Product Group Hierarchy in SAP - Business Intelligence

Applies to:

SAP BW 3.5 and SAP BI NW 7.0. For more information, visit the Business Intelligence homepage.

Summary

The purpose of this paper is to introduce the design for loading Product Group Hierarchy in BI. This paper is targeted at customers who are interested in implementing Product Group Hierarchy, so that we can display Product Group and their relevant materials in the report output, up to the leaf nodes.

Author: Dhanya A

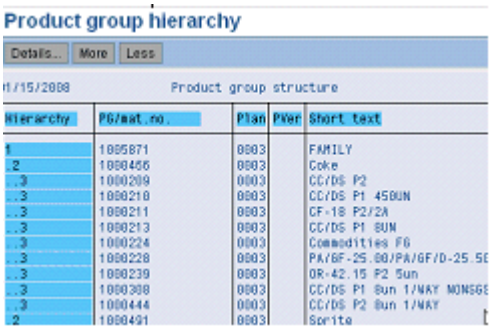
Company: Satyam Computers Services Ltd.

Created on: 3rd October 2008

Author Bio

Dhanya A is a SAP BI Consultant at Satyam Computer Services Limited, Hyderabad - India. She is working in BI space for past 4 years and has rich experience in Enterprise Data Warehousing, Enterprise Reporting and Performance Tuning for solutions to Retail, Utilities, Finance industry. She has experience in BW-BPS implementation of FI-AA and Microsoft Integration with SAP BI.

Table of Contents



Abstract

The purpose of this paper is to introduce the design for loading product Group Hierarchy in BI. This paper is targeted at customers who are interested in implementing Product Group Hierarchy so that we can display product Group and their relevant materials in the report output up to the leaf nodes. This paper is to explain the concept of building this in BI. There is no Business Content for this, hence it will save lot of time in development.

Firstly, develop an R/3 Generic DataSource based on Function module. The DataSource should be able to extract the Product group hierarchy based on the selections of Product Group and Plant. The hierarchy from R/3 will be loaded to a DSO through the Generic DataSource. The Open Hub Destination will retract the hierarchy data in the DSO to a flat file in application server. The Product Group hierarchy will be loaded to InfoObject 0MAT_PLANT through this flat file stored in Application Server.

Business Scenario

In BW, the client had Sales data available at Kit level (Material) through CO-PA extraction Actual and Plan DataSource. In these transactional records, there was no information indicating which materials belong to which product Group. For more information on product group see Appendix A.

The business wanted to have product group to be available in the BI targets so that they can select, filter, navigate and display product group and material relationship.

Challenges:

- There was no master data in BI providing the product group relationship with material.
- The option of getting the PGMI Table entries to BW and then writing ABAP Code in BI to populate Product Group Hierarchy was not a viable solution as there was no Function Module in BI which would generate the entire hierarchy link up to the last level.
- There was no option of creating Hierarchy DataSource through flat file in BI 7.0.

Solution Benefits:

The option of creating a DataSource in ECC system which will give the entire hierarchy data up to the leaf nodes was an efficient option, as we had standard FM in ECC system which would cater to this requirement. Further benefit was: we could incorporate the Hierarchy structure format (Node ID, Node Name, parent ID, Link ID) in the DataSource itself.

This minimized coding on the BI side too.

The ABAP code can be used as an reference, whenever we need to write ABAP Code for Custom DataSource to get hierarchy structure format.

Development Specification

Create a function module in ECC system which has the function module RSAX_GET_DATA_SIMPLE as the template.

Algorithm of the Function module:

- Selections to be included are Product Group and Plant. These selections need to be optional, which means that if no plant and/or product group is entered, it should execute for all plants and product groups. If plant/plants only are entered, then it should execute for all product groups under that plant/plants. If product group/ product groups only is entered, then it should execute for all plants under that product group.
- Based on selections given in optional fields in plant and/or Product group, call Function module MC_PG_STRUKTUR for that many combinations.
- Populate the NodeID and Parent ID fields.
- Also, place an indicator for the Childs which are under multiple parents for the same plant.
- Use the output table, which gets populated through the above function module to store it into an internal table.
- After all the records are extracted, place all the records into a database table YCPS_PRGP.
- Use fetch cursor mechanism to limit the packet size as per MAXSIZE.
- Also need to delete the entries in database table YCPS_PRGP before every extraction.
- The format of the internal table should be modified to be similar to DataSource Structure.

The ABAP source code for this FM is in Appendix B.
Create a new Generic DataSource of type Master data and assign it to application component PP. This Generic DataSource which will use the above function module to extract data.
The structure of the DataSource should be:
1. NodeID
2. Plant
3. Product Group
4. PlantProductGroup
5. LinkID
6. ParentID

The first process in BI is to replicate Generic DataSource in BW in application component PP.
Create a write optimized DSO which will have the Generic DataSource as the source.

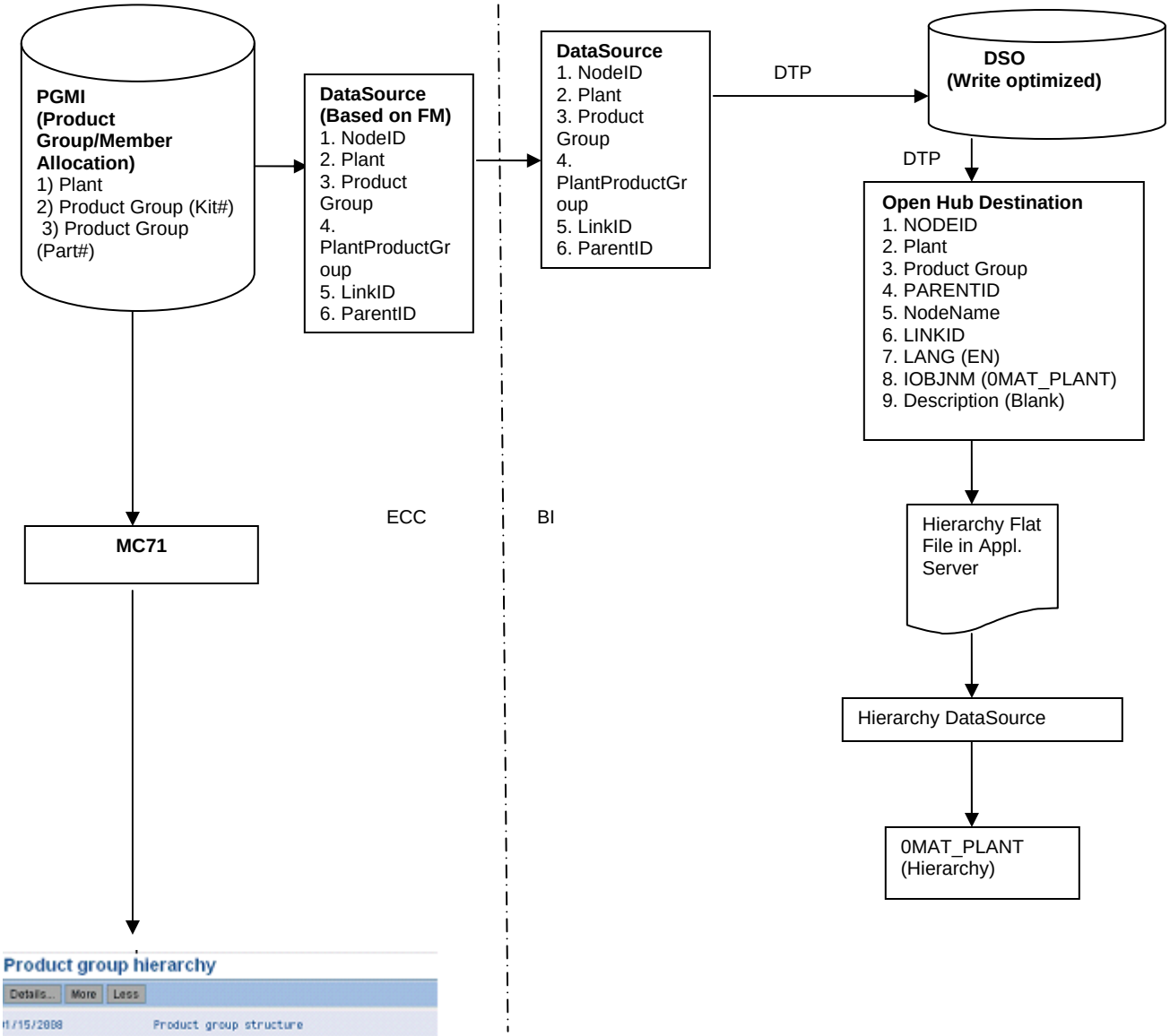
Note: The DSO is used in the dataflow because Open Hub Destination cannot fetch data from a DataSource.

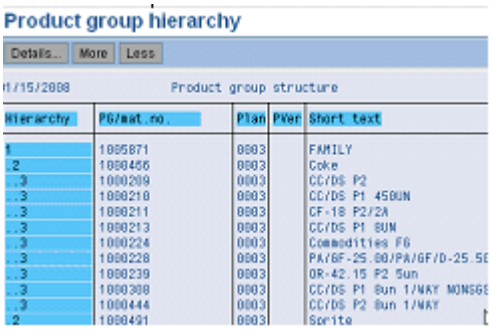
Create an Open Hub Destination which will have the DSO as the source.
The Open Hub Destination will retract the data from this DSO to a file in application server.

Note: Process of creating a logical path to store data in Appl. Server is in Appendix C.

Create a hierarchy DataSource in BW 3.5 with Flat file as source system.
The flat file stored in Appl. Server will be the source for the above hierarchy DataSource in BW.
This hierarchy data source will load the hierarchy to InfoObject 0MAT_PLANT.

Design Overview





Appendix A

Product groups are required to reflect the supply relationships between the subsidiaries in our consolidated group. A product group groups together products (materials). The criteria by which this grouping takes place can be defined individually by each user.

For example, the products may be similar to each other in some way, or they may be finished products that were produced on the same machine.

A product group can be multi-level or single-level. A product group is multi-level if its members are other product groups. However, the lowest product group in the hierarchy must contain materials.

The criteria to define about the product group will be decided by the business like for example product group Soft Drinks. In this one can include Pepsi, Cola and Thums Up etc and again within Thums Up you can divide 100ml, 250ml and 500ml as products. BOM and other master data will not be the criteria to define products group.

Quantities will be decided at product group header level...for example sales dept has decided to sell 100 pieces of soft drinks in east region of India and then they will maintain out of 100 pieces, 50% will be cola, 20% will be Thums up and 30% will be Pepsi.
Again 50% of cola product group will be 100% from this 40% will be 100ml and like wise.

Appendix B

```
FUNCTION YCPS_PRDGRP_FUNC.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_REQUNR) TYPE  SRSC_S_IF_SIMPLE-REQUNR
*"     REFERENCE(I_DSOURCE) TYPE  SRSC_S_IF_SIMPLE-DSOURCE OPTIONAL
*"     REFERENCE(I_MAXSIZE) TYPE  SRSC_S_IF_SIMPLE-MAXSIZE OPTIONAL
*"     REFERENCE(I_INITFLAG) TYPE  SRSC_S_IF_SIMPLE-INITFLAG OPTIONAL
*"     REFERENCE(I_READ_ONLY) TYPE  SRSC_S_IF_SIMPLE-READONLY OPTIONAL
*"  TABLES
*"      I_T_SELECT TYPE  SRSC_S_IF_SIMPLE-T_SELECT OPTIONAL
*"      I_T_FIELDS TYPE  SRSC_S_IF_SIMPLE-T_FIELDS OPTIONAL
*"      E_T_DATA STRUCTURE  YCPS_PRD_STR OPTIONAL
*"  EXCEPTIONS
*"      NO_MORE_DATA
*"      ERROR_PASSED_TO_MESS_HANDLER
*"----------------------------------------------------------------------
* Auxiliary Selection criteria structure
  DATA: L_S_SELECT TYPE SRSC_S_SELECT,
           LT_DATE    LIKE STKO-DATUV,
           L_TABIX    LIKE SY-TABIX.
  DATA: ICPS_PRD TYPE STANDARD TABLE OF YCPS_PRD_STR,
           ICPS_TMP TYPE TABLE OF YCPS_PRD_STR.

TABLES: YCPS_PRGP.

* Maximum number of lines for DB table
  STATICS: S_S_IF TYPE SRSC_S_IF_SIMPLE,
          S_COUNTER_DATAPAKID LIKE SY-TABIX,
          S_CURSOR TYPE CURSOR.
* 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 DataSource validity
   CASE I_DSOURCE.
     WHEN 'YCPS_PRDGRP_HIER'.
     WHEN OTHERS.
       IF 1 = 2. MESSAGE E009(R3). ENDIF.
* This is a typical log call. Please write every error message like this
       LOG_WRITE 'E'               "message type
                 'R3'              "message class
                 '009'             "message number
                 I_DSOURCE         "message variable 1
                 ' '.              "message variable 2
       RAISE ERROR_PASSED_TO_MESS_HANDLER.
   ENDCASE.

   APPEND LINES OF I_T_SELECT TO S_S_IF-T_SELECT.
* Fill 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.
 ELSE.               "Initialization mode or data extraction?
************************************************************************
* Data transfer: First Call      OPEN CURSOR + FETCH
*               Following Calls FETCH only
************************************************************************
* First data package -> OPEN CURSOR
   IF S_COUNTER_DATAPAKID = 0.
     CLEAR   :    R_PRGRP,
                  R_WERKS.
     REFRESH:   R_PRGRP,
                  R_WERKS.
*-- Product group
     LOOP AT S_S_IF-T_SELECT INTO L_S_SELECT WHERE FIELDNM = 'PRGRP'.
       MOVE-CORRESPONDING L_S_SELECT TO R_PRGRP.
       APPEND R_PRGRP.
     ENDLOOP.
*-- Plant
     LOOP AT S_S_IF-T_SELECT INTO L_S_SELECT WHERE FIELDNM = 'WERKS'.
       MOVE-CORRESPONDING L_S_SELECT TO R_WERKS.
       APPEND R_WERKS.
     ENDLOOP.
  IF R_PRGRP[] IS INITIAL.
    IF R_WERKS[] IS INITIAL.
* If both Product Group and Plant are NOT selected.
* Collect all parent nodes.
     SELECT DISTINCT PRGRP
                  WERKS
         FROM   PGMI
         INTO TABLE I_FORKIT.
     IF SY-SUBRC = 0.
       SORT I_FORKIT BY PRGRP WERKS.
     ENDIF.
* Collect all child nodes.
```

```
      SELECT DISTINCT  NRMIT
                    WEMIT
      FROM PGMI
      INTO TABLE I_FORPART.
      IF SY-SUBRC = 0.
        SORT I_FORPART BY NRMIT WEMIT.
      ENDIF.
      PERFORM COLLECT_PRGRP.
    ELSE.
* If Product Group is NOT selected.
* Collect all parent nodes.
      SELECT DISTINCT PRGRP
                    WERKS
          FROM   PGMI
          INTO TABLE I_FORKIT
          WHERE WERKS IN R_WERKS.
      IF SY-SUBRC = 0.
        SORT I_FORKIT BY PRGRP WERKS.
      ENDIF.
* Collect all child nodes.
      SELECT DISTINCT  NRMIT
                    WEMIT
      FROM PGMI
      INTO TABLE I_FORPART
      WHERE WEMIT IN R_WERKS.
      IF SY-SUBRC = 0.
        SORT I_FORPART BY NRMIT WEMIT.
      ENDIF.
      PERFORM COLLECT_PRGRP.
    ENDIF.

  ELSEIF R_WERKS[] IS INITIAL.
* If Plant is NOT selected.
* Collect all parent nodes.
    SELECT DISTINCT PRGRP
                  WERKS
        FROM   PGMI
        INTO TABLE I_FORKIT
        WHERE PRGRP IN R_PRGRP.
    IF SY-SUBRC = 0.
      SORT I_FORKIT BY PRGRP WERKS.
    ENDIF.
* Collect all child nodes.
    SELECT DISTINCT  NRMIT
                  WEMIT
    FROM PGMI
    INTO TABLE I_FORPART
    WHERE NRMIT IN R_PRGRP.
    IF SY-SUBRC = 0.
      SORT I_FORPART BY NRMIT WEMIT.
    ENDIF.
    PERFORM COLLECT_PRGRP.

  ENDIF.
* If both Product Group and Plant are selected.
  IF NOT R_PRGRP[] IS INITIAL.
    IF NOT R_WERKS[] IS INITIAL.
* Collect all parent nodes.
```

```
      SELECT DISTINCT PRGRP
                      WERKS
          FROM   PGMI
          INTO TABLE I_FORKIT
          WHERE PRGRP IN R_PRGRP AND
                WERKS IN R_WERKS.
      IF SY-SUBRC = 0.
        SORT I_FORKIT BY PRGRP WERKS.
      ENDIF.
* Collect all child nodes.
      SELECT DISTINCT  NRMIT
                       WEMIT
      FROM PGMI
      INTO TABLE I_FORPART
      WHERE NRMIT IN R_PRGRP AND
          WEMIT IN R_WERKS.
      IF SY-SUBRC = 0.
        SORT I_FORPART BY NRMIT WEMIT.
      ENDIF.
      PERFORM COLLECT_PRGRP.
      IF I_PRGPLA[] IS INITIAL.
        PERFORM COLLECT_PRGRP1.
      ENDIF.
    ENDIF.
  ENDIF.

SELECT * FROM YCPS_PRGP INTO TABLE ICPS_TMP.
DELETE YCPS_PRGP FROM TABLE ICPS_TMP.
IF SY-SUBRC = 0.
     COMMIT WORK.
ENDIF.
CLEAR ICPS_TMP.

*********************************************************************
  IF NOT I_PRGPLA[] IS INITIAL.
    LOOP AT I_PRGPLA INTO W_PRGPLA.

      CALL FUNCTION 'MC_PG_STRUKTUR'
        EXPORTING
          IDATUB                    = SY-DATUM
          IDATUV                    = SY-DATUM
          IPRGRP                     = W_PRGPLA-PRGRP
          IWERKS                     = W_PRGPLA-WERKS
        TABLES
          IPGTAB                    = I_PGTAB
        EXCEPTIONS
          PG_OR_MATERIAL_NOT_FOUND     = 1
          UNIT_CONVERSION_NOT_POSSIBLE = 2
          OTHERS                        = 3.

      IF SY-SUBRC <> 0.
        MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
              WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
      ENDIF.

      IF NOT I_PGTAB[] IS INITIAL.
* Move required fields of PGTAB data into another internal table
        LOOP AT I_PGTAB INTO W_PGTAB.
```

```
        W_PRD-PGHST = W_PGTAB-PGHST.
        W_PRD-PRGRP = W_PGTAB-PRGRP.
        W_PRD-WERKS = W_PGTAB-WERKS.
        APPEND W_PRD TO I_PRD.
      ENDLOOP.
* To find Max node value
      I_PGTMP[] = I_PGTAB[].
      SORT I_PGTMP DESCENDING BY PGHST.
      READ TABLE I_PGTMP INDEX 1 INTO W_PGTMP.
      IF SY-SUBRC = 0.
        V_MAX = W_PGTMP-PGHST.
* Hold the product groups that has child nodes into separate internal table
      LOOP AT I_PRD INTO W_PRD.
        W_TMP-PGHST = W_PRD-PGHST.
        W_TMP-PRGRP = W_PRD-PRGRP.
        W_TMP-WERKS = W_PRD-WERKS.
        AT NEW PGHST.
          IF W_PRD-PGHST < V_MAX.
            APPEND W_TMP TO I_TMP.
          ELSE.
            CONTINUE.
          ENDIF.
        ENDAT.
      ENDLOOP.

      LOOP AT I_PGTAB INTO W_PGTAB.
* Update parent nodes table with node number.
        CLEAR W_ICPS_PRD.
        CLEAR W_PARENTS.
        READ TABLE I_TMP INTO W_TMP WITH KEY PRGRP = W_PGTAB-PRGRP.
        IF SY-SUBRC = 0.
          IF W_TMP-PRGRP EQ W_PRGPLA-PRGRP.          "ROOT
            W_ICPS_PRD-P_NODE = 0.
            V_ROOT = V_NODE.
          ELSE.                                      "PARENT
            W_ICPS_PRD-P_NODE = V_ROOT.
            V_PARNT = V_NODE.
             IF NOT I_PARNT[] IS INITIAL.
              CONCATENATE W_PGTAB-WERKS W_PGTAB-PRGRP INTO V_PPGRP.
              READ TABLE I_PARNT INTO W_PARNT WITH KEY PPGRP = V_PPGRP.
              IF SY-SUBRC = 0.
                W_ICPS_PRD-LINK = 'X'.
              ENDIF.
             ENDIF.
          ENDIF.
          W_ICPS_PRD-NODE_ID = V_NODE.
          W_ICPS_PRD-WERKS = W_PGTAB-WERKS.
          W_ICPS_PRD-PRGRP = W_PGTAB-PRGRP.
          CONCATENATE W_PGTAB-WERKS W_PGTAB-PRGRP INTO W_ICPS_PRD-PPRGP.
          APPEND W_ICPS_PRD TO ICPS_PRD.
          CLEAR: W_ICPS_PRD-LINK, V_PPGRP.
          V_TABIX = SY-TABIX.
          V_NODE = V_NODE + 1.
        ELSE. " read i_tmp
          IF I_PARNT[] IS INITIAL.                "LEAFS
            DESCRIBE TABLE I_TMP LINES G_T.
            IF G_T EQ 1.
              W_ICPS_PRD-P_NODE = V_ROOT.
```

```
            ELSEIF NOT V_PARNT IS INITIAL.
              W_ICPS_PRD-P_NODE = V_PARNT.
            ELSE.
              W_ICPS_PRD-P_NODE = V_ROOT.
            ENDIF.
            W_ICPS_PRD-NODE_ID = V_NODE.
            W_ICPS_PRD-WERKS = W_PGTAB-WERKS.
            W_ICPS_PRD-PRGRP = W_PGTAB-PRGRP.
            CONCATENATE W_PGTAB-WERKS W_PGTAB-PRGRP INTO W_ICPS_PRD-PPRGP.
            APPEND W_ICPS_PRD TO ICPS_PRD.
            V_NODE = V_NODE + 1.
          ELSE.
            CLEAR V_PPGRP.
            CONCATENATE W_TMP-WERKS W_TMP-PRGRP INTO V_PPGRP.
            READ TABLE I_PARNT INTO W_PARNT WITH KEY PPGRP = V_PPGRP.
            IF SY-SUBRC = 0.
              CONTINUE.
            ELSE.
              DESCRIBE TABLE I_TMP LINES G_T.
              IF G_T EQ 1.
                W_ICPS_PRD-P_NODE = V_ROOT.
              ELSEIF NOT V_PARNT IS INITIAL.
                W_ICPS_PRD-P_NODE = V_PARNT.
              ELSE.
                W_ICPS_PRD-P_NODE = V_ROOT.
              ENDIF.
              W_ICPS_PRD-NODE_ID = V_NODE.
              W_ICPS_PRD-WERKS = W_PGTAB-WERKS.
              W_ICPS_PRD-PRGRP = W_PGTAB-PRGRP.
              CONCATENATE W_PGTAB-WERKS W_PGTAB-PRGRP INTO W_ICPS_PRD-PPRGP.
              APPEND W_ICPS_PRD TO ICPS_PRD.
              V_NODE = V_NODE + 1.
            ENDIF.
          ENDIF.
        ENDIF. " read i_tmp
      ENDLOOP.

    ENDIF. " READ TABLE I_PGTMP
  ENDIF.   " IF NOT I_PGTAB[]

  LOOP AT I_TMP INTO W_TMP.
    W_PARNT-ROOT = W_PRGPLA-PRGRP.
    W_PARNT-PRGRP = W_TMP-PRGRP.
    W_PARNT-WERKS = W_TMP-WERKS.
    CONCATENATE W_TMP-WERKS W_TMP-PRGRP INTO W_PARNT-PPGRP.
    APPEND W_PARNT TO I_PARNT.
    CLEAR: W_PARNT, W_TMP.
  ENDLOOP.

  CLEAR: W_ICPS_PRD, W_PARENTS, I_TMP[], I_PRD[], I_PGTAB[].
ENDLOOP. "    LOOP AT i_prgpla
CLEAR: I_PRGPLA[].

MODIFY YCPS_PRGP FROM TABLE ICPS_PRD.
IF SY-SUBRC = 0.
  COMMIT WORK.
  ENDIF.
```

```
  ENDIF.  "IF NOT i_prgpla[]

    IF  ICPS_PRD IS INITIAL.
      RAISE NO_MORE_DATA.
    ENDIF.

    OPEN CURSOR WITH HOLD S_CURSOR FOR
         SELECT * FROM YCPS_PRGP.
  ENDIF.   "First data package ?

    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.

    S_COUNTER_DATAPAKID = S_COUNTER_DATAPAKID + 1.
  ENDIF.
 REFRESH ICPS_PRD.
ENDFUNCTION.
```

The below code needs to be written in the function group for the perform statements stated above.

```
*&---------------------------------------------------------------------*
*&      FORM COLLECT_PRGRP
*&---------------------------------------------------------------------*
*       TEXT
*----------------------------------------------------------------------*
* --> P1        TEXT
* <-- P2        TEXT
*----------------------------------------------------------------------*
FORM COLLECT_PRGRP.
*PREPARE FINAL INTERNAL TABLE WITH REQUIRED PRODUCTGROUPS
*AND PLANT COMBINATION.
  LOOP AT I_FORKIT INTO W_FORKIT.
    READ TABLE I_FORPART INTO W_FORPART WITH KEY NRMIT = W_FORKIT-PRGRP
                                        WEMIT = W_FORKIT-WERKS
                                        BINARY SEARCH.
    IF SY-SUBRC <> 0.
      W_PRGPLA-PRGRP = W_FORKIT-PRGRP.
      W_PRGPLA-WERKS = W_FORKIT-WERKS.
      APPEND W_PRGPLA TO I_PRGPLA.
    ENDIF.
  ENDLOOP.
  SORT I_PRGPLA BY WERKS.
  DELETE ADJACENT DUPLICATES FROM I_FORKIT COMPARING PRGRP WERKS.
ENDFORM.                    "COLLECT_PRGRP
*&---------------------------------------------------------------------*
*&      FORM COLLECT_PRGRP1
*&---------------------------------------------------------------------*
*       TEXT
*----------------------------------------------------------------------*
* --> P1        TEXT
* <-- P2        TEXT
*----------------------------------------------------------------------*
FORM COLLECT_PRGRP1.
```

```
*PREPARE FINAL INTERNAL TABLE WITH REQUIRED PRODUCTGROUPS
*AND PLANT COMBINATION.
  LOOP AT I_FORKIT INTO W_FORKIT.
    READ TABLE I_FORPART INTO W_FORPART WITH KEY NRMIT = W_FORKIT-PRGRP
                                        WEMIT = W_FORKIT-WERKS
                                        BINARY SEARCH.
    IF SY-SUBRC = 0.
      W_PRGPLA-PRGRP = W_FORKIT-PRGRP.
      W_PRGPLA-WERKS = W_FORKIT-WERKS.
      APPEND W_PRGPLA TO I_PRGPLA.
    ENDIF.
  ENDLOOP.
  SORT  I_PRGPLA BY PRGRP WERKS.
  DELETE ADJACENT DUPLICATES FROM I_PRGPLA COMPARING PRGRP WERKS.
ENDFORM.                    "COLLECT_PRGRP1
```

Appendix C

Create a logical file path in T-Code FILE with following details:

Logical file path: **YCPS_PRDGP_EXTRACT_FILES**

Logical file: **YCPS_PRDGP**

Create a .CSV file in T-Code AL11 in the path: /usr/sap/interfaces/XXX/inbound/.

Assign the logical file path to the select the CSV file created above.

Related Content

http://help.sap.com/saphelp_46c/helpdata/en/a5/631b1243a211d189410000e829fbbd/frameset.htm

Loading Hierarchies

http://help.sap.com/saphelp_nw04/helpdata/en/80/1a6729e07211d2acb80000e829fbfe/content.htm

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.