

How to Build Your Own UI for MDM via Java API - Limit Search Criteria

Applicable Releases:

SAP NetWeaver Master Data Management 7.1 and below

Topic Area:

Information Management

Capability:

Master Data Management

Version 1.0

June 2009

© Copyright 2009 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, Informix, i5/OS, POWER, POWER5, OpenPower and PowerPC are trademarks or registered trademarks of IBM Corporation.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice.

These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

SAP NetWeaver "How-to" Guides are intended to simplify the product implementation. While specific product features and procedures typically are explained in a practical business context, it is not implied that those features and procedures are the only approach in solving a specific business problem using SAP NetWeaver. Should you wish to receive additional information, clarification or support, please refer to SAP Consulting.

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.

Disclaimer

Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressly prohibited, as is any decompilation of these components.

Any Java™ Source Code delivered with this product is only to be used by SAP's Support Services and may not be modified or altered in any way.

Document History

| Document Version | Description |
|-------------------------|--------------------------------------|
| 1.00 | First official release of this guide |

Typographic Conventions

| Type Style | Description |
|-----------------------------|--|
| <i>Example Text</i> | Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation |
| Example text | Emphasized words or phrases in body text, graphic titles, and table titles |
| Example text | File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools. |
| Example text | User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation. |
| <Example text> | Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system. |
| EXAMPLE TEXT | Keys on the keyboard, for example, F2 or ENTER. |

Icons

| Icon | Description |
|---|-----------------------|
|  | Caution |
|  | Note or Important |
|  | Example |
|  | Recommendation or Tip |

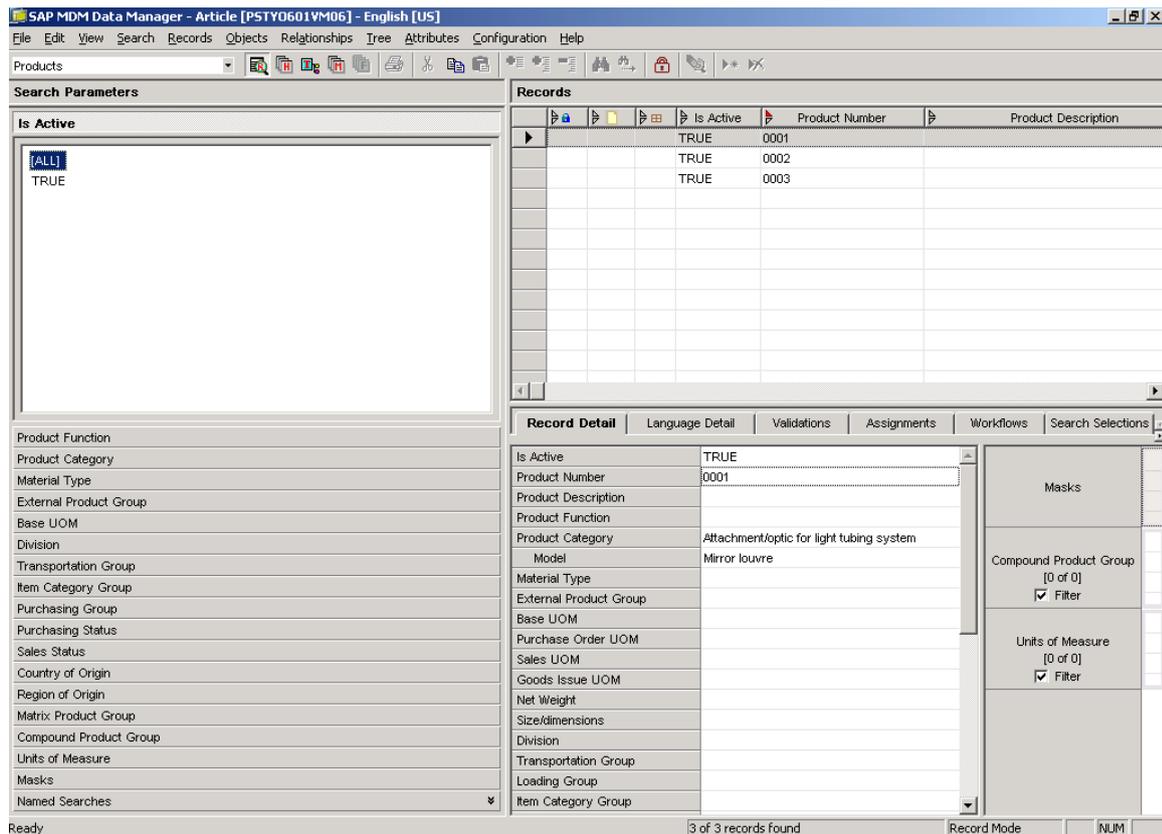
Table of Contents

- 1. **What are we going to achieve? 1**
- 2. **How to do it with Java API?..... 4**
 - 2.1 Preparation 4
 - 2.2 Command for lookup Pick List..... 5
 - 2.3 Command for Attribute Pick List..... 9
 - 2.4 Command for Attribute Value Pick List..... 10
 - 2.5 Retrieve Attribute Values 11
 - 2.5.1 Before 5.5 SP06 Patch 4 and 7.1 SP03 11
 - 2.5.2 After 5.5 SP06 Patch 4 and 7.1 SP03 13
 - 2.6 How to narrow down the search more by using Pick List? 15
 - 2.6.1 Before 5.5 SP06 Patch 4 and 7.1 SP03 15
 - 2.6.2 After 5.5 SP06 Patch 4 and 7.1 SP03 16
- 3. **Appendix 17**

1. What are we going to achieve?

There is a powerful function in MDM which is limiting the search criteria based on the existing values.

For example we have 3 records in the repository whose values for “Is Active” field are all TRUE. In this case if you take a look at MDM Data Manager’s **Search Parameters** pane on the left side, only TRUE is shown in “Is Active”. Therefore, users will be able to click on only the existing values in the **Search Parameters** pane to narrow down the search result.



The screenshot shows the SAP MDM Data Manager interface for Article [PSTY0601VM06] - English [US]. The interface is divided into several panes:

- Search Parameters:** The "Is Active" field is selected, and the value "TRUE" is displayed in the list below it.
- Records:** A table showing three records with the following data:

| Is Active | Product Number | Product Description |
|-----------|----------------|---------------------|
| TRUE | 0001 | |
| TRUE | 0002 | |
| TRUE | 0003 | |
- Record Detail:** The "Is Active" field is set to TRUE, and the "Product Number" is 0001. Other fields like "Product Description" and "Product Function" are also visible.
- Filters:** On the right side, there are filters for "Compound Product Group" and "Units of Measure", both with a checked "Filter" option.

The status bar at the bottom indicates "3 of 3 records found" and "Record Mode".

This is possible not only on lookup field values, but also in categories and attribute values as well.

Here we have a taxonomy definition as below

Accessories for lighting (Parent Category)

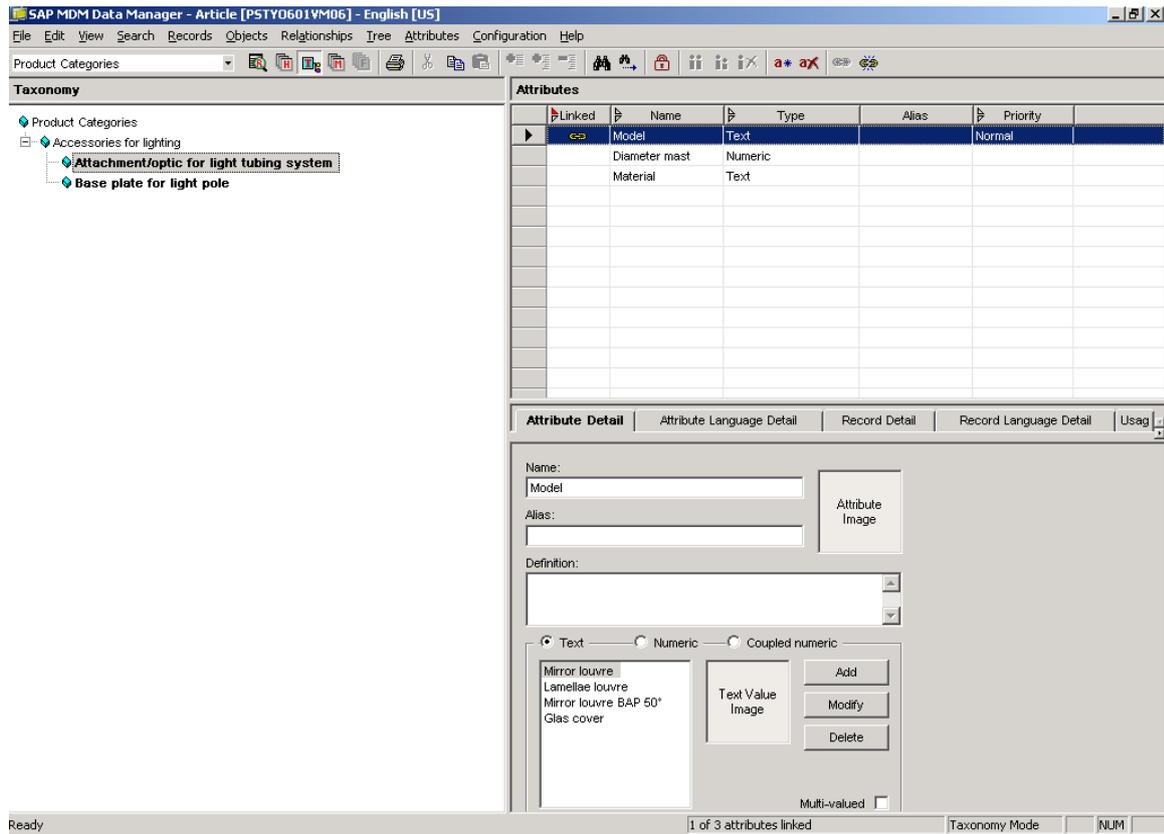
Attachment/optic for light tubing system (Child Category)

Model (Text Attribute)

Base plate for light pole (Child Category)

Diameter mast (Numeric Attribute)

Material (Text Attribute)



We have 3 records again which have the values below

| | |
|------|---|
| 0001 | Attachment/optic for light tubing system Model = Mirror Louvre |
| 0002 | Base plate for light pole Diameter mast = 12 mm Material = Aluminum |
| 0003 | Base plate for light pole Diameter mast = 12 mm Material = Steel |

As you can see again in "Product Category" in **Search Parameters** pane, Data Manager shows the category lookup values "Attachment/optic for light tubing system" and "Base plate for light pole", because category lookup value is not different than another lookup field value.

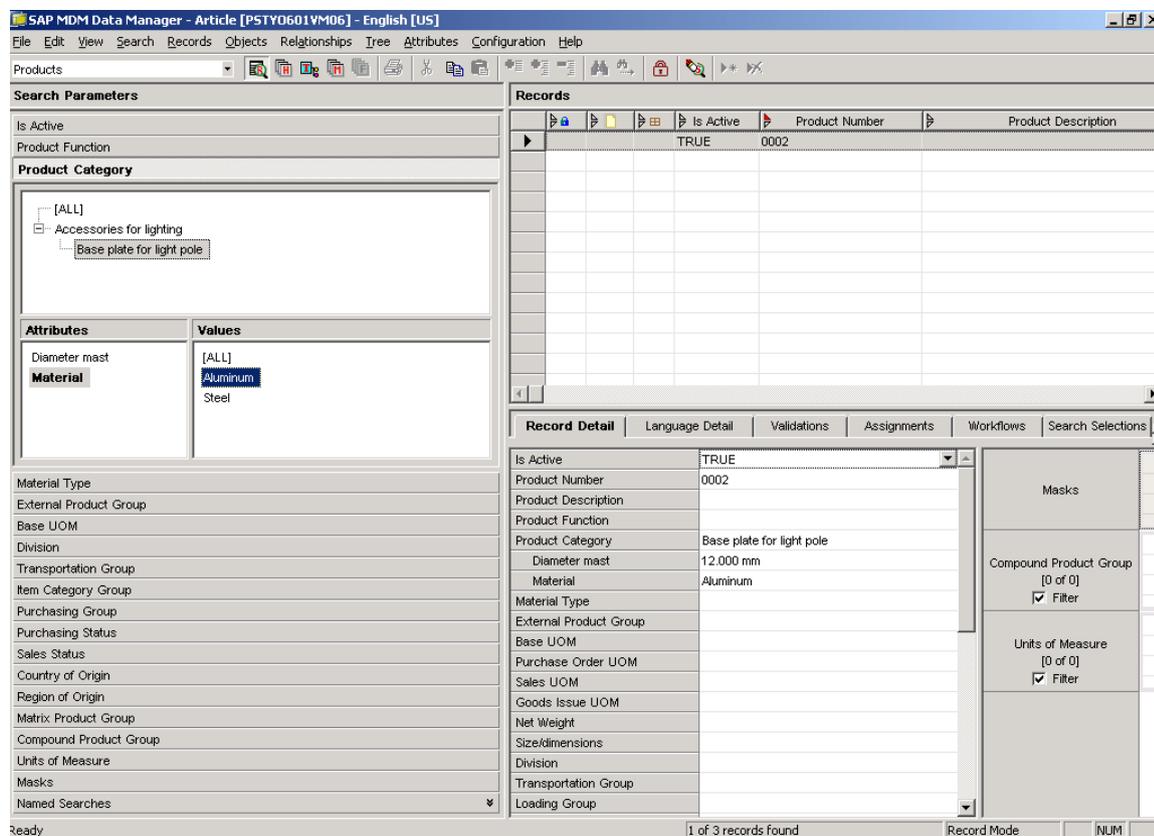
Below the category values, **Search Parameters** pane shows available attributes as well. Currently we have *Model*, *Material*, and *Diameter mast*. Please note that "available Attributes" doesn't mean all the definition for Attributes, but rather, it means that these Attributes are assigned to existing records.

The screenshot shows the SAP MDM Data Manager interface. The 'Search Parameters' section includes 'Product Category' with a tree view showing 'Accessories for lighting', 'Attachment/optic for light tubing system', and 'Base plate for light pole'. The 'Attributes' table shows 'Diameter mast' with value '[ALL]' and 'Material' with value '12.000 mm'. The 'Records' table displays three records with 'Is Active' set to TRUE and 'Product Number' values 0001, 0002, and 0003. The 'Record Detail' for Product Number 0001 shows 'Product Category' as 'Attachment/optic for light tubing system' and 'Model' as 'Mirror louvre'. The status bar at the bottom indicates '3 of 3 records found'.

The screenshot shows the SAP MDM Data Manager interface with the 'Base plate for light pole' category selected in the 'Product Category' tree. The 'Attributes' table shows 'Material' with value '12.000 mm'. The 'Records' table displays two records with 'Is Active' set to TRUE and 'Product Number' values 0002 and 0003. The 'Record Detail' for Product Number 0002 shows 'Product Category' as 'Base plate for light pole', 'Diameter mast' as '12.000 mm', and 'Material' as 'Aluminum'. The status bar at the bottom indicates '2 of 3 records found'.

When we click on any Attribute, e.g. *Material*, all assigned values for *Material*, e.g. *Aluminum* and *Steel*, are displayed in **Values**.

Afterward, if we would to click on any value, the search result will narrow down the search to get all the records which are assigned to this value. E.g. the screenshot below shows the records whose *Material* attribute value is *Aluminum*.



In some cases, you will need to realize it in other UI technologies, e.g. web applications. MDM is providing strong Java API capability for this. Let's take a look at the procedure step by step.

2. How to do it with Java API?

2.1 Preparation

First as usual, we will need a connection to connect to MDM Server. As this connection requires data access, we will use `UserSessionContext` here.

```
context =
    new UserSessionContext(server_name, repository_name,
        username_name);
```

Log on to MDM Server.

```
SessionManager.getInstance().createSession(  
    context, SessionTypes.USER_SESSION_TYPE, password);
```

In case you are building your UI on SAP Web Application System, you might need to leverage SAP's security infrastructure instead of the hard coded user and password above. Please refer to the document below on how to retrieve user and password via SAP Security API.

<http://help.sap.com/javadocs/NW04S/current/se/com/sap/security/api/umap/IUserMapping.html>

<http://help.sap.com/javadocs/NW04S/current/se/com/sap/security/api/umap/IUserMappingData.html>

We will use RepositorySchema as well, since it is taking good care of repository META data by using lazy load mechanism.

```
RepositorySchema schema =  
    MetadataManager.getInstance().getRepositorySchema(context);
```

2.2 Command for lookup Pick List

Here is the command which will retrieve the result set for the search, as well as return the pick list for available lookup values.

```
final RetrieveLimitedRecordsCompoundCommand command =  
    new RetrieveLimitedRecordsCompoundCommand(context);
```

We need to create the Result Definition to tell API what field values need to be retrieved together with the search result.

```
ResultDefinition rd =  
    new ResultDefinition(schema.getFieldId(MAIN_TABLE_CODE));  
rd.addSelectField(schema.getFieldId(MAIN_TABLE_CODE, FIELD_CODE));  
command.setResultDefinitions(new ResultDefinition[] { rd});
```

Next set the search criteria for the search.

```
Search search = ...;  
command.setSearch(search);
```

Performance issues will arise if we were to retrieve all records for search as many records would be hit. As such, we use the paging concept to retrieve records page by page.

```
command.setPageSize(page_size);  
command.setPageIndex(page_index);
```

Here we have to define what pick lists we need to retrieve from the command. E.g. we define 3 lookup fields.

```
command.setSearchDimensions(new SearchDimension[] {  
    new FieldSearchDimension(  
        schema.getFieldId(MAIN_TABLE_CODE,  
            PICKLIST1_FIELD_CODE)),  
    new FieldSearchDimension(  
        schema.getFieldId(MDM_PRODUCTS_TABLE_CODE,  
            PICKLIST1_FIELD_CODE)),  
    new FieldSearchDimension(  
        schema.getFieldId(MDM_PRODUCTS_TABLE_CODE,  
            PICKLIST1_FIELD_CODE))});
```

Now we can execute the command.

```
command.execute();
```

After the command was executed successfully, we can get the result from the result set.

```
final RecordResultSet recordResultSet =  
    command.getSearchTableResultSet();  
for (int i = 0; i < recordResultSet.getCount(); i++) {  
    final Record record = recordResultSet.getRecord(i);  
    print("got result: " + record.getDisplayValue());  
}
```

As mentioned previously, one command execution will retrieve only one page of results. We will need to identify the next page index and execute the command again to get different page.

Normally in web applications, we show the users how many pages existing. In order to calculate the number, we can divide the total result count by page size. We can get total result count by the method below.

```
final int count = command.getSearchTableMatchCount();
```

Now we will have to get the pick lists for limiting available values. Remember we defined 3 lookup fields before executing the command. We get an array here for different fields.

```
final PickList[] pickLists = command.getPickList();
```

We get the information from each PickList.

First is which field the particular pick list is about. `Picklist.getSearchDimension()` returns corresponding `SearchDimension` instance, from where we can get the field information.

`FieldSearchDimension.getFieldPath()` returns an array of field ids. As we are not talking about tuple, we will get only one `FieldId` in this array.

```
final FieldId[] fieldIds =
    ((FieldSearchDimension) pick.getSearchDimension()).
        getFieldPath();
```

Next we get the items for the pick list.

```
final PickListItem[] items = pick.getPickListItems();
```

There are only 2 types of pick list supported in `RetrieveLimitedRecordsCompoundCommand`. `RecordPickListItem` and `BooleanPickListItem`. `RecordPickListItem` is taking care of all the lookup types including taxonomy lookup, and `BooleanPickListItem` is taking care of all the boolean type fields. We can differentiate the types by the code below.

```
switch (item.getType()) {
    case PickListItem.RECORD_PICK_LIST_ITEM:
        RecordPickListItem rItem =
            (RecordPickListItem) item;
        ...
        break;
    case PickListItem.BOOLEAN_PICK_LIST_ITEM:
        BooleanPickListItem bItem =
            (BooleanPickListItem) item;
```

```
        ...
        break;
    default:
        assert (false);
    }
}
```

In case the pick list item is lookup (referring to another record) type of pick list item, we can read the record like below.

```
RecordPickListItem item;
final Record record = item.getRecord();
if (record instanceof HierNode) {
    ...
} else {
    String value = record.getDisplayValue();
}
```

In case the lookup field is for taxonomy or hierarchy, we will get a tree type HierNode instance. Here is an example of how to show all the nodes in the entire tree.

```
private void printTree(HierNode node, String indent) {
    if (node == null) {
        print("category: " + indent + "NULL");
    }
    else {
        print("category: " + indent + node.getDisplayValue());
        final HierNode[] children = node.getChildren();
        if (children != null) {
            for (HierNode child : children) {
                printTree(child, indent + "\t");
            }
        }
    }
}
```

In case the pick list item is Boolean type of pick list item, we can read the record like below.

```
BooleanPickListItem item;  
Boolean value = item.getValue();
```

2.3 Command for Attribute Pick List

You must have noticed why only record and Boolean types are supported. Didn't we say we can support attribute or attribute values? The answer is you need another 2 commands to achieve that.

The first one is for available attributes.

```
final RetrieveLimitedAttributesCommand command =  
    new RetrieveLimitedAttributesCommand(context);
```

Same as RetrieveLimitedRecordsCompoundCommand we need to set the parameters for search.

```
command.setSearch(search);
```

The difference is we need to identify which taxonomy lookup field we are going to get. In case we have multiple taxonomy lookup fields to retrieve the pick list, we will have to call the command multiple times.

```
command.setTaxonomyFieldId(taxonomyLookupFieldId);
```

Now execute the command.

```
command.execute();
```

Then we can get the available assigned attributes according to the search criteria.

```
final AttributeProperties[] attrProps =  
    retrieveLimitedAttributesCommand.getAttributes();
```

And print the information accordingly.

```
print("found available attribute: " + attrProp.getName());
```

As we have different ratings for each attribute when attribute is numeric type, by the code below, we can show the rating information.

```
if (attrProp.getType() == AttributeProperties.NUMERIC_TYPE) {
    final NumericAttributeProperties nap =
        (NumericAttributeProperties) attrProp;
    if (nap.supportsRating(AttributeId.AVERAGE_RATING)) {
        print("supports rating: [average]");
    }
    if (nap.supportsRating(AttributeId.MAXIMUM_RATING)) {
        print ("supports rating: [maximum]");
    }
    if (nap.supportsRating(AttributeId.MINIMUM_RATING)) {
        print ("supports rating: [minimum]");
    }
    if (nap.supportsRating(AttributeId.NOMINAL_RATING)) {
        print ("supports rating: [nominal]");
    }
    if (nap.supportsRating(AttributeId.TYPICAL_RATING)) {
        print("supports rating: [typical]");
    }
}
```

2.4 Command for Attribute Value Pick List

Here is the command to get corresponding available attribute values for one attribute. As you have seen in the Data Manager, a good response time would not be achieved if we were to retrieve all the attributes. Therefore we just retrieve the attribute values for each attribute type via individual command. We will need to build a UI to call this command once user clicked the attribute name.

```
final RetrieveLimitedAttributeValuesCommand command =
    new RetrieveLimitedAttributeValuesCommand(context);
```

Again we set the search criteria for records.

```
command.setSearch(search);
```

We set taxonomy lookup field id.

```
command.setTaxonomyFieldId(taxonomyLookupFieldId);
```

As this command is retrieving available attribute values for only one attribute, we need to identify the attribute id. There are multiple ways to get the attribute id. As we have already got attribute property instance from `RetrieveLimitedAttributesCommand`, we will get id from the attribute property.

```
command.setAttributeId(attrProp.getId());
```

Now execute the command.

```
command.execute();
```

2.5 Retrieve Attribute Values

Now we have to retrieve the value from the command result.

2.5.1 Before 5.5 SP06 Patch 4 and 7.1 SP03

Before 5.5 SP06 Patch4 and 7.1 SP03, we write the code below to get the attribute values.

First we use the method below to retrieve the attribute values directly.

```
final MdmValue[] values = command.getAttributeValues();
```

In order to get the attribute values, we need to check what type of attribute it is.

```
switch (attrProp.getType()) {
case AttributeProperties.COUPLED_TYPE:
    CoupledAttributeProperties cap =
        (CoupledAttributeProperties) attrProp,
    CoupledMeasurementValue cmv =
        (CoupledMeasurementValue) value;
    ...
    break;
case AttributeProperties.NUMERIC_TYPE:
    NumericAttributeProperties nap =
        (NumericAttributeProperties) attrProp;
```

```
        MeasurementValue mv = (MeasurementValue) value;
        ...
        break;
case AttributeProperties.TEXT_TYPE:
    TextAttributeProperties tap =
        (TextAttributeProperties) attrProp;
    TextAttributeValue tav = (TextAttributeValue) value;
    ...
    break;
default:
    assert (false);
}
```

In case of numeric type of attribute, we retrieve the value as below.

```
NumericAttributeProperties attrProp;
MeasurementValue value;
final DimensionsManager dm =
    MetadataManager.getInstance().
        getRepositoryDimensions(context);
final UnitProperties up =
    dm.getUnit(attrProp.getMeasurement().getDimensionId(),
        value.getUnitId());
print("got numeric attribute value: " + value.getMagnitude() +
    " " + up.getName());
```

In case of coupled type of attribute, we use the code below.

```
CoupledAttributeProperties attrProp;
CoupledMeasurementValue value;
final MeasurementValue value1 = value.getPrimaryValue();
final MeasurementValue value2 = value.getSecondaryValue();
final UnitProperties up1 =
    dm.getUnit(attrProp.getMeasurement().getDimensionId(),
        value1.getUnitId());
final UnitProperties up2 =
```

```
dm.getUnit(attrProp.getCoupledMeasurement().
    getDimensionId(), value2.getUnitId());
print("got attribute value: " + value1.getMagnitude() + " " +
    up1.getName() + " : " + value2.getMagnitude() + " " +
    up2.getName());
```

In case of text type of attribute, we use the code below.

```
TextAttributeProperties attrProp;
TextAttributeValue value;
print("got text attribute value: " +
    attrProp.getTextAttributeValue(value.getId()).getName());
```

2.5.2 After 5.5 SP06 Patch 4 and 7.1 SP03

After 5.5 SP06 Patch4 and 7.1 SP03, new `AttributeValuePickListItem` interface is introduced, and therefore we can write the code as per below stated.

```
final PickList[] attributePickLists = command.getPickLists();
for (PickList attributePickList : attributePickLists) {
    final PickListItem[] attributePickListItems =
        attributePickList.getPickListItems();
    for (PickListItem attributePickListItem : attributePickListItems)
    {
        final AttributeValuePickListItem
        attributeValuePickListItem =
            (AttributeValuePickListItem) attributePickListItem;
        ...
    }
}
```

Here is the code to identify the item type.

```
switch (attributeValuePickListItem.getType()) {
case AttributeValuePickListItem.COUPLED_ATTRIBUTE_VALUE_TYPE:
    CoupledAttributeProperties cap = (CoupledAttributeProperties)
    attrProp;
```

```
CoupledMeasurementValuePickListItem citem =
    (CoupledMeasurementValuePickListItem)
    attributePickListItem;
...
break;
case AttributeValuePickListItem.NUMERIC_ATTRIBUTE_VALUE_TYPE:
    NumericAttributeProperties nap =
        (NumericAttributeProperties) attrProp;
    MeasurementValuePickListItem mitem =
        (MeasurementValuePickListItem) attributePickListItem;
    ...
    break;
case AttributeValuePickListItem.TEXT_ATTRIBUTE_VALUE_TYPE:
    TextAttributeProperties tap =
        (TextAttributeProperties) attrProp;
    TextAttributeValuePickListItem titem =
        (TextAttributeValuePickListItem) attributePickListItem;
    ...
    break;
default:
    // no other types existing
    assert (false);
    break;
}
```

Here is the code for getting the value wrapper classes for each item. The concrete code for value retrieval from wrapper classes is the same as before SP06 Patch4, and therefore it is not shown here.

In case of numeric type attributes,

```
MeasurementValuePickListItem item;
final MeasurementValue value = item.getValue();
```

In case of coupled type of attributes,

```
CoupledMeasurementValuePickListItem item;
final CoupledMeasurementValue value = item.getValue();
```

In case of text type of attributes,

```
TextAttributeValuePickListItem item;  
final TextAttributeValue value = item.getValue();
```

Now we got all the pick lists for lookup fields, taxonomy lookup fields, attributes, and attribute values.

2.6 How to narrow down the search more by using Pick List?

Like in Data Manager, we probably will need a UI to narrow down the search results by putting more search criteria when users clicked on one of the pick lists. We can use the code below to create new SearchParameter and then append it into Search instance which would help in narrowing down via attribute values.

2.6.1 Before 5.5 SP06 Patch 4 and 7.1 SP03

Before 5.5 SP06 Patch 4 or 7.1 SP03, you can create the search criteria from scratch as below.

In case of numeric attributes,

```
NumericAttributeProperties attrProp;  
MeasurementValue value;  
final PickListSearchConstraint pickListSearchConstraint = new  
PickListSearchConstraint(new MdmValue[] { value});  
SearchParameter par = new SearchParameter(  
    new AttributeSearchDimension(taxonomyLookupFieldId,  
        attrProp.getId()), pickListSearchConstraint);
```

In case of coupled attributes,

```
CoupledAttributeProperties attrProp;  
CoupledMeasurementValue value;  
final PickListSearchConstraint pickListSearchConstraint =  
    new PickListSearchConstraint(new MdmValue[] { value});  
SearchParameter par = new SearchParameter(  
    new AttributeSearchDimension(taxonomyLookupFieldId,  
        attrProp.getId()), pickListSearchConstraint);
```

In case of text attributes,

```
TextAttributeProperties attrProp;
TextAttributeValue value;
final PickListSearchConstraint pickListSearchConstraint =
    new PickListSearchConstraint(new MdmValue[] { value});
SearchParameter par = return new SearchParameter(
    new AttributeSearchDimension(taxonomyLookupFieldId,
        attrProp.getId()), pickListSearchConstraint);
```

2.6.2 After 5.5 SP06 Patch 4 and 7.1 SP03

As mentioned previously, from 5.5 SP06 Patch 4 or 7.1 SP03, `RetrieveLimitedAttributeValuesCommand` returns `PickList` instances.

If you have stored the `PickList` instances beforehand, you can reuse these instances when creating `SearchParameter` as below. This will make the code much simpler.

```
int selectedPickList = xxx;
PickList pickList = pickLists[selectedPickList];
int selectedPickListItem = yyy;
PickListItem pickListItem =
    pickList.getPickListItems()[selectedPickListItem];

SearchParameter par =
    new SearchParameter(pickList.getSearchDimension(),
        pickListItem.getSearchConstraint());
```

3. Appendix

Sample Code

- Sample code download
<https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/b01c248f-e13a-2c10-259c-a6b3bfbc350>

References

- MDM Java API Online Help:
http://help.sap.com/saphelp_nwmdm71/helpdata/en/46/7a11f8a13c0ad6e1000000a11466f/fra/meset.htm
- MDM Java API javadoc:
<http://help.sap.com/javadocs/MDM71/current/API/index.html>
- How to Use the Master Data Management Java API for MDM SP06 Patch 1:
<https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/b02f4f88-7bbb-2a10-67ad-d435b9f0c643>
- MDM Data Manager Reference Guide on SAP Service Marketplace:
<http://service.sap.com/instguides>.

www.sdn.sap.com/irj/sdn/howtoguides