Validate Record in SAP NetWeaver MDM7.1

Applies to:

The beginners, developer developing java/j2ee applications to consume the features available in SAP NetWeaver Master Data Management 7.1 For more information, visit the Master Data Management homepage.

Summary

This document explains how to validate the record stored / to be stored in a repository in MDM.Here we will discuss about usage of MDM JAVA API's for validating a MDM Record against different field type validation including Tuples which is a new data structure introduced in 7.1.

Author: Vibha Sharma

Company: Infosys Technologies Limited

Created on: 31 August 2009

Author Bio

Vibha Sharma associated with Infosys Technologies for over 2 years has extensively worked on WebDynpro Java and worked on MDM.

Table of Contents

Need to Validate Data in MDM

SAP MDM is a component of SAP's Netweaver product group and is used as a platform to consolidate, cleanse and synchronize master data within a heterogeneous application landscape. Key capabilities of MDM include:

- Master Data Consolidation.
- Master Data Harmonization.
- Central Master Data Management and many more.

Whether we are consolidating or harmonizing or maintaining the data the key point is data should be cleaned.

Cleaning involves

- Assignment.
- Validation.
- Deduplication check.

We can interact with MDM either via JAVA / ABAP API's or via XI. Benefit of MDM validation is we need to define it once and it can be used by any of the interaction from there only. The key difference is when we are accessing via API's we can validate the data either before or after storing to MDM but when we are importing the data then we can run validation on imported records once they are present in MDM.

In present document we will be discussing about only one cleansing activity i.e. validation that too via MDM JAVA API's for different field types including Tuples which is a new data structure introduced in 7.1.

This document will concentrate in the details of validating a record having Text fields, Lookup fields and Tuple fields.

1: Validating a record before actually creating it.

2: Validating a record which is already present in repository.

Assumptions

The user has already created the connection with MDM server.

Important Note

While connecting to MDM server the important thing to keep in mind is that version of server and API should be compatible. Since the MDM Java API communicates directly on top of TCPI/IP, these two components must be compatible. You can ensure it by using the same build version of both the MDM Server and the API.

Repository Structure

For simplicity consider the following repository structure.

Customer: Main Table.

Account Group: Look up field in main table.

City: Text field in main table.

Contact Information: Tuple field in main table.

Contact_Name_Check - Name of validation specified in repository for Contact Information tuple.

City_Check - Name of validation specified in repository for City.

Account_Group_Check- Name of validation specified in repository for Account Group field.

Table mentioned below contains the field codes corresponding to the fields mentioned above.

Structure of Customers Table (Main Table)

Field Name	Field Code	Field Type
Account Group	Account_Group	Look up
City	City	Text
Contact Information	Contact_Information	Tuple

Table 1.1

1: Validating a record before actually creating it:

Validating a record before creating it implies that at the time of validation, the record does not exist in the repository. First question coming in mind is if the record is not there how we can validate it? Answer is if we want to validate any field against validation defined in MDM then for that we need to create an empty record.

The **RecordFactory** is a factory class responsible for creation of empty record instances on client side. These instances can be used for creation of new records in repository.

/**creating empty record using Record Factory*/

Record emptyRecord = RecordFactory.createEmptyRecord(tableId);

/**tableId- TableId of Customers table fetched using repository schema*/

We set the MDM Value of the field to the record which differs from field type to field type. So we can say that setting the field value to the record created above depends upon the field type.

setFieldValue (FieldId fieldId, MdmValue value);

This method takes FieldId as one of its parameter which can be found using the getField() of the schema. getField() takes table name and field code as input. In our case table name is Customers and the field codes are the codes mentioned in table1 column number 2.

Every thing appears to be straight forward till here, twist occurs when it comes to find the MDM values of different field types.

- **For look up fields is**

```
            value = new LookupValue(new RecordId(AccountGrRecId));
            /**Account Group is a look up field and accountGrRecId is the record Id
of the account group that
            needs to be set in empty record*/
```

- **For Text fields**

```
            value =  new StringValue(City);
            /**City is a text field and City is the value to be set in City field in
empty record*/
```

- **For MultiValued Tuples**

```
            MultiTupleValue multiTupleVal = new MultiTupleValue();
            /**Basic steps for Creating MultiTuple Value are first Creat a Tuple
value then set field value and
            Then finally add that Tuple Value to Multi TupleValue. (These steps are
not in scope of this
            document */
```

For validating a record against the validation defined in MDM first thing is to retrieve the validation. We can retrieve all validations defined in MDM via command **RetrieveValidationsCommand**. If the user is searching for some specific validation then user has to iterate the list fetched using this command, make a comparison and find out the validation that is required.

Validations for each type of fields are explained below.

1.1 Look up fields and Text fields

```
/**Creating Object of RetrieveValidationsCommand**/
RetrieveValidationsCommand    retrieveValidationCommand  = new
RetrieveValidationsCommand(simpleConnection);
This command needs the following parameters for its execution:
Session and TableId .

retrieveValidationCommand.setSession(userSession);
retrieveValidationCommand.setTableId(tableId);
 try {
/**executing retrieveValidationCommand*/
retrieveValidationCommand.execute();
  } catch (CommandException e) {
 e.printStackTrace();
  }

/**simpleConnection – is ConnectionPool Object retrieved after getting connected.*/
/** userSession – is the session object*/
/** tableId – Table Id of Customers table fetched using repository schema.*/
```

Note: If there is an error in executing the command it throws CommandException.

From RetrieveValidationsCommands object we can get the **ValidationPropertiesResult** which in turn gives the **ValidationProperties** from which we can find the Id of the validation that we want to execute.

```
ValidationPropertiesResult    validationPropertiesResult =
retrieveValidationCommand.getValidationPropertiesResult();

/**Getting the Validation*/

ValidationProperties validationProperties[] =validationPropertiesResult.getValidations();

ValidationId validationIds[] =new ValidationId[1];

 /**Looping through the validation properties comparing the name and finding the validation we are looking for. It can be single validation  or multiple validation Here we are looking for City and Account Group field validation*/

 for (int i = 0; i < validationProperties.length; i++) {

if(validationProperties[i].getName().toString().trim().equalsIgnoreCase(City_Check)){

validationIds[0] = validationProperties[i].getId();

 }

 if(validationProperties[i].getName().toString().trim().equalsIgnoreCase(Account_Group_Check)){

                validationIds[0] = validationProperties[i].getId();

}

 }
```

As the record we created does not exist in table yet so for validating it we use command **ValidateNewRecordValuesCommand**.This is a command to validate record values against the specified validation rules. The record values can be for an existing record or for a completely new record. The results of the validations execution specify which validation rule failed.

The results also specify which record failed. Declaring **MdmValidationResult's** object which will hold the result of the execution of validation.

```
MdmValidationResult mdmValidationResult = null;

/**Creating Object of ValidateNewRecordValuesCommand*/

ValidateNewRecordValuesCommand  validateCommand = new
ValidateNewRecordValuesCommand(simpleConnection);

/**Setting the parameters needed to execute ValidateNewRecordValuesCommand*/

validateCommand.setSession(userSession);

validateCommand.setTableId(tableId);

/**set the empty record created above which is mandatory parameter for execution of this command */

validateCommand.setRecord(emptyRecord);

/**set the validation Id of the validation that you want to get executed over this record*/

validateCommand.setValidationIds(validationIds);
```

After executing the ValidateNewRecordValuesCommand its object gives the validation results.

```
 try {

/**Executing ValidateNewRecordValuesCommand*/

validateCommand.execute();

/**Fetching the ValidationResult */
```

ValidationResult validationResult = validateCommand.getValidationResult();

/** * RecordId for a new record (empty record) will be null. Creating a recordId of -1* */

RecordId recordId = new RecordId(-1);

validationProperties = validationResult.getFailedValidations(recordId);

from validationProperties we can find the error messages defined in MDM to be displayed if the validation fails.

1.2 Tuple fields:

Validation on tuples can be defined at two levels.

(a) Table level: A validation is said to be on table level when it do not deals with the data inside the record of a tuple. It can be elaborated as if we define validation over tuple field in Customers table then it is a Table level validation.

(b) Tuple level: If a validation is defined over a tuple irrespective of table then it is tuple level validation.

How do we get to know whether the validation is Table level or Tuple level?

If the Tuple field validation is present under the main table validation then it is Table level.

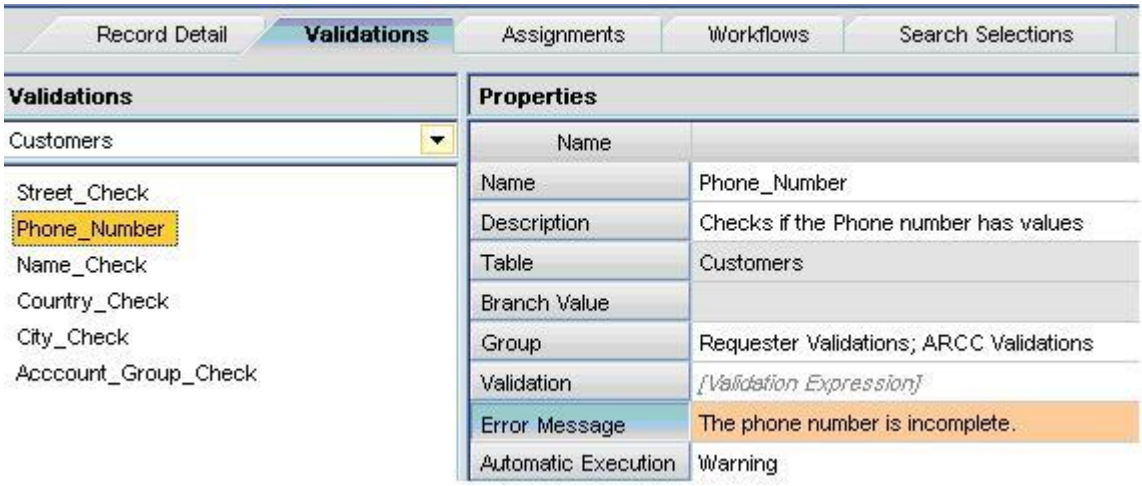

Figure 1: Showing Table Level Validation

Here Phone_Number is a tuple field in Customers table and we are specifying validation on that field under main table hence it is Table level.

If the validation is present under Tuple not under main table then it is a Tuple level validation. As shown in figure 2.

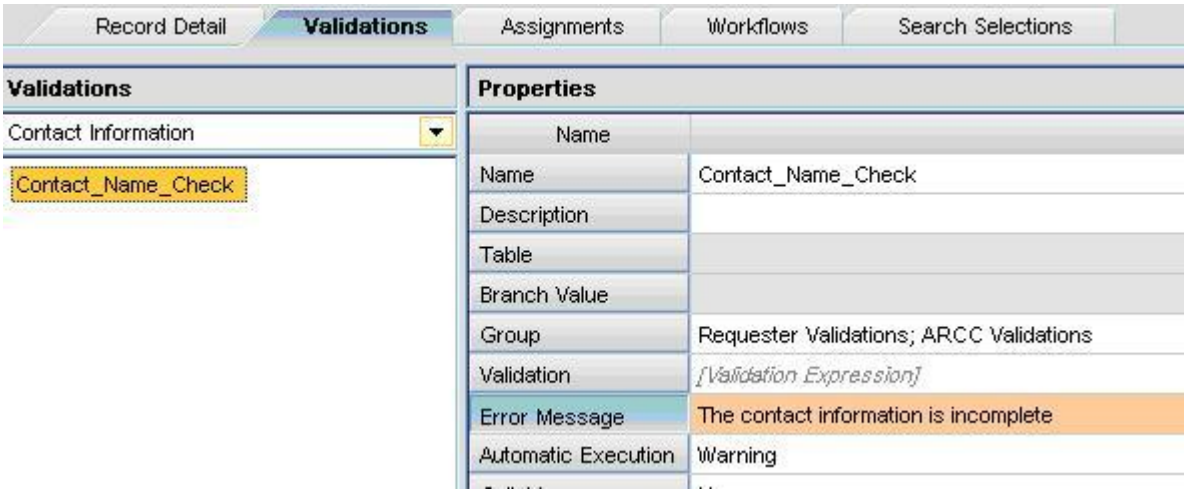

Figure 2: Showing Tuple Level Validation

Here Contact Information is a tuple field in Customers table and we are specifying validation on that field separately inside the tuple hence it is Tuple level validation.

Now a big question comes in our mind is then how to use these validations via API.

The answer is simple in case of table level validation the procedure for validating the data remains the same but if it is a tuple level validation then in order to retrieve the validations we need to pass the TupleDefinitionId while setting the parameters to RetrieveValidationsCommand. TupleDefinitionId is an identifier for a tuple definition. Once we set the TupleDefinitionId the list of validations retrieved will be for this tuple definition. The following code can be used for that.

```
/**First fetching the TupleDefinitionProperties and from TupleDefinitionProperties getting Tuple DefinitionId*/

TupleDefinitionProperties tupleDePro = schema.getTupleDefinition("ZContact_Information");

TupleDefinitionId tupleDefId = tupleDePro.getId();

/**Creating Object of RetrieveValidationsCommand**/

RetrieveValidationsCommand retrieveValidationCommand =new RetrieveValidationsCommand(simpleConnection);

This command needs the following parameters for its execution:Session and TableId

retrieveValidationCommand.setSession(userSession);retrieveValidationCommand.setTableId(tableId);

retrieveValidationCommand.setTupleDefinitionId(tupleDefId);

try {

/**executing retrieveValidationCommand*/

retrieveValidationCommand.execute();

 } catch (CommandException e) {

e.printStackTrace();

 }
```

Rest of the steps remains the same.

In present scenario in case of Tuples the name of the validation is Contact_Name_Check so while looping through validation properties compare with Contact_Name_Check

```
 for (int i = 0; i < validationProperties.length; i++)
```

```
{
if(validationProperties[i].getName().toString().trim().equalsIgnoreCase(Contact_Name_Check))
{ validationIds[0] = validationProperties[i].getId();
 }
}
```

2: Validating existing record:

For validating existing record we have two options either to use

ValidateNewRecordValuesCommand or

ValidateRecordsCommand

In case of ValidationNewRecordValuesCommand we need to set the already existing record where as in case of ValidateRecordsCommand we need to set the recordIds of already existing records that needs to be validated using setRecordIds method. Major difference is that with with ValidateNewRecordValuesCommand we ca**n validate only single record at a time where as with ValidateRecordsCommands we can validate more than one record at a time.** Other basic steps are similar to that for validating a record before creating it.

Related Content

http://help.sap.com/javadocs/MDM71/current/API/index.html

For more information, visit the Master Data Management homepage.

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.