each parent level in the JSON data must also be mapped as an Object, even if none of the fields at that level are being mapped. For example the JSON data has these levels:

/

deviceinfo { }

metatdata { }

monitor { }

measuretype{ }

measuretype{ }

status { }

alert{ }

In order to map data from the deviceinfo{} object, the root level must first be mapped to an object. You could map:

ASSET to / (root)

ASSET to deviceinfo{}

with this mapping you could then map fields from deviceinfo{} to fields of the ASSET object.

Likewise if you wanted to map fields from the measuretype{} object to ASSETMETER, the mapping must have

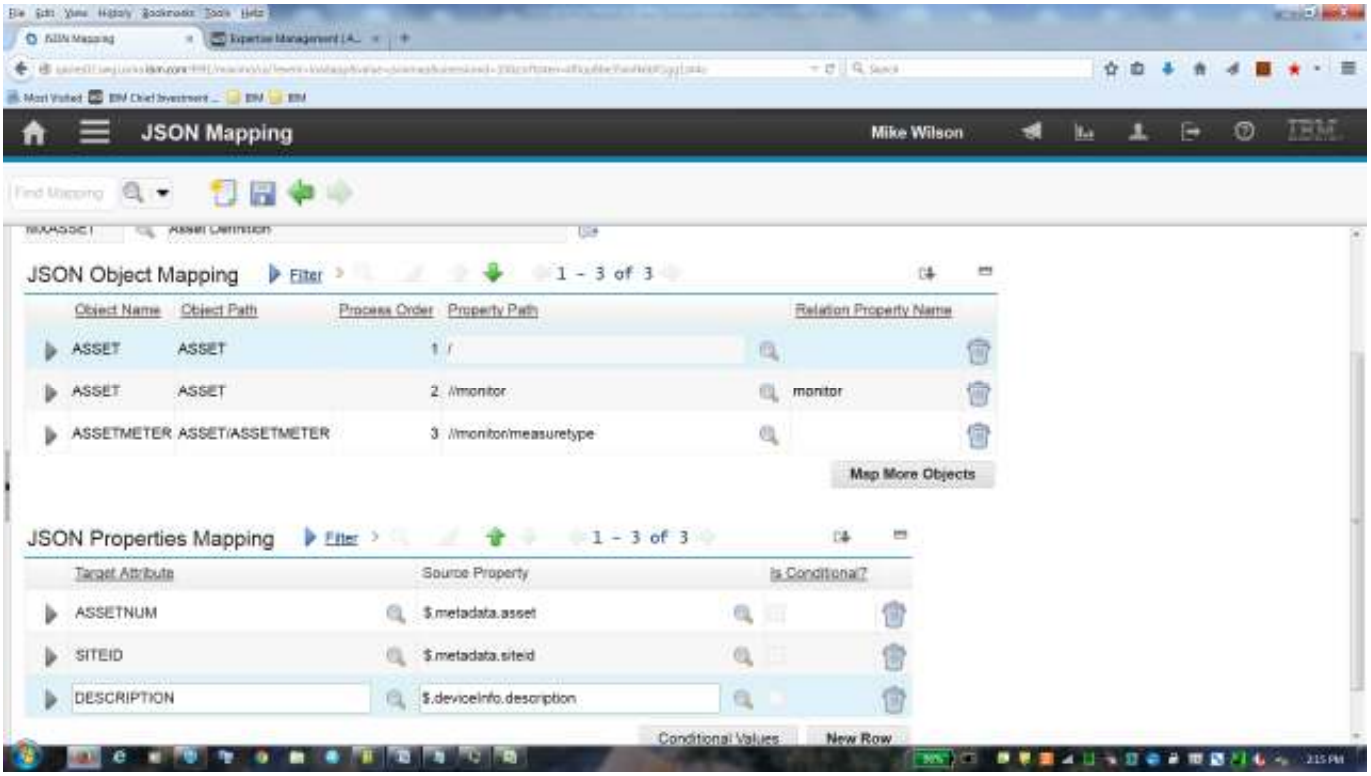
ASSET to / *(root)*

ASSET to monitor{}

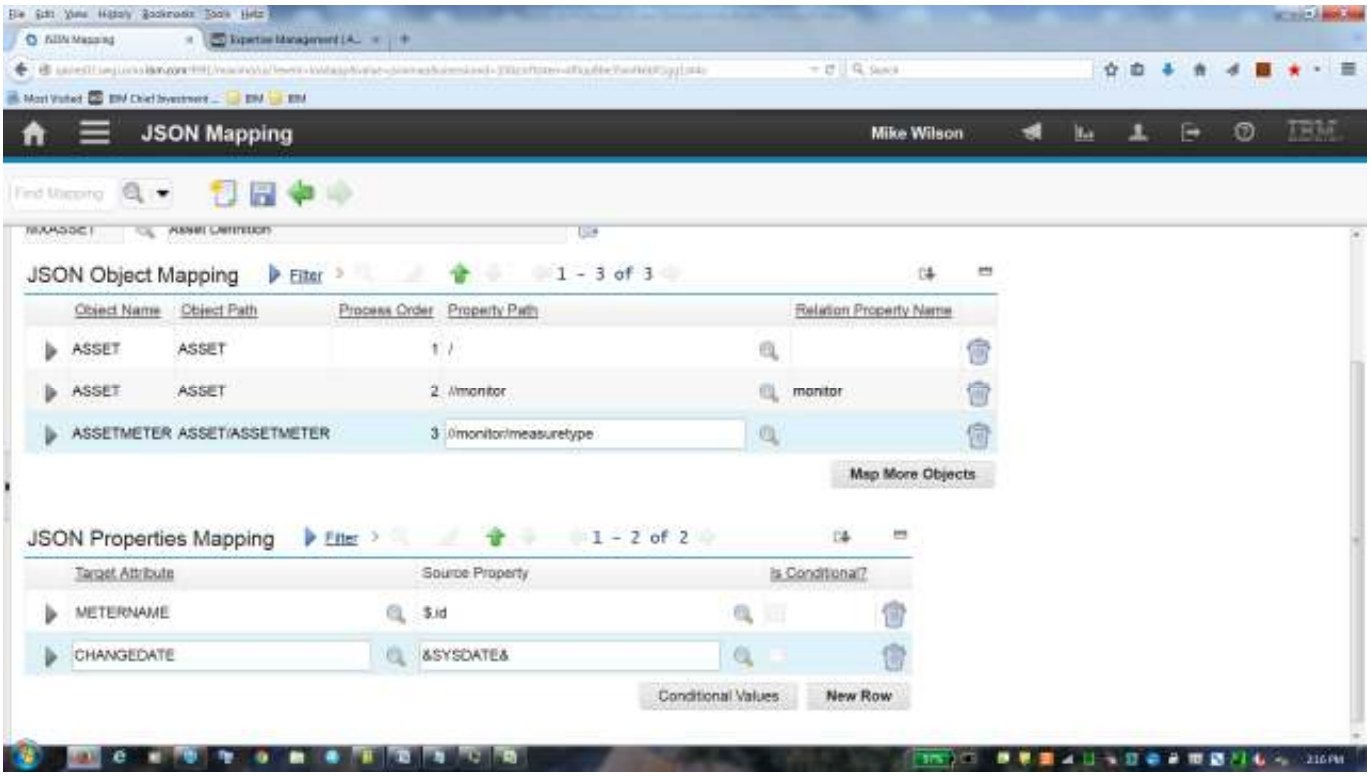
ASSETMETER to measuretype{}

You cannot skip the mapping of a layer, such as monitor, if you plan to map data from its child object . Since measuretype is an array, it is expected that it would map to a child object of ASSET, such as ASSETMETER, since ASSETMETER can hold multiple meters for an asset. Mapping monitor to ASSET makes sense since it is a single object, not an array. Keep in mind there is no requirement to map any fields from monitor to ASSET. Monitor is required in order to map data from measuretype.

In the JSON Mapping application the mapping would look like this:



Above is the mapping of the objects and the properties mapping for the ASSET and Root level of the JSON data. There are no mappings for properties of monitor{}. Below are the mappings for the measuretype:


JSON Mapping
Mike Wilson
JSON Object Mapping
Filter
1 - 3 of 3
Object Name
Object Path
Process Order
Property Path
Relation Property Name
ASSET
ASSET
1
/
ASSET
ASSET
2
/monitor
monitor
ASSETMETER
ASSET/ASSETMETER
3
/monitor/measuretype
Map More Objects
JSON Properties Mapping
Filter
1 - 2 of 2
Target Attribute
Source Property
Is Conditional?
METERNAME
$.id
CHANGEDATE
&SYSDATE&
Conditional Values
New Row

You could also map two different fields from one JSON Object to the same field in one Maximo object (two mapping rows) in order to create multiple occurrences of that Maximo object.

As seen in the screen shot above, there are some 'reserved' words that can be used to map a Maximo object field with:

&SYSDATE& = System Date/Time

&SITEID& = Default Siteid of the user of the transaction

&USERNAME& = User Name associated to the transaction

For an inbound transaction, the XML Action attribute can be assigned to each object in the field &ACTION&. This field is not selectable and must be keyed into the Target Attribute value. The value for the Action can be manually entered in the Source Property field and must be a valid Object structure action such as "AddChange", "Add" etc.

The above mapping screen shots show mapping that is for inbound processing where JSON data is being converted to Maximo data. For the case of outbound data using a Publish Channel, conceptually, the mapping is the same except in reverse. The source fields would be the Maximo object and the target fields would be the JSON data. As with inbound, if you are mapping to a child object in the JSON data, you must include its parent object in the mapping even if none of the fields in that object are being mapped.

Mapping Conditions

When you map a property you can apply a condition to conditionally map the value. In the application focus on the row to apply a condition and hit the Conditional Values button. Hit new row and use javascript syntax to enter a condition. The lookup for the Condition field allows you to select fields in the JSON data (when doing inbound mapping). Example:

The Object attribute DESCRIPTION is mapped to $.deviceInfo.description from the JSON data. A condition is added as follows:

Condition	**Value**
$.typeId=="PUMP"	$.deviceInfo.deviceClass

What this will do is if the typeID from the JSON data is equal to PUMP, the value mapped to the DESCRIPTION will be the deviceinfo.deviceClass, rather than the deviceinfo.description. The value can also be a hardcoded string such as "XYZ".

The select-able fields available for the Condition and Value are the JSON Data fields for an inbound mapping and Maximo Object fields for an outbound mapping.

You can use any valid javascript notation in your condition and you can have multiple conditions for the mapping of one field, which allows you to set the value to '1' when condition field is 'A'; set the value to '2' when condition field is 'B' and so on.

In the Properties Mapping section of the application when the 'Is Conditional' flag is check, that means 1 or more conditions have been defined for this property mapping.

TEST Action

The JSON Mapping application has a Test Action that allows an implementer to provide JSON data (for inbound processing) and the test utility will display the result Maximo XML based on the configured mapping. For outbound the implementer provides the sample Maximo XML and the test utility will display the JSON data produced using the mapping.