INPUT MAPPER

File Pre - Processor

File Pre-Processor screen is used to process the data before mapping it to CAN field. This is helpful when some data needs to be excluded from data load or some input data value needs to be modified before mapping it to CAN field.

This code will implement IPreprocessor interface that provides record object as parameter. Record object is a key value pair of header name (In case there is no header name, its convention starts with 0 as first column,1 as second column and so on) and header value.

File Post-processor

Splitting:

Direct Mapping

It is defined as converting single record into multiple records.

Mapped attribute header requires list of columns to be split. The splitting takes place in such a way that number of newly added records is equal to size of mapped attribute. Two columns are newly added in the record parser to split.

1. New attribute name - This field name is configured from UI. It stores the column headers that is added in mapped attribute.
2. Attribute value - This field contains the value corresponding to column that is stored in new attribute value.

These newly added columns are passed to parser where they can be mapped to CAN field.

Custom Mapping

The custom mapping can be used if single column cannot be used as split logic. Criteria that are more complex can be configured by writing code in IDE.

Grouping

Multiple records can be grouped into single record using this screen.

1. Attributes for group - The fields must be grouped overall and then used as grouped logic. The attributes to be grouped are configured from UI.

2. Attributes for sort - Sorting can be done on any required field by configuring it in UI.

3.Table consisting of attribute and group logic - The name of column must be considered as single and corresponding logic of group can be specified using either dropdown or complex code.

Record Parser

Parser code area accepts only the return type of field type that is being returned.

Example: If cause is configured, then it should be done as below:

```
Cause cause =new Cause();
cause.setName(“corresponding name”);
return cause;
```

In such cases, name is mandatory and it should be set to avoid rejection of the record. The return type of the field can be verified from the getRow method definition. In addition, IDE restricts returning of values other than the return type.

File Type:

- ***JSON Parser***

  JSON file to be parsed is configured by selecting file type as JSON from UI. JSON file parsing involves configuration of fields as shown below.

JSON key can be used as Mapping Name of parser mapped field area and same to be used in IDE.

If inner JSON is present, it needs to specified using dot(.)

```
{
"fault" : {
        "equipmentComponent" : "EQUIPMENT",
        "cause" : "SOME_CAUSE",
        "date" : 1.660542668E12
    },
}
```

fault.equipmentComponent denotes equimentComponent which is inside inner JSON fault.

Complex inner JSONs should be handled in same way using dots. For each inner JSON followed by key at end.

JSON array

```
{
"items" : [
        "BS8200",
        "B8200",
        "BC8910"
    ]
}
```

To specify field as BS8200 we should denote it as items[0]. It is same denoting array index starting from (0...n).

- ***XML Parser***

  XML can be selected in file type and 3 fields are:

  1. Start tag - Tag to be used as start.
  2. XML tag filter - Tag to be filtered from parsing.
  3. XML attribute - If this switch is ON, then the XML attribute is enabled and can be used during parsing.

  Consider the below XML:

```
<root>
<Customer CustomerID="GREAL">
<CompanyName>Great Lakes Food Market</CompanyName>
<ContactName>Howard Snyder</ContactName>
<ContactTitle>Marketing Manager</ContactTitle>
<Phone>(503) 555-7555</Phone>
<FullAddress>
<Address>2732 Baker Blvd.</Address>
<City>Eugene</City>
<Region attribute=”a”>OR</Region>
<PostalCode>97403</PostalCode>
```

```
<Country>USA</Country>
</FullAddress>
</Customer>
</root>
```

Here, <root> can be filtered by specifying it in XML tag filter.

<row> is used as start tag. Specifying correct start tag is important as it is entry point of parsing.

Every text element or sub element inside main element is denoted using [index] index that starts with 0 for single element.

When CompanyName, the text element inside main element needs to be denoted, it is done as CompanyName[0]. If multiple Company Name is present inside same main element, it is denoted as CompanyName[0], CompanyName[1]......

If attribute **attribute** of needs **Region** tag must be retrieved, it is done using dot(.) such as **Region[0].Attribute**. Attributes can be retrieved only if XML attribute is enabled.

Record Post – Processor

Post-processor is used to modify or discard the data after parsing and just before loading of data.

Post-processor screen UI and functionality is almost similar to Pre-processor screen.

However, code snippet written here will implement IPostprocessor interface that provides a map of troubleTicket object as parameter.