Applies to:
SAP R/3, SAP ECC 6.0 and SAP BI NetWeaver 2004s. For more information, visit the Business Intelligence homepage.

Summary
We all struggle with generic extraction with Function Modules. Here I am trying to simplify things around it. If you have primary ABAP knowledge and want to learn basics of writing function module for extraction, this paper will definitely help you doing that.

Author:  Ashish Tewari
Company:  Deloitte Consulting India Pvt. Ltd
Created on: 03 July 2009

Author Bio
Ashish Tewari is currently working with Deloitte. He is a Business Intelligence Solution Consultant (NetWeaver 2004) with 4 years of experience in SAP BW space.
Table of Contents
Introduction .........................................................................................................................................................3
  Few important Parameters:...........................................................................................................................................3
Steps for creating generic datasource from FM: ........................................................................................................3
Steps for creating your FM from RSAX_BIW_GET_DATA_SIMPLE: ............................................................................4
  Table: .................................................................................................................................................................4
  Auxiliary Selection and Max no of lines: .................................................................................................................4
  Select/Define Ranges: ............................................................................................................................................4
  After Ranges: .......................................................................................................................................................5
  Buffer and populate field list...................................................................................................................................5
  Now write 'ELSE' statement...................................................................................................................................5
  Filling Range Tables: .............................................................................................................................................5
  Opening curser: ....................................................................................................................................................6
  Fetch records into interface table:..........................................................................................................................7
  Now increment the datapackage id: ........................................................................................................................7
Related Content......................................................................................................................................................8
Copyright..............................................................................................................................................................8

Error! Bookmark not defined.
Introduction

In this article, I will discuss details of Function Module based Generic Extraction. This will provide insight into the various elements of such extraction. It will also help users to build their own Datasources around Function Module. I understand that there are many posts around this topic still I hope reader will benefit from this.

Reference/Sample FM: RSAX_BIW_GET_DATA_SIMPLE and RSAX_BIW_GET_DATA.

<table>
<thead>
<tr>
<th>Function Module</th>
<th>Extraction Method</th>
<th>Data Load</th>
<th>RSO2 Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSAX_BIW_GET_DATA_SIMPLE</td>
<td>F2: Function Module (Simple Interface)</td>
<td>Full Load preferable</td>
<td>No effect of delta settings</td>
</tr>
<tr>
<td>RSAX_BIW_GET_DATA</td>
<td>F1: Function Module (Complete Interface)</td>
<td>Delta Load possible</td>
<td>Standard delta settings</td>
</tr>
</tbody>
</table>

Note: These Function Modules will be called several times in succession during an extraction process.

1. Initialize Call up: Only the request control parameters are transferred. No data transfer takes place in this call. (It is first call of FM. Do not confuse it with initialization of Datasource).
2. First Read call up: Now according to I_MAXSIZE, data is returned as per the extract structure.
3. Second read call up: Data returned according to I_MAXSIZE (if first read call reaches its limit)
4. Till NO_MORE_DATA exception is raised this FM will be iterated as per the package size.

Few important Parameters:

- I_DSOURCE is for Datasource
- I_INITFLAG is for Initialize call up: When the FM is called for the first time, this parameter is set to ’X’, afterwards it is set to ’ ’ (blank).
- I_MAXSIZE is Package size: This parameter contains the no of rows expected for a read call up.
- I_REQUNR Request number

Steps for creating generic datasource from FM:

1. Copy and modify RSAX_BIW_GET_DATA_SIMPLE as per your data load requirements. Change name and assign it to your Function Group.
2. Go to the ‘Tables’ tab and associate E_T_DATA with your structure (which you created for extraction)
3. Navigate to the ‘Source Code’ tab and declare/code items.
4. Use this function module and pre defined extraction structure in RSO2 for creating generic Datasource.
Steps for creating your FM from RSAX_BIW_GET_DATA_SIMPLE:

Table:

Give table names for which you are creating Datasource
  e.g. TABLES: VBAK, VBAP.

Auxiliary Selection and Max no of lines:

Provide details for maximum number of lines (rows in datapackage) from database table. This is optional.

  * Auxiliary Selection criteria structure
    DATA: L_S_SELECT TYPE SRSC_S_SELECT.
  * Maximum number of lines for DB table
    STATICS: S_S_IF TYPE SRSC_S_IF_SIMPLE,
  * counter
    S_COUNTER_DATAPAKID LIKE SY-TABIX,
  * cursor
    S_CURSOR TYPE CURSOR.

Select/Define Ranges:

We are required to pass sign, option, low and high. It is for those fields/objects, which you will use for selection in the infopackage.

Standard syntax for Ranges:

  RANGES rtab FOR dobj [occurs n].
  DATA:
    BEGIN of rtab OCCURES {10/n},
    sign TYPE c LENGTH 1,
    option TYPE c LENGTH 2,
    low LIKE dobj,
    high LIKE dobj,
    END of rtab.

We'll not use this complete syntax in these function modules, this is just for your information about Ranges. We will use it in this way to declare:

  RANGES: L_R_VBELN FOR VBAK-VBELN.
  L_R_AUDAT FOR VBAK-AUDAT.
  L_R_POSNR FOR VBAP-POSNR.

BW will pass down simple selection criterion of type SIGN = I (include) and OPTION = EQ (equal) or OPTION = BT (between).

Now, say you need sales document no (VBELN) and item no of sales document (POSNR) for selection, so you have to take them in RANGES.

When you'll pass parameters for certain Document no or item no, it will fetch accordingly.
After Ranges:
Check for the first call or initialization of FM
It will check input parameters, buffer them and prepare data selection.

* Initialization mode (first call by SAPI) or data transfer mode
* (following calls) ?
  IF i_initflag = sbiwa_c_flag_on.
************************************************************************
* Initialization: check input parameters
*                 buffer input parameters
*                 prepare data selection
************************************************************************
* Check datasoure validity
* with CASE statement in RSAX_BIW_GET_DATA_SIMPLE.
CASE I_DSOURCE.
  WHEN '<datasource name>'.
  WHEN OTHERS.
    IF 1 = 2. MESSAGE E009(R3). ENDIF.
  ENDcase.

* this is a typical log call. Please write every error message like this
  LOG_WRITE 'E' 'R3' '009' I_DSOURCE ' '.
  RAISE ERROR_PASSED_TO_MESS_HANDLER.
ENDcase.

Buffer and populate field list
APPEND LINES OF I_T_SELECT TO S_S_IF-T_SELECT.
Fill these control parameter buffer for data extraction calls:

  S_S_IF-REQUNR = I_REQUNR.
  S_S_IF-DSOURCE = I_DSOURCE.
  S_S_IF-MAXSIZE = I_MAXSIZE.

Fill field list table for an optimized select statement (in case that there is no 1:1 relation between InfoSource fields and database table fields this may be far from being trivial)
APPEND LINES OF I_T_FIELDS TO S_S_IF-T_FIELDS.
These interfaces are used for buffering and populating field list.

Now write ‘ELSE’ statement.
This is for second call of FM. First call was written with
IF I_INITFLAG = SBIWA_C_FLAG_ON.
So, when you write ‘ELSE’, it tells system about the mode, which is initialization or data extraction. For first call of data transfer, you need to open curser and then start fetching of data. For subsequent calls (if any), no need to open curser again, just fetch.

Filling Range Tables:
For initial datapackage, you need to fill ranges. BW will pass down only simple selection criteria of the type: SIGN = 'I' and OPTION = 'EQ' or 'BT'.
Now you have to move all selected (this selection will be provided by BW) values in pre-declared ranges tables from their base tables. During final selection of data for sending to BW, you’ll code in such a way that system will bundle data according to values in ranges tables.

```plaintext
IF S_COUNTER_DATAPAKID = 0.
  LOOP AT S_S_IF-T_SELECT INTO L_S_SELECT WHERE FIELDNM = 'VBELN'.
    MOVE-CORRESPONDING L_S_SELECT TO L_R_VBELN.
    APPEND L_R_VBELN.
  ENDLOOP.

  LOOP AT S_S_IF-T_SELECT INTO L_S_SELECT WHERE FIELDNM = 'POSNR'.
    MOVE-CORRESPONDING L_S_SELECT TO L_R_POSNR.
    APPEND L_R_POSNR.
  ENDLOOP.

Opening cursor:
This is opening of a connection with database. This is very important step so it needs to be understood carefully.

Syntax:
```plaintext
OPEN CURSOR [WITH HOLD] S_CURSOR FOR
  SELECT result
  FROM source
  [[FOR ALL ENTRIES IN itab WHERE sql_cond]
  [GROUP BY group] [HAVING group_cond]
  [ORDER BY sort_key].
```n
This statement opens a database cursor for the selection defined after FOR, and links a cursor variable S_CURSOR with this database cursor (database pointer).

For S_CURSOR, a declared variable with the specific predefined data type cursor must be entered. After the OPEN CURSOR statement, the database cursor is positioned in front of the first line of the resulting set.

After FOR, the syntax of a SELECT statement can be entered, which contains all the additions of the normal SELECT statement, except for INTO and APPENDING. In the addition result, the addition SINGLE can also not be used after SELECT.

If the addition WITH HOLD is specified, the database cursor is not closed in an explicitly triggered database commit or database rollback, for example Native SQL. The addition WITH HOLD cannot be specified if the cursor is to be opened for a secondary database connection.

In our case, it will be like this:
```plaintext
OPEN CURSOR WITH HOLD S_CURSOR FOR
  SELECT VBAK~VBELN VBAK~AUDAT VBAK~AUGRU VBAK~KUNNR VBAK~VBUR
  VBAK~AWAHR VBAK~KVGR1 VBAK~KVGR2 VBAK~KVGR3
  VBAK~VBELN VBAP~POSNR VBAP~MATNR VBAP~KWMENG
  VBAP~VRKME VBAP~NETWR VBAP~WAERK
  FROM VBAK
  INNER JOIN VBAP ON VBAP~VBELN = VBAK~VBELN
  WHERE VBAK~VBELN IN L_R_VBELN AND
  VBAP~POSNR IN L_R_POSNR.
```

Note: Remember to keep max no of lines, counter and curser under static declaration. These parameters are maintained as static because we need them with fix values during iterations of the FM
**Fetch records into interface table:**

```plaintext
FETCH NEXT CURSOR S_CURSOR
APPENDING CORRESPONDING FIELDS
OF TABLE E_T_DATA
PACKAGE SIZE S_S_IF-MAXSIZE.

IF SY-SUBRC <> 0.
   CLOSE CURSOR S_CURSOR.
   RAISE NO_MORE_DATA.
ENDIF.
```

SY-SUBRC = 0 At least one line was extracted.
= 4 No line was extracted from the resulting set.

**Note:** Remember to keep exception NO_MORE_DATA, otherwise function calls will not stop.

**Now increment the datapackage id:**

```plaintext
S_COUNTER_DATAPAKID = S_COUNTER_DATAPAKID + 1.
ENDIF.  
```

This is ‘ELSE’ closes If-Else condition started by ‘IF I_INITFLAG = SBIWA_C_FLAG_ON.’ and checks whether it is an initialization mode or extraction mode during FM call.

Well that’s it, this Function Module is ready to use, just debug once in RSA3 to check for any inconsistencies.

*How does the RSAX_BIW_GET_DATA works and what about delta? I will discuss this in my next entry, part 2 of this multi-part series on Function Module based Generic Datasource.*
Related Content

Generic Delta Function Module

Generic Extraction via Function Module

Function Module: Interface Description and Procedure

For more information, visit the Business Intelligence 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.