**Introduction**

The source-record-manager(SRM) converts MARC records to Inventory instances while handling incoming data. The process of converting MARC record into Instance object is usually called **MARC-to-Instance mapping**.

Conversion logic is defined by mapping rules and these rules are described in JSON. Mapping rule basically has functions for data normalization(trim leading whitespaces, remove slashes, remove ending punctuation) and the target Instance field, where to put result of mapping.

### What is mapping rule

Basically, rule is a simple key-value JSON element. The key serves a MARC record's field(tag). The value itself is a rule.

Rule:

{

 "001":[

 {

 "target":"hrid",

 "description":"The human readable ID"

 }

 ]

}

This rule belongs to the "001" field and handles all the "001" fields from incoming record. It takes value from "001" field and puts it into Instance "hrid" field. Such rules are usual for **Control field**.

#### **Normalization functions**

In most cases the record value needs to be normalized before getting into Instance field, because of record data is often raw and mixed . For this purpose we have to use such structure:

MARC Record: "001": "393/89/3"

Rule:

 "001":[

 {

 "target":"hrid",

 "description":"The human readable ID",

 "rules":[

 {

 "conditions":[

 {

 "type":"remove\_substring",

 "parameter":{

 "substring":"/"

 }

 }

 ]

 }

 ]

 }

 ]

remove\_substring is normalization function, that removes given substring from field's value. The function just doing a job and returns string that gets into Instance "hrid" field. An outcome Instance looks like this in Json:

Instance:

{

 "hrid": "393893"

}

[Here](https://github.com/folio-org/mod-source-record-manager/blob/master/mod-source-record-manager-server/src/main/java/org/folio/services/mappers/processor/functions/NormalizationFunction.java) all the formatting functions defined. Most useful are: trim, capitalize, remove\_ending\_punc.

In most cases there are sub-fields present in field, that is important for mapping. Example for "250" field with a, b, 6 sub-fields comes below:

MARC Record: "250":{"ind1":"", "ind2":"", "subfields":[ { "a":" fifth ed." }, { "b":"Editor in chief Lord Mackay of Clashfern. " } , {"6":"880-02"}]}

Rule:

 "250":[

 {

 "target":"edition",

 "description":"Edition",

 "subfield":["a", "b"],

 "rules":[

 {

 "conditions":[

 {

 "type":"capitalize, trim"

 }

 ]

 }

 ]

 }

 ]

This rule takes only "a" and "b" sub-fields and calls normalization functions for each sub-field. The result is concatenated in one string and written to the Instance "edition" field. An outcome Instance looks like this in Json:

Instance:

{

 "edition": "Fifth ed. Editor in chief Lord Mackay of Clashfern."

}

#### **Mapping for complex fields**

What if the target Instance field is not simple String, but List of complex objects with several fields in ? This happends usually if record field is a **Data field**. We can write rule to map record as below:

MARC Record: "264":{"subfields":[{"a":"Chicago, Illinois :"}, {"b":"The HistoryMakers,"}, {"c":"[2016]"}], "ind1":" ", "ind2":"1"}

Rule:

"264": [

 {

 "target": "publication.place",

 "description": "Place of publication",

 "subfield": ["a"],

 "rules": []

 },

 {

 "target": "publication.publisher",

 "description": "Publisher of publication",

 "subfield": ["b"],

 "rules": []

 },

 {

 "target": "publication.dateOfPublication",

 "description": "Date of publication",

 "subfield": ["c"],

 "rules": []

 }

]

An outcome Instance looks like this in Json:

Instance:

{

 "publication":[

 {

 "place":"Chicago, Illinois :",

 "publisher":"The HistoryMakers,",

 "dateOfPublication":"[2016]"

 }

 ]

}

If there are repeated "264" fields in a single record, then Instance gets several elements in the "publication" field. To skip mapping for repeated fields and take only first occurrence We can use ignoreSubsequentFields flag:

MARC Record:

"336":{"subfields":[{"a":"text"}, {"b":"txt"}, {"2":"rdacontent"}], "ind1":" ", "ind2":" "}, ...

"336":{"subfields":[{"a":"performed music"}, {"b":"prm"}, {"2":"rdacontent"}], "ind1":" ", "ind2":" "}

Rule:

"336": [

 {

 "target": "instanceTypeId",

 "description": "Instance type ID",

 "ignoreSubsequentFields": true,

 "subfield": ["b"],

 "rules": []

 }

 ]

An outcome Instance looks like this in Json:

Instance:

{

 "instanceTypeId": "txt"

}

#### **Multiple objects from one field**

Usually, the [Rule Processor](https://github.com/folio-org/mod-source-record-manager/blob/master/mod-source-record-manager-server/src/main/java/org/folio/services/mappers/processor/Processor.java) creates only one instance of the 'target' field for each record field. What if We need to create several objects from single record field ?

##### **New object for group of sub-fields**

In example below we map several 'publication' elements from a single "264" record field. To do so, we have to wrap mapping structure into entity:

MARC Record:

 "264": {

 "subfields":[

 {"a":"Chicago, Illinois :"},

 {"b":"The HistoryMakers,"},

 {"c":"[2016]"},

 {"f":"Nashville, Tennessee"},

 {"g":"Revenant Records"},

 {"h":"[2015]"}

 ],

 "ind1":" ",

 "ind2":"1"

 }

Rule:

 "264": [

 {

 "entity": [

 {

 "target": "publication.place",

 "description": "Place of publication",

 "subfield": ["a"],

 "rules": []

 },

 {

 "target": "publication.publisher",

 "description": "Publisher of publication",

 "subfield": ["b"],

 "rules": []

 },

 {

 "target": "publication.dateOfPublication",

 "description": "Date of publication",

 "subfield": ["c"],

 "rules": []

 }

 ]

 },

 {

 "entity": [

 {

 "target": "publication.place",

 "description": "Place of publication",

 "subfield": ["f"],

 "rules": []

 },

 {

 "target": "publication.publisher",

 "description": "Publisher of publication",

 "subfield": ["g"],

 "rules": []

 },

 {

 "target": "publication.dateOfPublication",

 "description": "Date of publication",

 "subfield": ["h"],

 "rules": []

 }

 ]

 }

 ]

An outcome Instance looks like this in Json:

Instance:

{

 "publication":[

 {

 "place":"Chicago, Illinois :",

 "publisher":"The HistoryMakers,",

 "dateOfPublication":"[2016]"

 },

 {

 "place":"Nashville, Tennessee :",

 "publisher":"Revenant Records,",

 "dateOfPublication":"[2015]"

 }

 ]

}

##### New object per repeated sub-fields

If there are several repeated sub-fields in one single record, then entity will concatenate them. To create a new object per each sub-field even if they are repeated, we can use entityPerRepeatedSubfield flag:

MARC Record:

 "264": {

 "subfields":[

 {"a":"Chicago, Illinois :"},

 {"a":"Nashville, Tennessee"},

 {"f": "Austin Texas"}

 ],

 "ind1":" ",

 "ind2":"1"

 }

Rule:

 "264": [

 {

 "entityPerRepeatedSubfield": true,

 "entity": [

 {

 "target": "publication.place",

 "description": "Place of publication",

 "subfield": ["a", "f"],

 "rules": []

 },

 {

 "target": "publication.publisher",

 "description": "Publisher of publication",

 "subfield": ["a", "f"],

 "rules": [

 {

 "conditions": [],

 "value": "STUB publisher"

 }

 ]

 },

 {

 "target": "publication.dateOfPublication",

 "description": "Date of publication",

 "subfield": ["a", "f"],

 "rules": [

 {

 "conditions": [],

 "value": "STUB date"

 }

 ]

 }

 ]

 }

 ]

An outcome Instance looks like this in Json:

Instance:

{

 "publication":[

 {

 "place":"Chicago, Illinois :",

 "publisher":"STUB publisher,",

 "dateOfPublication":"STUB date"

 },

 {

 "place":"Nashville, Tennessee :",

 "publisher":"STUB publisher",

 "dateOfPublication":"STUB date"

 },

 {

 "place":"Austin Texas",

 "publisher":"STUB publisher",

 "dateOfPublication":"STUB date"

 }

 ]

}

#### **Required sub-fields**

Sometimes there is a need to map target field depending on existence of some sub-field. We use requiredSubfield to define sub-field required to map target field:

MARC Record:

{

 "020":{

 "subfields":[

 {

 "z":"9780190494889"

 },

 {

 "q":"hardcover ;"

 },

 {

 "c":"alkaline paper"

 }

 ],

 "ind1":" ",

 "ind2":" "

 }

}

Rule:

"020": [

 {

 "entity": [

 {

 "target": "identifiers.value",

 "description": "Invalid ISBN",

 "subfield": ["z","q","c"],

 "requiredSubfield": ["z"],

 "rules": []

 }

 ]

 }

 ]

"z" sub-field is required for mapping "identifiers.value".

* If no "z" in record sub-fields, then "identifiers.value" remains empty(null).
* If "z" exists among record sub-fields, then "identifiers.value" gets filled by all the ["z","q","c"].

### REST API

When the source-record-manager starts up, it performs initialization for default mapping rules for given tenant. There are 3 REST methods to work with rules.

| **Method** | **URL** | **Content type** | **Description** |
| --- | --- | --- | --- |
| **GET** | /mapping-rules |  | Get rules for given tenant |
| **PUT** | /mapping-rules | application/json | Update rules for given tenant |
| **PUT** | /mapping-rules/restore | application/json | Restore rules to default |

To get rules you can send this request using GET method

curl -w '\n' -X GET \

-H "Content-type: application/json" \

-H "x-okapi-tenant: {tenant}" \

-H "x-okapi-token: {token}" \

https://folio-snapshot-load-okapi.aws.indexdata.com/mapping-rules