ID-Name Mapping Using Query

Applies to:

Business Rules Framework plus shipped with **SAP NetWeaver 7.0 Enhancement Package 1**.

Summary

The tutorial explains the usage of BRFplus query object. The query object can be used to understand ID to name mapping and vice versa.

Authors: Carsten Ziegler, Shashi Kanth Kasam, Orenthung Ovung

Company: SAP

Created on: 01 September 2008

About the Authors

Carsten Ziegler is the Architect and Project Manager of Business Rules Framework plus. He joined SAP in 2000. Since then he has been working in various projects as a developer, development architect and project lead.

Shashi Kanth Kasam is a Rules Developer in the BRFplus team. He has 2 years of experience in building business rules and has been part of this team since April 2008.

Orenthung Ovung is an Information Developer in the BRFplus team. He has been part of this team since March 2008.

Table of Contents

Prerequisites

- You have a basic knowledge of BRFplus

Learning Objectives

- Use query object to obtain ID from Name and Name from ID.
- To understand that objects can uniquely be identified by their unique IDs.
- How to obtain commonly used attributes statically.

Designing Rules Using the API

Procedure

A query object is used to show the mapping of BRFplus objects' UUIDs to their respective names and vice-versa. This query object can be created as application specific or application independent.

The following methods in **IF_FDT_QUERY** help in this ID-Query mapping.

Method	Description
GET_IDS	Returns the IDs of all the BRFplus objects which have a particular name and object type The object type is an optional parameter. If the object type parameter is not passed then the method returns all the objects with the chosen name irrespective of the object type.
GET_NAME	Gives the name of the BRFplus object with a particular ID It also returns the ID of the application to which the object is attached to. If the object is unnamed, then the exporting parameter **EV_UNNAMED** is set to **ABAP_TRUE**. We can also search the memory using the parameter **IV_INCL_MEMORY**.
GET_OBJECT_TYPE	Gives the object type of the BRFplus object with a particular ID Here also, we can search memory by passing **IV_INCL_MEMORY** parameter as **ABAP_TRUE**.

Note: The methods **'GET_IDS'** and **'GET_NAME'** are used to get *Name to ID* and *ID to Name* mappings respectively.

Parameter	Description
IV_INCL_MEMORY	This parameter is used to decide if the query needs to search in the memory. If this parameter is set to **ABAP_TRUE**, then the query also searches in the memory (for unsaved objects).

Data declarations

```
DATA: lo_factory      TYPE REF TO if_fdt_factory,
      lo_application  TYPE REF TO if_fdt_application,
      lo_query        TYPE REF TO if_fdt_query,
      lv_obj_type     TYPE if_fdt_types=>object_type,
      lv_appl_id      TYPE if_fdt_types=>id,
      lv_id           TYPE if_fdt_types=>id,
      lts_obj_ids     TYPE if_fdt_types=>ts_object_id,
      lts_name        TYPE if_fdt_query=>ts_name,
      ls_name         TYPE if_fdt_query=>s_name,
      lv_name         TYPE if_fdt_types=>name,
      lv_access_lvl   TYPE if_fdt_types=>access_level,
      lv_id_unknown   TYPE abap_bool,
      lv_unnamed      TYPE abap_bool,
      lv_local        TYPE abap_bool,
      lv_customizing  TYPE abap_bool,
      lv_system       TYPE abap_bool,
      lv_masterdata   TYPE abap_bool,
      lv_obsolete     TYPE abap_bool,
      lv_delete       TYPE abap_bool,
      lv_marked_del   TYPE abap_bool.
```

Factory

Use the following code to get a reference to the instance of the factory.

```
lo_factory = cl_fdt_factory=>if_fdt_factory~get_instance( ).
```

Creating the application

Use the following code to get an application object from the factory.

```
lo_application = lo_factory->get_application( if_fdt_constants=>gc_application_tmp ).
lo_application ?= lo_application->if_fdt_transaction~copy( ).
```

You can set values for the application. The application needs to have a unique name. The **CL_FDT_SERVICE** class method is used to get the unique name.

```
lv_name = cl_fdt_services=>get_unique_name( ).
lo_function->if_fdt_admin_data~set_name( lv_name ).
```

The Application should always have a unique name. And the function should have a unique name inside an application. But there can be more than one data object with the same name.

Creating the Query Object

A query object can be created as application specific or application independent. An application specific query object can be created as

```
lo_factory = cl_fdt_factory=>if_fdt_factory~get_instance( lo_application->mv_id ).
lo_query = lo_factory->get_query( ).
```

Now this query object is confined to the application boundaries of the factory instance.

A query object can also be created as application independent by using following lines of code.

```
lo_factory = cl_fdt_factory=>if_fdt_factory~get_instance( ).
lo_query = lo_factory->get_query( ).
```

Creating an ID-Name Table

Using the following code we create an internal table that holds IDs and names of different BRFplus objects.

```
ls_name-id   = lo_application->mv_id.
ls_name-name = lo_application->if_fdt_admin_data~get_name( ).
INSERT ls_name INTO TABLE lts_name.
ls_name-id = if_fdt_obj_system_variables=>gc_expr_syuser.
ls_name-name = 'SYUSER'.
INSERT ls_name INTO TABLE lts_name.
ls_name-id = if_fdt_obj_system_variables=>gc_dobj_syuser.
ls_name-name = 'SYUNAME'.
INSERT ls_name INTO TABLE lts_name.
```

Using the Query Object

For Name to ID Mapping

Loop through the internal table that is created, as in the above step and use **'GET_IDS'** method of the **IF_FDT_QUERY** to get the IDs of all the objects that have a particular name. The *include memory* flag decides if the memory has to be queried. If the **IV_INCL_MEMORY** is **ABAP_TRUE** then the memory is queried.

```
LOOP AT lts_name INTO ls_name.
* Using GET_IDS method of IF_FDT_QUERY to get the IDs of all
* the objects which have a particular name. The 'include memory' flag decides
* if the memory is included into the query, i.e for unsaved changes.
  lo_query->get_ids( EXPORTING iv_incl_memory = abap_true
                               iv_name        = ls_name-name
                     IMPORTING ets_object_id  = lts_obj_ids ).
  WRITE:/ 'The IDs with name ', ls_name-name, 'are:'.       "#EC NOTEXT
  LOOP AT lts_obj_ids INTO lv_id.
    WRITE lv_id.
  ENDLOOP.
ENDLOOP.
```

There is an optional parameter **IV_IBJECT_TYPE** for the above method. Using this parameter we can query for a specific object type. For example, the lines of code below show the querying of IDs for the data object types.

```
LOOP AT lts_name INTO ls_name.
* now we restrict the query to data objects
  lo_query->get_ids(
             EXPORTING iv_incl_memory = abap_false
                       iv_name        = ls_name-name
                       iv_object_type = if_fdt_constants=>gc_object_type_data_object
           IMPORTING ets_object_id  = lts_obj_ids ).
  LOOP AT lts_obj_ids INTO lv_id.
    WRITE:/ 'Data Object:', ls_name-name, 'Object Id:', lv_id. "#EC NOTEXT
  ENDLOOP.
ENDLOOP.
```

For ID to Name Mapping

The **GET_NAME** method of **IF_FDT_QUERY** can be used to get the name of the object with a particular ID.

```
  LOOP AT lts_name INTO ls_name.
* We use the GET_NAME METHOD of IF_FDT_QUERY to get the name of the
* object with a particular ID.
* Returns the name of the object irrespective of the application.
  lo_query->get_name( EXPORTING iv_incl_memory = abap_true
                                iv_id          = ls_name-id
                      IMPORTING ev_name        = lv_name
                                ev_unnamed     = lv_unnamed
                                ev_id_unknown  = lv_id_unknown ).
  IF lv_name IS NOT INITIAL.
    WRITE:/ 'The name of the Object with ID ', ls_name-id, 'is', lv_name.
  ELSE.
    WRITE:/ 'Object ID',ls_name-id, ' ID unknown: ', lv_id_unknown,
                'Un-named: ', lv_unnamed.
  ENDIF.
```

The **GET_OBJECT_TYPE** method of **IF_FDT_QUERY** can be used to get the object type of the object with a particular ID.

```
  lo_query->get_object_type( EXPORTING iv_incl_memory = abap_true
                                       iv_id          = ls_name-id
                             IMPORTING ev_object_type = lv_obj_type
                                       ev_id_unknown  = lv_id_unknown ).
  WRITE: 'And Object Type is ', lv_obj_type.
ENDLOOP.
```

Static Retrieval of Object Information

The method **'GET_ID_INFORMATION'** of **CL_FDT_FACTORY** class returns some of the important attributes of the object that the id refers to. This method always includes memory search.

```
cl_fdt_factory=>get_id_information(
  EXPORTING iv_id                = if_fdt_constants=>gc_dobj_element_text
  IMPORTING ev_object_type       = lv_obj_type
            ev_name              = lv_name
            ev_access_level      = lv_access_lvl
            ev_unnamed           = lv_unnamed
            ev_local_object      = lv_local
            ev_customizing_object = lv_customizing
            ev_masterdata_object  = lv_masterdata
            ev_system_object     = lv_system
            ev_id_unknown        = lv_id_unknown
            ev_obsolete          = lv_obsolete
            ev_deleted           = lv_delete
            ev_marked_for_delete  = lv_marked_del
            ev_application_id    = lv_appl_id )
```

Parameters

Name	Description
IV_ID	The Unique Universal Identifier of the object
EV_OBJECT_TYPE	Object type
EV_NAME	Name of the object
EV_ACCESS_LEVEL	Access Level
EV_UNNAMED	Set to **ABAP_TRUE** if no name is defined for the object
EV_LOCAL_OBJECT	Set to **ABAP_TRUE** if the ID refers to a local object
EV_CUSTOMIZING_OBJECT	Set to **ABAP_TRUE** if the ID refers to a customizing object
EV_MASTERDATA_OBJECT	Set to **ABAP_TRUE** if the object is a master data object
EV_SYSTEM_OBJECT	Set to **ABAP_TRUE** if the object is a system object
EV_ID_UNKNOWN	Set to **ABAP_TRUE** if the object ID is not a BRFplus object ID
EV_OBSOLETE	Set to **ABAP_TRUE** if the object is marked as obsolete
EV_DELETED	Set to **ABAP_TRUE** if the object is logically deleted
EV_MARKED_FOR_DELETE	Set to **ABAP_TRUE** if the object is marked for delete
EV_APPLICATION_ID	ID of the application to which the object belongs to

Complete Code

```
*&---------------------------------------------------------------------*
*& Report  FDT_SDN_QUERY_MAPPING
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  FDT_SDN_QUERY_MAPPING.

DATA: lo_factory      TYPE REF TO if_fdt_factory,
      lo_application  TYPE REF TO if_fdt_application,
      lo_query        TYPE REF TO if_fdt_query,
      lv_obj_type     TYPE if_fdt_types=>object_type,
      lv_appl_id      TYPE if_fdt_types=>id,
      lv_id           TYPE if_fdt_types=>id,
      lts_obj_ids     TYPE if_fdt_types=>ts_object_id,
      lts_name        TYPE if_fdt_query=>ts_name,
      ls_name         TYPE if_fdt_query=>s_name,
      lv_name         TYPE if_fdt_types=>name,
      lv_access_lvl   TYPE if_fdt_types=>access_level,
      lv_id_unknown   TYPE abap_bool,
      lv_unnamed      TYPE abap_bool,
      lv_local        TYPE abap_bool,
      lv_customizing  TYPE abap_bool,
      lv_system       TYPE abap_bool,
      lv_masterdata   TYPE abap_bool,
      lv_obsolete     TYPE abap_bool,
      lv_delete       TYPE abap_bool,
      lv_marked_del   TYPE abap_bool.

ULINE.

FORMAT COLOR COL_HEADING.
WRITE:/ 'This example shows the mapping between IDs and names.',
        'It also shows the static retrieval of further information.'.

FORMAT COLOR OFF.
ULINE.

* we get a new object to demostrate how to query on unsaved objects
lo_factory = cl_fdt_factory=>if_fdt_factory~get_instance( ).
lo_application = lo_factory->get_application( if_fdt_constants=>gc_application_tmp ).
lo_application ?= lo_application->if_fdt_transaction~copy( ).

* setting a unique name for the application
lv_name = cl_fdt_services=>get_unique_name( ).
lo_application->if_fdt_admin_data~set_name( lv_name ).

* now we build a table with some objects for the demostration
ls_name-id   = lo_application->mv_id.
ls_name-name = lo_application->if_fdt_admin_data~get_name( ).
INSERT ls_name INTO TABLE lts_name.
ls_name-id = if_fdt_obj_system_variables=>gc_expr_syuser.
```

```
ls_name-name = 'SYUSER'.
INSERT ls_name INTO TABLE lts_name.
ls_name-id = if_fdt_obj_system_variables=>gc_dobj_syuser.
ls_name-name = 'SYUNAME'.
INSERT ls_name INTO TABLE lts_name.

* 1.1 Name to ID Mapping - Querying names from Ids.
*---------------------------------------------------------------------------------
----
FORMAT COLOR COL_HEADING.
WRITE:/ 'Name to ID Mapping: Querying Ids from names using "GET_IDS" method'.
FORMAT COLOR OFF.

* this creates an application specific query
lo_factory = cl_fdt_factory=>if_fdt_factory~get_instance( lo_application->mv_id ).
lo_query = lo_factory->get_query( ).

** this creates an application independent query
*lo_factory = cl_fdt_factory=>if_fdt_factory~get_instance( ).
*lo_query = lo_factory->get_query( ).

* if the query instance is application specific, then the objects from that applicati
on are found, else
* all the the objects regardless of the application are found.
LOOP AT lts_name INTO ls_name.
* We may use the GET_IDS method of IF_FDT_QUERY to get the IDs of all
* the objects which have a particular name. The 'include memory' flag decides
* if the memory is included into the query, i.e for unsaved changes.
  lo_query->get_ids( EXPORTING iv_incl_memory = abap_true
                               iv_name        = ls_name-name
                     IMPORTING ets_object_id  = lts_obj_ids ).
  WRITE:/ 'The IDs with name ', ls_name-name, 'are:'.
  LOOP AT lts_obj_ids INTO lv_id.
    WRITE lv_id.
  ENDLOOP.
ENDLOOP.

ULINE.

FORMAT COLOR COL_HEADING.
WRITE:/ 'Querying for "Data Element" object types from name'.
FORMAT COLOR OFF.
LOOP AT lts_name INTO ls_name.
* now we restrict the query to data objects
  lo_query->get_ids(
            EXPORTING iv_incl_memory = abap_false
                      iv_name        = ls_name-name
                      iv_object_type = if_fdt_constants=>gc_object_type_data_object
          IMPORTING ets_object_id  = lts_obj_ids ).
  LOOP AT lts_obj_ids INTO lv_id.
    WRITE:/ 'Data Object:', ls_name-name, 'Object Id:', lv_id.
  ENDLOOP.
ENDLOOP.
ULINE.

* 1.2 ID to Name Mapping - Querying Ids from names.
```

```
*---------------------------------------------------------------------------------
----
FORMAT COLOR COL_HEADING.
WRITE:/ 'ID to Name Mapping: Querying names (using "GET_NAME") and Object types (usin
g "GET_OBJECT_TYPE") from Ids'.
FORMAT COLOR OFF.


LOOP AT lts_name INTO ls_name.
* We use the GET_NAME METHOD of IF_FDT_QUERY to get the name of the
* object with a particular ID.
* Returns the name of the object irrespective of the application.
  lo_query->get_name( EXPORTING iv_incl_memory = abap_true
                                iv_id          = ls_name-id
                      IMPORTING ev_name        = lv_name
                                ev_unnamed     = lv_unnamed
                                ev_id_unknown  = lv_id_unknown ).
  IF lv_name IS NOT INITIAL.
    WRITE:/ 'The name of the Object with ID ', ls_name-id, 'is', lv_name.
  ELSE.
    WRITE:/ 'Object ID',ls_name-id, ' ID unknown: ', lv_id_unknown,
       'Un-named: ', lv_unnamed.
  ENDIF.
* We use the GET_OBJECT_TYPE method of IF_FDT_QUERY to get the object type
* of the object with a particular ID.
* Returns the object type irrespective of the application.
  lo_query->get_object_type( EXPORTING iv_incl_memory = abap_true
                                       iv_id          = ls_name-id
                             IMPORTING ev_object_type = lv_obj_type
                                       ev_id_unknown  = lv_id_unknown ).
  WRITE: 'And Object Type is ', lv_obj_type.
ENDLOOP.

ULINE.

* 2.0 Static Object Information Retrieval
*---------------------------------------------------------------------------------
----
FORMAT COLOR COL_HEADING.
WRITE:/
   'Usage of "GET_ID_INFORMATION" to retrieve important attributes of the object. '.
FORMAT COLOR OFF.
* Method GET_ID_INFORMATION in CL_FDT_FACTORY can be used to retrieve
* some important information for an ID. This methods always includes the
* memory (unsaved changes).
cl_fdt_factory=>get_id_information(
  EXPORTING iv_id                = if_fdt_constants=>gc_dobj_element_text
  IMPORTING ev_object_type       = lv_obj_type
            ev_name              = lv_name
            ev_access_level      = lv_access_lvl
            ev_unnamed           = lv_unnamed
            ev_local_object      = lv_local
            ev_customizing_object = lv_customizing
            ev_masterdata_object  = lv_masterdata
            ev_system_object     = lv_system
            ev_id_unknown        = lv_id_unknown
```

```
            ev_obsolete           = lv_obsolete
            ev_deleted            = lv_delete
            ev_marked_for_delete  = lv_marked_del
            ev_application_id     = lv_appl_id ).

  WRITE: / 'Following are the attributes of the object:',
                 if_fdt_constants=>gc_element_type_text,
         / 'Object Name:', lv_name,
         / 'Object Type:', lv_obj_type,
         / 'Access Level:', lv_access_lvl,
         / 'Application Id:', lv_appl_id.

  IF lv_id_unknown EQ abap_true.
    WRITE:/ 'This is not a BRFplus object'.
  ENDIF.

  IF lv_unnamed EQ abap_true.
    WRITE:/ 'Unnamed: TRUE'.
  ELSE.
    WRITE:/ 'Unnamed: FALSE'.
  ENDIF.

  IF lv_local EQ abap_true.
    WRITE:/ 'Local Object: TRUE'.
  ELSE.
    WRITE:/ 'Local Object: FALSE'.
  ENDIF.

  IF lv_customizing EQ abap_true.
    WRITE:/ 'Customizing Object: TRUE'.
  ELSE.
    WRITE:/ 'Customizing Object: FALSE'.
  ENDIF.

  IF lv_masterdata EQ abap_true.
    WRITE:/ 'Master Data Object: TRUE'.
  ELSE.
    WRITE:/ 'Master Data Object: FALSE'.
  ENDIF.

  IF lv_system EQ abap_true.
    WRITE:/ 'System Object: TRUE'.
  ELSE.
    WRITE:/ 'System Object: FALSE'.
  ENDIF.

  IF lv_id_unknown EQ abap_true.
    WRITE:/ 'Id Unknown: TRUE'.
  ELSE.
    WRITE:/ 'Id Unknown: FALSE'.
  ENDIF.

  IF lv_obsolete EQ abap_true.
    WRITE:/ 'Marked as Obsolete: TRUE'.
  ELSE.
    WRITE:/ 'Marked as Obsolete: FALSE'.
```

```
ENDIF.

IF lv_delete EQ abap_true.
  WRITE:/ 'Logically Deleted: TRUE'.
ELSE.
  WRITE:/ 'Logically Deleted: FALSE'.
ENDIF.

IF lv_marked_del EQ abap_true.
  WRITE:/ 'Marked For Delete: TRUE'.
ELSE.
  WRITE:/ 'Marked For Delete: FALSE'.
ENDIF.
```

Related Content

- BRFplus – The Very Basics
- How to Create and Process a Simple Rule
- How to Create Formula Functions
- Using BRFplus with a Third-Party Rules Engine
- Wikipedia, Business Rules: http://en.wikipedia.org/wiki/Business_rules
- Wikipedia, Business Rule Management System: http://en.wikipedia.org/wiki/Business_Rule_Management_System
- Carsten Ziegler, About Business Rules: https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/9713
- Carsten Ziegler, BRFplus a Business Rule Engine written in ABAP, https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/8889
- Carsten Ziegler, Important Information for Using BRFplus

  https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/11632
- Rajagopalan Narayanan, Business Rules and Software Requirements, https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/40aae118-42c2-2a10-fcaf-fdd9d30bcb1a
- Rajagopalan Narayanan, Seven Tips for Your First Business Rules Project, https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/201a9e3d-3ec2-2a10-85b2-ce56d276dd7a
- Rajagopalan Narayanan, Real World Return of Investment Scenarios with Business Rules Management, https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/b050905e-3cc2-2a10-979a-81a57a787f56
- Rajagopalan Narayanan, Five Reasons to Build Agile Systems Using Business Rules Management Functionality, https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/504486eb-43c2-2a10-f5a7-e84ef3fd45be
- Rajagopalan Narayanan, How Business Rules Management Functionality Helps Tariff Plans Management in Transportation and Shipping, https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/40a9cf69-40c2-2a10-8a8b-969fb311dd31
- Rajagopalan Narayanan, Getting Started with Business Rules Management, https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/70c669d8-3ac2-2a10-0e96-c7c3786168f0

Copyright