SDN Community Contribution

(This is not an official SAP document.)

Disclaimer & Liability Notice

RFC_READ_TABLE for SAP Web AS

Applies To:

SAP Web Application Server 6.20 and above

Summary

When working with RFC enabled Function Modules, you would have noticed that you always have to use predefined structures. This is a limiting factor when you want the output to be dynamic. For example RFC_READ_TABLE function can read any table (with/without restrictions on what fields you want to output) but the output table (DATA) is not a neat one to work with. With the introduction of XSLT programming and XML transformation, handling this scenario has become easier as you now have the ability to output the result in XML format, which any system can read and use. In this article I am going to show you how a similar function module can be created which can output the results in XML format and how an ABAP program can take that XML result and convert it to an ABAP internal table.

By: Durairaj Athavan Raja

Company: Atos Origin Middle East

Date: 20 July 2005

Table of Contents

Introduction

As I have explained in the summary, we will be creating a function module which works like RFC_READ_TABLE and outputs the result in XML format and create a XSLT program to convert the resulting XML into an ABAP internal table.

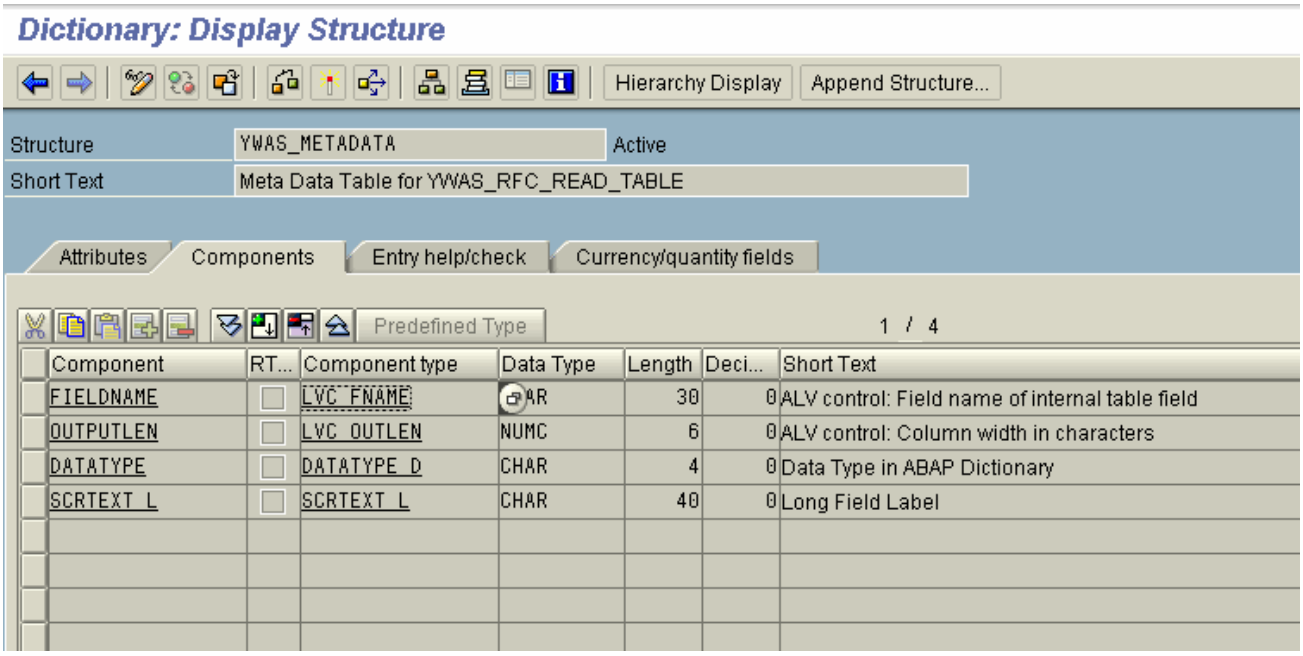
Following are the steps we will be performing.

1. Create a function module and name it YWAS_RFC_READ_TABLE.
2. Create a generic XSLT program to convert XML produced by YWAS_RFC_READ_TABLE into ABAP internal table.
3. Create a report program to test the all the pieces together.

The New RFC_READ_TABLE

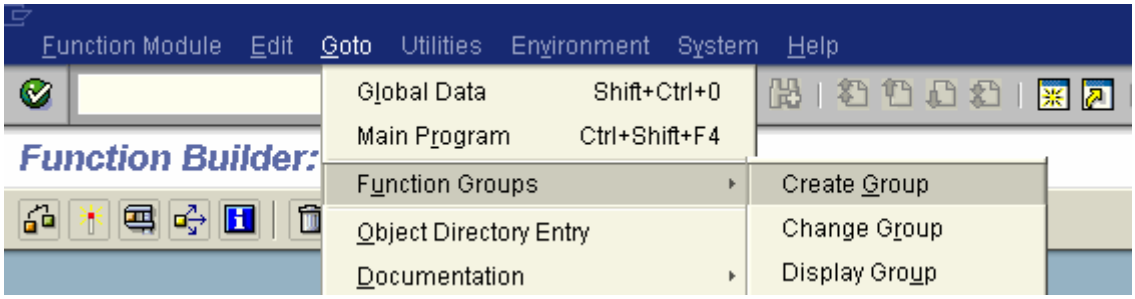
1. First we need to create a structure which will hold the Meta data of the XML output. Got to transaction SE11 and create a structure as you see below with the same name as below. (YWAS_METADATA).

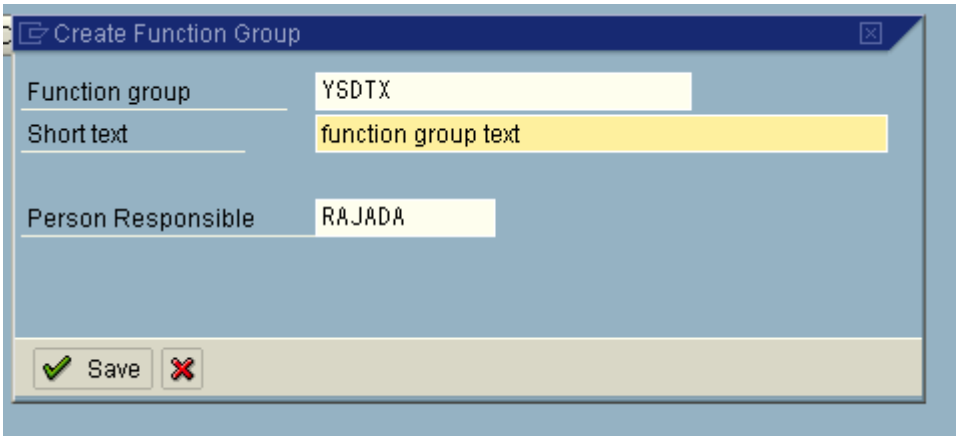
   Note: if you create the structure with a different name, please make sure that you use that name in the XSLT program which we will be creating later in this exercise.



2. Now we will create the new function module. Create function Group (YSDTX). Transaction SE37 –>Go to->Function Groups->Create Group**.**

3.

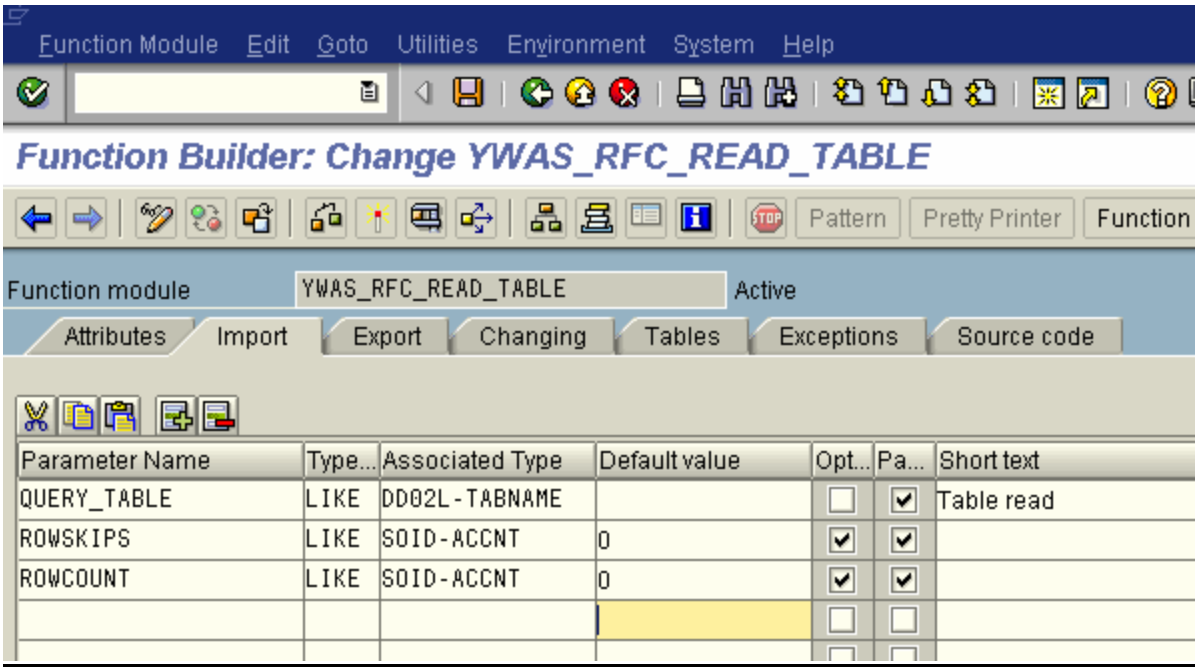




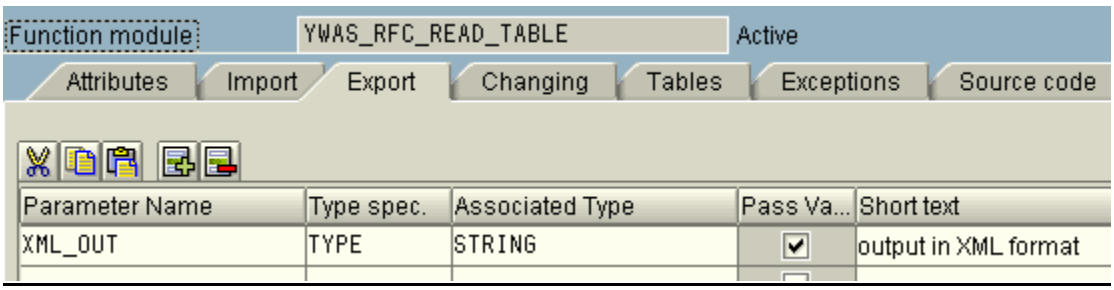
4. Now create the function module. Transaction se37 enter the name (YWAS_RFC_READ_TABLE) and click **create**. This would prompt you for assignment of function group. Enter the newly created function group (YSDTX) and hit Save. (you can also mark it as RFC enabled in the attributes screen of the FM).

5. Declare the import/export/tables and exception parameters as shown in the screen dump below.

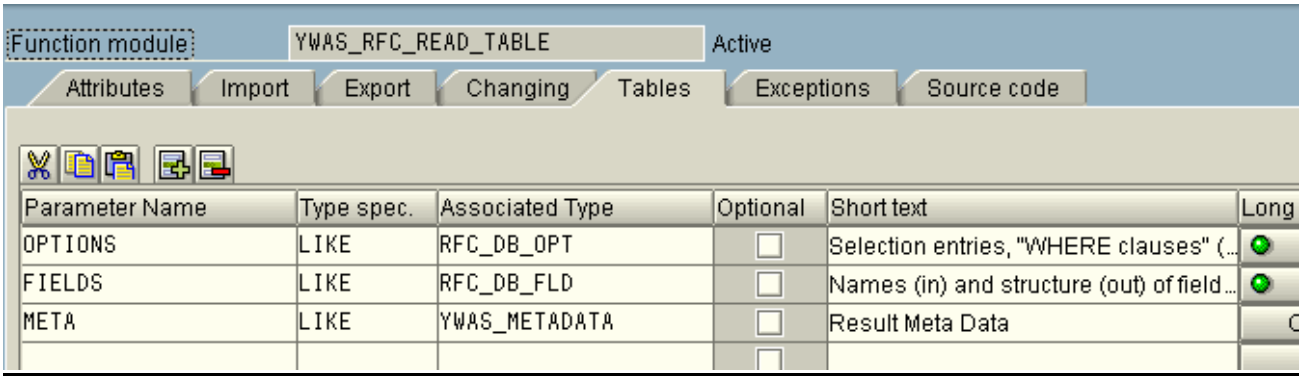
Import Parameters:



EXPORT Parameters:

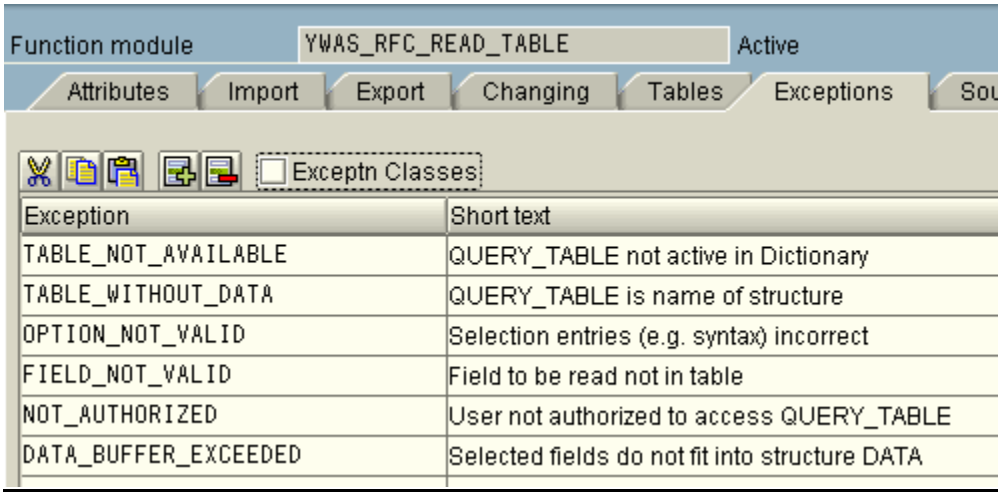


TABLES Parameter:



Parameter Name	Type spec.	Associated Type	Optional	Short text	Long
OPTIONS	LIKE	RFC_DB_OPT	☐	Selection entries, "WHERE clauses" (...	
FIELDS	LIKE	RFC_DB_FLD	☐	Names (in) and structure (out) of field...	
META	LIKE	YWAS_METADATA	☐	Result Meta Data	
			☐		

Exception Tab:



Exception	Short text
TABLE_NOT_AVAILABLE	QUERY_TABLE not active in Dictionary
TABLE_WITHOUT_DATA	QUERY_TABLE is name of structure
OPTION_NOT_VALID	Selection entries (e.g. syntax) incorrect
FIELD_NOT_VALID	Field to be read not in table
NOT_AUTHORIZED	User not authorized to access QUERY_TABLE
DATA_BUFFER_EXCEEDED	Selected fields do not fit into structure DATA

6. Now go to the source code table and copy/paste the following code.

```
FUNCTION ywas_rfc_read_table.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(QUERY_TABLE) LIKE  DD02L-TABNAME
```

```
*"     VALUE(ROWSKIPS) LIKE  SOID-ACCNT DEFAULT 0
*"     VALUE(ROWCOUNT) LIKE  SOID-ACCNT DEFAULT 0
*"  EXPORTING
*"     VALUE(XML_OUT) TYPE  STRING
*"  TABLES
*"      OPTIONS STRUCTURE  RFC_DB_OPT
*"      FIELDS STRUCTURE  RFC_DB_FLD
*"      META STRUCTURE  YWAS_METADATA
*"  EXCEPTIONS
*"      TABLE_NOT_AVAILABLE
*"      TABLE_WITHOUT_DATA
*"      OPTION_NOT_VALID
*"      FIELD_NOT_VALID
*"      NOT_AUTHORIZED
*"      DATA_BUFFER_EXCEEDED
*"----------------------------------------------------------------------

  DATA: it_fieldcat TYPE lvc_t_fcat,
          is_fieldcat LIKE LINE OF it_fieldcat.
  DATA: new_table TYPE REF TO data.
  DATA: new_line  TYPE REF TO data.
  FIELD-SYMBOLS: <ltable> TYPE ANY TABLE,
                 <l_line>  TYPE ANY,
                 <l_field> TYPE ANY.

  CALL FUNCTION 'VIEW_AUTHORITY_CHECK'
    EXPORTING
      view_action                    = 'S'
      view_name                      = query_table
    EXCEPTIONS
      no_authority                   = 2
      no_clientindependent_authority = 2
      no_linedependent_authority     = 2
      OTHERS                         = 1.
```

```
  IF sy-subrc = 2.
    RAISE not_authorized.
  ELSEIF sy-subrc = 1.
    RAISE table_not_available.
  ENDIF.

* ---------------------------------------------------------------------
*  find out about the structure of QUERY_TABLE
* ---------------------------------------------------------------------
  DATA: table_structure TYPE STANDARD TABLE OF dfies .
  DATA: wa_table_structure LIKE LINE OF table_structure .
  "DATA TABLE_HEADER LIKE X030L.
  DATA table_type TYPE dd02v-tabclass.

  CALL FUNCTION 'DDIF_FIELDINFO_GET'
    EXPORTING
      tabname              = query_table
*   FIELDNAME            = ' '
*   LANGU                = SY-LANGU
*   LFIELDNAME           = ' '
*   ALL_TYPES            = ' '
*   GROUP_NAMES          = ' '
    IMPORTING
*   X030L_WA             =
      ddobjtype            = table_type
*   DFIES_WA             =
*   LINES_DESCR          =
    TABLES
      dfies_tab            = table_structure
*   FIXED_VALUES         =
    EXCEPTIONS
      not_found            = 1
      internal_error       = 2
      OTHERS               = 3
            .
```

```
  IF sy-subrc <> 0.
    RAISE table_not_available.
  ENDIF.
  IF table_type = 'INTTAB'.
    RAISE table_without_data.
  ENDIF.


* ---------------------------------------------------------------------
*  if FIELDS are not specified, read all available fields
* ---------------------------------------------------------------------
  IF fields[] IS INITIAL .
    CLEAR wa_table_structure .
    LOOP AT table_structure INTO wa_table_structure.
      MOVE wa_table_structure-fieldname TO fields-fieldname.
      APPEND fields.
    ENDLOOP.
  ENDIF.

* ---------------------------------------------------------------------
*  for each field which has to be read, copy structure information
*  into tables FIELDS_INT (internal use) and FIELDS (output)
* ---------------------------------------------------------------------
  TYPES: BEGIN OF fields_int_type,
           fieldname  LIKE wa_table_structure-fieldname,
           type       LIKE wa_table_structure-inttype,
           decimals   LIKE wa_table_structure-decimals,
           length_src LIKE wa_table_structure-intlen,
           length_dst LIKE wa_table_structure-leng,
           offset_src LIKE wa_table_structure-offset,
           offset_dst LIKE wa_table_structure-offset,
         END OF fields_int_type .
  DATA: fields_int TYPE STANDARD TABLE OF fields_int_type .
  DATA: wa_fields_int LIKE LINE OF fields_int .
```

```
*  for each field which has to be read ...
  CLEAR wa_fields_int .
  LOOP AT fields .
    CLEAR wa_table_structure .
    READ TABLE table_structure INTO wa_table_structure WITH KEY fieldname =
fields-fieldname.
    IF sy-subrc NE 0.
      RAISE field_not_valid.
    ENDIF.



* ... copy structure information into tables FIELDS_INT
* (which is used internally during SELECT) ...
    wa_fields_int-fieldname  = wa_table_structure-fieldname.
    wa_fields_int-length_src = wa_table_structure-intlen.
    wa_fields_int-length_dst = wa_table_structure-leng.
    wa_fields_int-offset_src = wa_table_structure-offset.
    wa_fields_int-type       = wa_table_structure-inttype.
    wa_fields_int-decimals   = wa_table_structure-decimals.
    APPEND wa_fields_int TO fields_int .

* ... and into table FIELDS (which is output to the caller)
    fields-fieldtext = wa_table_structure-fieldtext.
    fields-type      = wa_table_structure-inttype.
    fields-length    = wa_fields_int-length_dst.
    MODIFY fields.

  ENDLOOP.

  DATA: BEGIN OF work, buffer(30000), END OF work.

  FIELD-SYMBOLS: <wa> TYPE ANY .
  ASSIGN work TO <wa> CASTING TYPE (query_table).

  IF rowcount > 0.
```

```
    rowcount = rowcount + rowskips.
  ENDIF.

  CLEAR meta.
  REFRESH meta .
  LOOP AT fields .
    MOVE: fields-fieldname TO meta-fieldname ,
          fields-length TO meta-outputlen ,
          fields-type TO meta-datatype ,
          fields-fieldtext TO meta-scrtext_l .
    APPEND meta .
    CLEAR meta .
  ENDLOOP .

  CLEAR: is_fieldcat .
  REFRESH it_fieldcat .
  LOOP AT meta .
    is_fieldcat-fieldname = meta-fieldname .
    is_fieldcat-datatype = meta-datatype.
    is_fieldcat-outputlen = meta-outputlen .
    is_fieldcat-scrtext_l = meta-scrtext_l.
    APPEND is_fieldcat TO it_fieldcat.
    CLEAR : is_fieldcat .

  ENDLOOP .

  CALL METHOD cl_alv_table_create=>create_dynamic_table
    EXPORTING
*    I_STYLE_TABLE             =
     it_fieldcatalog           = it_fieldcat
*    I_LENGTH_IN_BYTE          =
    IMPORTING
     ep_table                  = new_table
*    E_STYLE_FNAME             =
    EXCEPTIONS
```

```
      generate_subpool_dir_full = 1
      OTHERS                    = 2
          .
  IF sy-subrc <> 0.
    MESSAGE e000(00) WITH 'Error creating internal table' .
  ENDIF.

  ASSIGN new_table->* TO <ltable>.
  CREATE DATA new_line LIKE LINE OF <ltable>.
  ASSIGN new_line->* TO <l_line>.

  SELECT * FROM (query_table) INTO <wa> WHERE (options).

    IF sy-dbcnt GT rowskips.
      MOVE-CORRESPONDING <wa> TO <l_line> .
      INSERT <l_line> INTO TABLE <ltable>.

      IF rowcount > 0 AND sy-dbcnt GE rowcount. EXIT. ENDIF.

    ENDIF.

  ENDSELECT.
  CLEAR xml_out .
  IF NOT <ltable>[] IS INITIAL .

    CALL TRANSFORMATION (`ID`)
             SOURCE meta   = meta[]
                    output = <ltable>[]
             RESULT XML xml_out.
  ENDIF .

ENDFUNCTION.
```

7. **Save and activate** the function module (along with the function group). This function module when executed would return the results as XML (**xml_out** parameter).

8. Now as the result is in XML format this function module can be called by any system. For our example we will call it from within the same system and see how we can convert this xml into ABAP internal table.

Generic XSLT Program to Convert XML to ABAP Internal Table

Create a XSLT program (call it Y_RRT_TRANSFORM) from SE80->Edit Object->XSLT program (WAS6.2) or Transformation (WAS6.4).

Note: If you are on WAS6.4 choose the transformation type as XSLT Program when creating the XSLT program.

Copy paste the following code and save and activate.

Note: I assume that you have created the structure explained in step 1 with the name YWAS_METADATA. If you have used a different name, incorporate the same in this XSLT program.

XSLT Program Code

```
<?xml version="1.0" encoding="iso-8859-1"?>


<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:msxsl="urn:schemas-microsoft-com:xslt" xmlns:xa="urn:schemas-microsoft-
com:xml-analysis:mddataset" xmlns:asx="http://www.sap.com/abapxml"
version="1.0">
<xsl:output method="xml" />
<xsl:variable name="fieldNames" select="//asx:abap/asx:values/FIELDCAT" />
<xsl:template match="/">
<asx:abap xmlns:asx="http://www.sap.com/abapxml" version="1.0">
<asx:values>
<!--FIELDCAT>
<xsl:apply-templates select="//YWAS_METADATA" />
</FIELDCAT-->


<OUTTAB>
<xsl:apply-templates select="//item" />
</OUTTAB>
  </asx:values>
</asx:abap>
```

```
</xsl:template>


<xsl:template match="//YWAS_METADATA">
<item>
<xsl:for-each select="*">

<!--column ><xsl:apply-templates/></column-->

<xsl:copy>
    <xsl:apply-templates select="@*|*|text()"/>
  </xsl:copy>

</xsl:for-each>
</item>

</xsl:template>

<xsl:template match="//item">
<item>
<xsl:for-each select="*">

<!--column ><xsl:apply-templates/></column-->

<xsl:copy>
    <xsl:apply-templates select="@*|*|text()"/>
  </xsl:copy>

</xsl:for-each>
</item>
</xsl:template>

</xsl:stylesheet>
```

Sample Program to Test the Whole Application

To test this all the pieces together, create a report program from SE38 and copy paste the following code.

```
*&---------------------------------------------------------------------*
*& Report  Y_RRT_TEST
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  y_rrt_test.

TYPE-POOLS sscr.
DATA: restrict TYPE sscr_restrict,
      opt_list TYPE sscr_opt_list,
      ass      TYPE sscr_ass.
DATA: options TYPE STANDARD TABLE OF rfc_db_opt ,
      fields TYPE STANDARD TABLE OF rfc_db_fld  ,
      meta TYPE STANDARD TABLE OF ywas_metadata  ,
      xml_out TYPE string .
DATA: wa_options LIKE LINE OF options,
      wa_fields LIKE LINE OF fields ,
      wa_meta LIKE LINE OF meta .

FIELD-SYMBOLS: <outtab> TYPE ANY TABLE,
  <l_line>  TYPE ANY,
  <l_field> TYPE ANY.
DATA: new_table TYPE REF TO data.
DATA: new_line  TYPE REF TO data.
DATA: xslt_error TYPE REF TO cx_xslt_exception ,
      xslt_message TYPE string .

DATA: it_fieldcat TYPE lvc_t_fcat,
```

```
          is_fieldcat LIKE LINE OF it_fieldcat.

PARAMETERS: tab TYPE dd02l-tabname OBLIGATORY ,
            r_skips TYPE soid-accnt ,
            r_count TYPE soid-accnt .

SELECT-OPTIONS: opt FOR wa_options-text NO INTERVALS,
                flds FOR wa_fields-fieldname NO INTERVALS .

INITIALIZATION .
  CLEAR restrict .
  CLEAR opt_list.
  MOVE 'EQ'  TO opt_list-name.
  opt_list-options-eq = 'X'.
  APPEND opt_list TO restrict-opt_list_tab.
  ass-kind    = 'S'.
  ass-name    = 'OPT'.
  ass-sg_main = 'I'.
  ass-sg_addy = ' '.
  ass-op_main = 'EQ'.
  ass-op_addy = 'EQ'.
  APPEND ass TO restrict-ass_tab.
  CLEAR ass.
  ass-kind    = 'S'.
  ass-name    = 'FLDS'.
  ass-sg_main = 'I'.
  ass-sg_addy = ' '.
  ass-op_main = 'EQ'.
  ass-op_addy = 'EQ'.
  APPEND ass TO restrict-ass_tab.


  CALL FUNCTION 'SELECT_OPTIONS_RESTRICT'
    EXPORTING
      restriction = restrict
```

```
    EXCEPTIONS
      OTHERS      = 1.

START-OF-SELECTION .

  IF NOT opt[] IS INITIAL .
    LOOP AT opt .
      CLEAR wa_options .
      MOVE: opt-low TO wa_options-text .
      APPEND wa_options TO options .
      CLEAR wa_options .
    ENDLOOP .
  ENDIF .

  IF NOT flds[] IS INITIAL .
    LOOP AT flds .
      CLEAR wa_fields .
      MOVE: flds-low TO wa_fields-fieldname .
      APPEND wa_fields TO fields .
      CLEAR wa_fields .
    ENDLOOP .
  ENDIF .
  CALL FUNCTION 'YWAS_RFC_READ_TABLE'
    EXPORTING
      query_table          = tab
      rowskips             = r_skips
      rowcount             = r_count
    IMPORTING
      xml_out              = xml_out
    TABLES
      OPTIONS              = options
      fields               = fields
      meta                 = meta
    EXCEPTIONS
      table_not_available  = 1
```

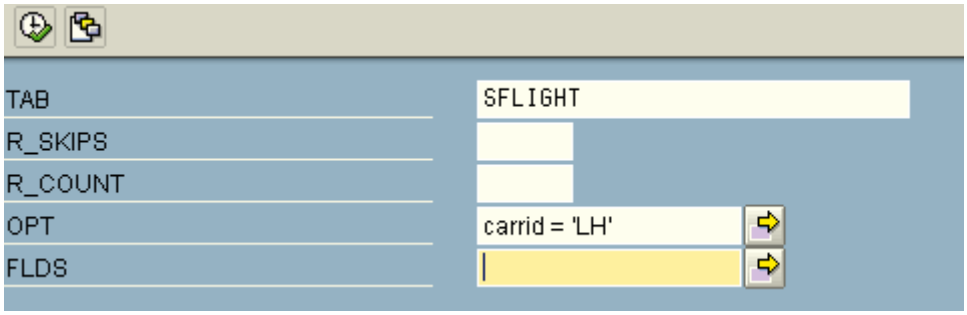
```
    table_without_data   = 2
    option_not_valid     = 3
    field_not_valid      = 4
    not_authorized       = 5
    data_buffer_exceeded = 6
    OTHERS               = 7.
IF sy-subrc <> 0.
  MESSAGE e000(00) WITH 'Error' .
ENDIF.

IF NOT meta[] IS INITIAL .
  CLEAR: is_fieldcat .
  REFRESH: it_fieldcat .
  CLEAR wa_meta .
  LOOP AT meta INTO wa_meta.
    is_fieldcat-fieldname = wa_meta-fieldname.
    is_fieldcat-outputlen = wa_meta-outputlen .
    is_fieldcat-datatype = wa_meta-datatype.
    is_fieldcat-scrtext_l = wa_meta-scrtext_l.
    APPEND is_fieldcat TO it_fieldcat.
    CLEAR : is_fieldcat .
  ENDLOOP .

  IF NOT it_fieldcat[] IS INITIAL .
    CALL METHOD cl_alv_table_create=>create_dynamic_table
      EXPORTING
        it_fieldcatalog = it_fieldcat
      IMPORTING
        ep_table        = new_table.
    ASSIGN new_table->* TO <outtab>.
    CREATE DATA new_line LIKE LINE OF <outtab>.
    ASSIGN new_line->* TO <l_line> .
  ENDIF .
  IF NOT xml_out IS INITIAL .
    TRY .
```

```
          CALL TRANSFORMATION (`Y_RRT_TRANSFORM`)
          SOURCE XML  xml_out
          RESULT     outtab = <outtab>.
        CATCH cx_xslt_exception INTO xslt_error.
          xslt_message = xslt_error->get_text( ).
          WRITE:/ xslt_message .
      ENDTRY.
    ENDIF .
    IF NOT <outtab>[] IS INITIAL .
      LOOP AT <outtab> ASSIGNING <l_line>.
        WRITE:/ <l_line> .
      ENDLOOP .
    ENDIF .
  ENDIF .
```

Sample Selection Screen Value:



Now run the program in debug mode and see how the dynamic results are coming as XML and how the XML is getting transformed in to ABAP internal table using XSLT program.

Author Bio

Durairaj Athavan Raja works as Business System Analyst with Atos Origin Middle East and has been involved in SAP development for over 8 years. He is a big fan of SDN.