Using the PowerExchange CallProg Function to Call a User Exit Program

Abstract

This article describes how to use the PowerExchange CallProg function in an expression in a data map record to call a user exit program that returns the class type of a specified field in the data map record.

Supported Versions

- PowerExchange 10.0 or later

Table of Contents

Overview

In this example, you add user-defined fields to a data map record to invoke the PowerExchange CallProg function to call a user exit program.

The user exit program returns the class type for the data in a specified field. The user exit program tests any field with a maximum length of 15 bytes.

The user exit program returns one of the following class types:

Class Type	Description
A	Alphabetic
H	High values
L	Low values
N	Numeric zoned decimal
S	Spaces

This example shows how to complete the following tasks:

1. Add a data map by using a sample data file and copybook that ship with PowerExchange on z/OS.
2. Create and compile a user exit program. Save the DLL in the PowerExchange Listener LOADLIB library on z/OS.

3. Add the following user-defined fields to the MASTER_REC data map record:
    - The classtype_bin_no, classtype_dec_no, and classtype_rec_type fields.

      Defined as a one-byte CHAR fields. The user exit program that is called by the CallProg function returns the class type of a specified field in these fields. You must define a separate classtype field for each field for which you want to check the class type.
    - The rc_bin_no, rc_decimal_no, and rc_rec_type fields.

      Defined as NUM32 fields. Use these fields to call the CallProg function and to contain the return code from the user exit program call.

   Before PowerExchange completes data checking for a data map record, it runs any expressions and program calls defined in user-defined fields in the data map record.
4. Refresh the columns in the MASTER_REC table to pick up the user-defined fields that you added to the MASTER_REC record.
5. Run a database row test on the data map record to test the results of the user exit program and to verify that the user exit program runs correctly.

For more information about using PowerExchange functions in user-defined fields to process data in data map records, see the *PowerExchange Navigator User Guide*.

Step 1. Add a Data Map

In this step, you add a data map for a sequential flat file and import a COBOL copybook.

To add the data map, use the following data set members that ship with PowerExchange on the z/OS system:

Data Set Member	Description
KSDSDAT	Data file that contains source data.
KSDSCOB	COBOL copybook that you import to define the layout of the data.

1. On the **Resources** tab in the **Resource Explorer**, click **Add** > **Data Map**.
2. In the **Name** dialog box, define the following properties for the data map:

Property	Value
Schema Name	demo
Data Map Name	userexit
Access Method	SEQ

   Also, select the **Import Record Definitions** option.
3. Click **Next**.

4. In the **SEQ Access Method** dialog box, define the following properties for the data map:

Property	Value	Notes
File Name	*PWX_installation_dataset*.DTLDEMO(KSDSDAT)	Where *PWX_installation_dataset* is the PowerExchange installation data set. For example, you might enter the following value for the file name: `PWX.V901.DTLDEMO(KSDSDAT)`
Record Format	Default	
Skip First	0	

   Also, verify that the **File List Processing** option is disabled. By default, this option is disabled.

5. Click **Finish**.
6. In the **Import Copybook - Source Details** dialog box, define the following properties for the copybook:

Property	Value
Source	Remote
Type	COBOL
Column Range Start	7
Column Range End	72

7. Click **Next**.
8. In the **Import Copybook - Remote Cobol Details** dialog box, define the following properties for the copybook:

Property	Value	Notes
File Name	*PWX_installation_dataset*.DTLDEMO(KSDSCOB)	Where *PWX_installation_dataset* is the PowerExchange installation data set. For example, you might enter the following value for the file name: `PWX.V901.DTLDEMO(KSDSCOB)`
Location	Node for the z/OS system	
UserID	The user ID for the z/OS system	
Password	The password for the user ID on the z/OS system	
Save File Locally As	A name for the copybook file on the local Windows system	

   Note: If you want to preview the copybook, click **Preview**. After you preview the copybook, close the preview window.

9. Click **Next**.

10. In the **Import Copybook - Configuration Details** dialog box, review the selected actions for imported records, fields, and tables, and click **Finish**.
11. In the **Import Copybook Information** window, review the information for the import and click **OK**.
12. In the **Record Definition** dialog box for the MASTER_REC record, click **OK**.

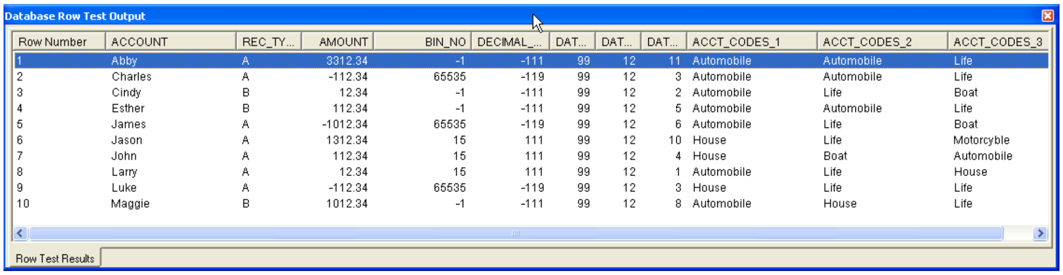
    The **Copybook Redefines** message box appears indicating that two definitions for the BIN_NO field exist in the copybook. For this example, the first definition is the correct one.
13. To accept the first definition for the BIN_NO field, click **Import** > **Goto Current Line**. Then, click **Import** > **Resume** to resume the import.

    The **Cobol Import** window displays the imported copybook and the **Copybook Message Log** window displays the results of the import operation.
14. Close the **Cobol Import** window.
15. To verify that the data map was added correctly, on the **Resources** tab in the **Resource Explorer**, select the MASTER_REC table and click **File** > **Database Row Test**.
16. When you are prompted to send the data map to a remote location, click **Yes**.

    The **Data Map Remote Node** dialog box appears.
17. In the **Data Map Remote Node** dialog box, enter the user ID, password, and the node for the z/OS system.

    The **Database Row Test** dialog box appears.
18. In the **Database Row Test** dialog box, accept the default values and click **Go**.

    The **Database Row Test Output** window displays the the following results for the database row test:



Database Row Test Output

Row Number	ACCOUNT	REC_TY...	AMOUNT	BIN_NO	DECIMAL_...	DAT...	DAT...	DAT...	ACCT_CODES_1	ACCT_CODES_2	ACCT_CODES_3
1	Abby	A	3312.34	-1	-111	99	12	11	Automobile	Automobile	Life
2	Charles	A	-112.34	65535	-119	99	12	3	Automobile	Automobile	Life
3	Cindy	B	12.34	-1	-111	99	12	2	Automobile	Life	Boat
4	Esther	B	112.34	-1	-111	99	12	5	Automobile	Automobile	Life
5	James	A	-1012.34	65535	-119	99	12	6	Automobile	Life	Boat
6	Jason	A	1312.34	15	111	99	12	10	House	Life	Motorcyble
7	John	A	112.34	15	111	99	12	4	House	Boat	Automobile
8	Larry	A	12.34	15	111	99	12	1	Automobile	Life	House
9	Luke	A	-112.34	65535	-119	99	12	3	House	Life	Life
10	Maggie	B	1012.34	-1	-111	99	12	8	Automobile	House	Life

Step 2. Create and Compile the User Exit Program

In this step, you copy the UCPEP user exit program and modify it. Then, compile the user exit program and save the DLL in the PowerExchange Listener LOADLIB library.

PowerExchange ships sample user exit programs, including the UCPEP program, in the SRCLIB library in the installation data set on z/OS.

1. In the SRCLIB library in the PowerExchange installation data set on z/OS, copy the UCPEP user exit program and name it UCPGCLSC.
2. Edit the UCPGCLSC program and make the changes marked in bold, as shown in the following code:

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. UCPGCLSC.
*****************************************************
* GLOBAL CUSTOMER SUPPORT SAMPLE CLASS TEST
* EXAMPLE COBOL PROGRAM CALLED VIA CALLPROG.
*
*****              *****
* USER EXITS ARE NOT SUPPORTED BY INFORMATICA
* USER EXITS ARE USED AT THE CUSTOMERS OWN RISK
*
```

```
*
* USING SYNTAX :-
* CALLPROG('UCPGCLSC','UCPGCLSC','COBOL','VOID',
*            TEXT_FIELD,NUMBER1_FIELD)
*
* RECEIVES THE FOLLOWING ARGUMENTS :-
*  1. NUMBER-ARGUMENTS - REQUIRED
*     THE NUMBER OF ARGUMENTS WHICH FOLLOW.
*     THE PROGRAM WILL EXIT SETTING A BAD RETURN CODE
*     IF THE NUMBER IS NOT WHAT IT EXPECTS.
*
*  2. FAILURE-CODE. - REQUIRED
*     AN INTEGER PASSED BACK TO THE CALLER TO INDICATE IF
*     PROCESSING WAS WAS NOT SUCCESSFUL.
*     THE FAILURE-CODE IS MONITORED
*     SO THAT ACTION CAN BE TAKEN TO HANDLE ERRORS.
*
*     BECAUSE THE MVS COBOL LINKAGE TYPE ONLY SUPPORTS A
*     RETURN TYPE OF 'VOID', IT IS NECESSARY TO PASS IT
*     BACK AS A NORMAL FIELD WITH ITS ACCOMPANYING LENGTH.
*     (SEE CLLPRGL2 FOR HOW THE RETURN CODE CAN BE PASSED
*     USING A LINKAGE TYPE OF 'OS' RETURNING 'INT')
*
*  3. MESSAGE-BUFFER. - REQUIRED
*     AN ERROR INTO WHICH THE PROGRAM CAN PUT A MESSAGE
*     TO ACCOMPANY A NON-ZERO FAILURE CODE, INDICATING
*     THE REASON.
*
*  4. MESSAGE-BUFFER-LENGTH. - REQUIRED
*     THE LENGTH OF MESSAGE-BUFFER
*
*  5. TEXT-AREA.
*     THIS IS THE 5TH ARGUMENT TO CALLPROG DEFINED IN THE
*     NAVIGATOR EXPRESSIONS SCREEN.
*     IN THIS EXAMPLE, IT IS A FIELD CONTAINING A MAX OF 15 BYTES
*
*  6. TEXT-AREA-LENGTH.
*     THE LENGTH OF TEXT-AREA WHICH VARIES ACCORDING
*     TO THE ACTUAL FIELD LENGTH ON THE FILE.
*
*  7. CLASS-TYPE.
*     THIS IS THE 6TH ARGUMENT TO CALLPROG DEFINED IN THE
*     NAVIGATOR EXPRESSIONS SCREEN.
*     IN THIS EXAMPLE, IT IS A 1 BYTE CHARACTER FIELD WITH VALUES
*     S=SPACES, L=LOW-VALUES, H=HIGH-VALUES, A=ALPHABETIC, N=NUMERIC
*
*  8. CLASS-TYPE-LENGTH.
*     THE LENGTH OF FIELD CLASS-TYPE WHICH WILL ALWAYS
*     BE 1.
****************************************************
*
 ENVIRONMENT DIVISION.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 01  WS-DATE      PIC X(6).
 01  WS-TIME      PIC X(8).
 01  WS-DATA.
     05  WS-DATA15   PIC X(15).
 01  WS-DATAL14 REDEFINES WS-DATA.
     05  WS-DATA14   PIC X(14).
     05  FILLER      PIC X(01).
 01  WS-DATAL13 REDEFINES WS-DATA.
     05  WS-DATA13   PIC X(13).
     05  FILLER      PIC X(02).
 01  WS-DATAL12 REDEFINES WS-DATA.
     05  WS-DATA12   PIC X(12).
     05  FILLER      PIC X(03).
 01  WS-DATAL11 REDEFINES WS-DATA.
     05  WS-DATA11   PIC X(11).
     05  FILLER      PIC X(04).
 01  WS-DATAL10 REDEFINES WS-DATA.
     05  WS-DATA10   PIC X(10).
     05  FILLER      PIC X(05).
 01  WS-DATAL09 REDEFINES WS-DATA.
     05  WS-DATA09   PIC X(09).
     05  FILLER      PIC X(06).
 01  WS-DATAL08 REDEFINES WS-DATA.
     05  WS-DATA08   PIC X(08).
     05  FILLER      PIC X(07).
 01  WS-DATAL07 REDEFINES WS-DATA.
     05  WS-DATA07   PIC X(07).
     05  FILLER      PIC X(08).
 01  WS-DATAL06 REDEFINES WS-DATA.
     05  WS-DATA06   PIC X(06).
     05  FILLER      PIC X(09).
 01  WS-DATAL05 REDEFINES WS-DATA.
     05  WS-DATA05   PIC X(05).
     05  FILLER      PIC X(10).
 01  WS-DATAL04 REDEFINES WS-DATA.
     05  WS-DATA04   PIC X(04).
     05  FILLER      PIC X(11).
 01  WS-DATAL03 REDEFINES WS-DATA.
     05  WS-DATA03   PIC X(03).
     05  FILLER      PIC X(12).
 01  WS-DATAL02 REDEFINES WS-DATA.
     05  WS-DATA02   PIC X(02).
     05  FILLER      PIC X(13).
 01  WS-DATAL01 REDEFINES WS-DATA.
     05  WS-DATA01   PIC X(01).
     05  FILLER      PIC X(14).
*
 LINKAGE SECTION.

 01  LK-NUMBER-ARGUMENTS        PIC S9(9) COMP.

 01  LK-FAILURE-CODE            PIC S9(9) COMP.

 01  LK-MESSAGE-BUFFER.
     05  LK-MESSAGE-BUFFER-BYTE PIC X(1)
                 OCCURS 1 TO 255
                 DEPENDING ON LK-MESSAGE-BUFFER-LENGTH.
 01  LK-MESSAGE-BUFFER-LENGTH   PIC S9(9) COMP.

 01  LK-TEXT-AREA.
     05  LK-TEXT-AREA-BYTE      PIC X(1) OCCURS 15.
 01  LK-TEXT-AREA-LENGTH        PIC S9(9) COMP.

 01  LK-CLASS                   PIC X.
 01  LK-CLASS-LENGTH            PIC S9(9) COMP.

 PROCEDURE DIVISION USING
     LK-NUMBER-ARGUMENTS
     LK-FAILURE-CODE
     LK-MESSAGE-BUFFER
     LK-MESSAGE-BUFFER-LENGTH
     LK-TEXT-AREA
     LK-TEXT-AREA-LENGTH
     LK-CLASS
     LK-CLASS-LENGTH
     .

 MAIN SECTION.
 0100-MAIN.

     MOVE ZERO TO LK-FAILURE-CODE.
     MOVE ZERO TO LK-MESSAGE-BUFFER-LENGTH.
```

```
**************************************************************
* EXIT FLAGGING AN ERROR IF THE WRONG NUMBER OF ARGUMENT PAIRS
**************************************************************
     IF LK-NUMBER-ARGUMENTS NOT = 2
        DISPLAY 'UCPGCLSC:NUMBER-ARGUMENTS=' LK-NUMBER-ARGUMENTS
                  ' (REQUIRED 2)'
                  ' EXITTING WITH RC=401'
        MOVE 401 TO LK-FAILURE-CODE
        MOVE 'UCPGCLSC:NOT ENOUGH ARGUMENTS '
                            TO LK-MESSAGE-BUFFER
        GO TO 0900-MAIN-EXIT
     END-IF.

***********************************************************************
* IF DATA PRESENT FIND LENGTH AND TEST CLASS
***********************************************************************
*
     IF  LK-TEXT-AREA-LENGTH = ZERO
         MOVE 'FIELD EMPTY' TO LK-MESSAGE-BUFFER
         MOVE 11   TO LK-MESSAGE-BUFFER-LENGTH
         MOVE 402  TO LK-FAILURE-CODE
         GO TO 0900-MAIN-EXIT
       ELSE
          IF  LK-TEXT-AREA-LENGTH > +15
              MOVE 'LENGTH > 15' TO LK-MESSAGE-BUFFER
              MOVE 11   TO LK-MESSAGE-BUFFER-LENGTH
              MOVE 403  TO LK-FAILURE-CODE
              GO TO 0900-MAIN-EXIT
            ELSE
              MOVE LK-TEXT-AREA TO WS-DATA
     END-IF.
*
     MOVE SPACES TO LK-CLASS.
     MOVE +1     TO LK-CLASS-LENGTH.
*
 0100-CLASS15.
*
     IF  LK-TEXT-AREA-LENGTH < +15
         GO TO 0100-CLASS14
     END-IF.
*
     IF  WS-DATA ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS14.
```

```
*
     IF  LK-TEXT-AREA-LENGTH < +14
         GO TO 0100-CLASS13
     END-IF.
*
     IF  WS-DATA14 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA14 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA14 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA14 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA14 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS13.
*
     IF  LK-TEXT-AREA-LENGTH < +13
         GO TO 0100-CLASS12
     END-IF.
*
     IF  WS-DATA13 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA13 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA13 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA13 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA13 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS12.
*
     IF  LK-TEXT-AREA-LENGTH < +12
         GO TO 0100-CLASS11
     END-IF.
*
```

```
     IF  WS-DATA12 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA12 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA12 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA12 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA12 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS11.
*
     IF  LK-TEXT-AREA-LENGTH < +11
         GO TO 0100-CLASS10
     END-IF.
*
     IF  WS-DATA11 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA11 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA11 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA11 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA11 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS10.
*
     IF  LK-TEXT-AREA-LENGTH < +10
         GO TO 0100-CLASS09
     END-IF.
*
     IF  WS-DATA10 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
```

```
     IF  WS-DATA10 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA10 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA10 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA10 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS09.
*
     IF  LK-TEXT-AREA-LENGTH < +9
         GO TO 0100-CLASS08
     END-IF.
*
     IF  WS-DATA09 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA09 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA09 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA09 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA09 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS08.
*
     IF  LK-TEXT-AREA-LENGTH < +8
         GO TO 0100-CLASS07
     END-IF.
*
     IF  WS-DATA08 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA08 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
```

```
     IF  WS-DATA08 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA08 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA08 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS07.
*
     IF  LK-TEXT-AREA-LENGTH < +7
         GO TO 0100-CLASS06
     END-IF.
*
     IF  WS-DATA07 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA07 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA07 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA07 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA07 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS06.
*
     IF  LK-TEXT-AREA-LENGTH < +6
         GO TO 0100-CLASS05
     END-IF.
*
     IF  WS-DATA06 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA06 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA06 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
```

```
     IF  WS-DATA06 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA06 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS05.
*
     IF  LK-TEXT-AREA-LENGTH < +5
         GO TO 0100-CLASS04
     END-IF.
*
     IF  WS-DATA05 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA05 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA05 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA05 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA05 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS04.
*
     IF  LK-TEXT-AREA-LENGTH < +4
         GO TO 0100-CLASS03
     END-IF.
*
     IF  WS-DATA04 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA04 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA04 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA04 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
```

```
     IF  WS-DATA04 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS03.
*
     IF  LK-TEXT-AREA-LENGTH < +3
         GO TO 0100-CLASS02
     END-IF.
*
     IF  WS-DATA03 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA03 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA03 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA03 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA03 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
 0100-CLASS02.
*
     IF  LK-TEXT-AREA-LENGTH < +2
         GO TO 0100-CLASS01
     END-IF.
*
     IF  WS-DATA02 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA02 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA02 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA02 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA02 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
```

```
*
 0100-CLASS01.
*
     IF  LK-TEXT-AREA-LENGTH < +1
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA01 ALPHABETIC
         MOVE 'A' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA01 NUMERIC
         MOVE 'N' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA01 = LOW-VALUES
         MOVE 'L' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA01 = HIGH-VALUES
         MOVE 'H' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
*
     IF  WS-DATA01 = SPACES
         MOVE 'S' TO LK-CLASS
         GO TO 0900-MAIN-EXIT
     END-IF.
     GO TO 0900-MAIN-EXIT.
*
*
 0900-MAIN-EXIT.
     GOBACK.
```

3. Compile the UCPGCLSC program and save the DLL in the PowerExchange Listener LOADLIB library.

Step 3. Add User-Defined Fields

In this step, you add user-defined fields.

The user-defined fields invoke the PowerExchange CallProg function, which calls the user exit program. For more information about the CallProg function, see “CallProg” on page 20.
The user exit program processes data in and returns the class types of the REC_TYPE, BIN_NO, and DECIMAL_NO fields.

1. Open the demo.userexit data map and the MASTER_REC record.
2. In the **Record** window, click the **Expr(0)** tab.
3. Right-click anywhere on the **Expr(0)** tab and click **Add Field at End**.
4. Add the classtype_rec_type field, which is an output field that contains the result from the user exit program when it is invoked for the REC_TYPE field. Define the following properties for the field:

Property	Value
Name	classtype_rec_type
Type	CHAR

Property	Value
Precision	0
Scale	0
Length	1

5. Right-click anywhere on the **Expr(0)** tab and click **Add Field at End**.
6. Add the rc_rec_type field, which calls the user exit program to process the REC_TYPE field. Define the following properties for the field:

Property	Value	Notes
Name	rc_rec_type	
Type	NUM32	
Precision	0	
Scale	0	
Length	0	
Phase	RW	Indicates that the operation is read or write.
Expression	`CallProg('UCPGCLSC', 'UCPGCLSC','COBOL', REC_TYPE,classtype_rec_type)`	To enter the expression for the field, complete the following steps: 1. Click in the cell in the **Expression** column and click the Browse button. The **Expression Editor** dialog box appears. 2. In the **Function List** list in the **Expression Editor** dialog box, double-click the CallProg function. 3. In the **Expression List** list, enter `('UCPGCLSC','UCPGCLSC','COBOL' ,REC_TYPE,classtype_rec_type)` at the end of the CallProg function name. 4. Click **Validate**. In the **Validate** box, the `No Errors` message appears. 5. Click **OK**.

7. In the **Record** window, click the **Expr(0)** tab.
8. Right-click anywhere on the **Expr(0)** tab and click **Add Field at End**.

9. Add the classtype_bin_no field, which is an output field that contains the result from the user exit program when it is invoked for the BIN_NO field. Define the following properties for the field:

Property	Value
Name	classtype_bin_no
Type	CHAR
Precision	0
Scale	0
Length	1

10. Right-click anywhere on the **Expr(0)** tab and click **Add Field at End**.
11. Add the rc_bin_no field, which calls the user exit program to process the BIN_NO field. Define the following properties for the field:

Property	Value	Notes
Name	rc_bin_no	
Type	NUM32	
Precision	0	
Scale	0	
Length	0	
Phase	RW	Indicates that the operation is read or write.
Expression	`CallProg('UCPGCLSC', 'UCPGCLSC','COBOL', BIN_NO,classtype_bin_no)`	To enter the expression for the field, complete the following steps: 1. Click in the cell in the **Expression** column and click the Browse button. The **Expression Editor** dialog box appears. 2. In the **Function List** list in the **Expression Editor** dialog box, double-click the CallProg function. 3. In the **Expression List** list, enter `('UCPGCLSC','UCPGCLSC','COBOL',BIN_NO,classtype_bin_no)` at the end of the CallProg function name. 4. Click **Validate**. In the **Validate** box, the `No Errors` message appears. 5. Click **OK**.

12. Right-click anywhere on the **Expr(0)** tab and click **Add Field at End**.

13. Add the classtype_dec_no field, which is an output field that contains the result from the user exit program when it is invoked for the DECIMAL_NO field. Define the following properties for the field:

Property	Value
Name	classtype_dec_no
Type	CHAR
Precision	0
Scale	0
Length	1

14. Right-click anywhere on the **Expr(0)** tab and click **Add Field at End**.
15. Add the rc_decimal_no field, which calls the user exit program to process a copy of the DECIMAL_NO field. Define the following properties for the field:

Property	Value	Notes
Name	rc_decimal_no	
Type	NUM32	
Precision	0	
Scale	0	
Length	0	
Phase	RW	Indicates that the operation is read or write.
Expression	`CallProg('UCPGCLSC', 'UCPGCLSC','COBOL', DECIMAL_NO,classtype_dec_no)`	To enter the expression for the field, complete the following steps: 1. Click in the cell in the **Expression** column and click the Browse button. The **Expression Editor** dialog box appears. 2. In the **Function List** list in the **Expression Editor** dialog box, double-click the CallProg function. 3. In the **Expression List** list, enter `('UCPGCLSC','UCPGCLSC','COBOL' ,DECIMAL_NO,classtype_dec_no)` at the end of the CallProg function name. 4. Click **Validate**. In the **Validate** box, the `No Errors` message appears. 5. Click **OK**.

Step 4. Refresh Columns in the MASTER_REC Table

In this step, you refresh the columns in the MASTER_REC table to pick up the user-defined fields that you added to the MASTER_REC record.

1. Open the demo.userexit data map.
2. On the **Data Map** tab in the **Resource Explorer**, right-click the MASTER_REC table and click **Properties**.

   The **Table Properties - Definition** dialog box appears.
3. In the **Column Generation** list, select **Refresh with missing columns**.

   Because the record on which the table is based contains new fields, this action adds the corresponding columns to the table.
4. Click **OK**.

For more information about the **Table Properties - Definition** dialog box, see "Table Properties - Definition" on page 21.

Step 5. Test the Results of the User Exit Program

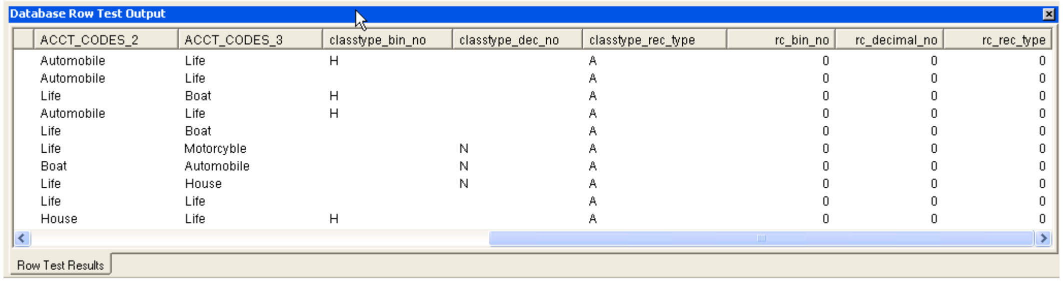
In this step, you run a database row test to test the results of the user exit program in the data map record.

1. Open the demo.userexit data map.
2. On the **Data Map** tab in the **Resource Explorer**, select the MASTER_REC table and click **File** > **Database Row Test**.
3. When prompted to send the data map to a remote location, click **Yes**.

   The **Data Map Remote Node** dialog box appears.
4. In the **Data Map Remote Node** dialog box, enter the user ID, password, and the node for the z/OS system.

   The **Database Row Test** dialog box appears.
5. In the **Database Row Test** dialog box, accept the default values and click **Go**.

   The **Database Row Test Output** window displays the results for the database row test, including the following user-defined fields:



ACCT_CODES_2	ACCT_CODES_3	classtype_bin_no	classtype_dec_no	classtype_rec_type	rc_bin_no	rc_decimal_no	rc_rec_type
Automobile	Life	H		A	0	0	0
Automobile	Life			A	0	0	0
Life	Boat	H		A	0	0	0
Automobile	Life	H		A	0	0	0
Life	Boat			A	0	0	0
Life	Motorcyble		N	A	0	0	0
Boat	Automobile		N	A	0	0	0
Life	House		N	A	0	0	0
Life	Life			A	0	0	0
House	Life	H		A	0	0	0

The user-defined fields display the following information:

User-defined Field	Value	Description
classtype_bin_no	H	Indicates that the BIN_NO field for that row contains a high value.
classtype_dec_no	N	Indicates that the DECIMAL_NO field for that row contains a numeric zoned decimal value.
classtype_rec_type	A	Indicates that the REC_TYPE field for all rows contains an alphabetic value.
rc_bin_no	0	Indicates that the UCPGCLSC user exit program ran successfully for all rows.
rc_decimal_no		
rc_rec_type		

Reference Information

Use the following information to invoke the CallProg function in a user-defined field in a data map record, and to refresh columns in a table to pick up changes to the data map record.

CallProg

Calls a user-defined program or subroutine to process the source data in a record.

Syntax:

```
[result=]CallProg('program','subroutine','linkage'[,arg1][,arg2][,...])
```

The parameters are:

- *result*. Optional. NUM32. This argument contains the return value from program called by the CallProg function, which is one of the following values:
  - **0**. Success.
  - **Non-zero**. Failure.

    If you do not specify a result argument and a non-zero return code is returned from the external program, CallProg executes the following default map-level error responses:
  - Ends the extract.
  - Skips this subroutine.
- *program*. The name of the program that contains the subroutine. Depending on the operating system, the program is one of the following:
  - **i5/OS**. A service program.
  - **Linux or UNIX**. A shared object.
  - **Windows**. A DLL.
  - **z/OS**. A load module.

    Enclose the program name in single quotes.

- *subroutine*. The name of the entry point in the program. Depending on the operating system, the subroutine is one of the following:
  - **i5/OS**. The subroutine name.
  - **Linux, UNIX, or Windows**. The subroutine name.
  - **z/OS for Assembler, C, or COBOL programs**. You must provide a value, but the value is ignored and the default entry point for the load module is used. Specify the same name as the program.
  - **z/OS for PL/I programs**. If multiple fetchable subroutines reside in the same load module, specify the subroutine name.

  Enclose the subroutine name in single quotes.
- *linkage*. The type of linkage, which determines the way that arguments are passed to and return codes are returned from the program or subroutine. The linkage type is one of the following values:

Linkage Type	Supported Operating Systems	Arguments	Returns
C	- i5/OS - Linux, UNIX, and Windows - z/OS	Passed through the stack	Program return code
COBOL	z/OS	Passed as a list of addresses	Address of failure code integer
OS	z/OS		Program return code
OS400	i5/OS		Address of failure code integer
PLI	z/OS		Address of failure code integer

  Enclose the linkage type in single quotes.
- [,*arg1*][,*arg2*][,...]. One or more optional arguments passed to the program or subroutine.

Table Properties - Definition

View or edit table definition properties.

Available Records

A list of the records in the data map that are not in the complex table.

To add a record to the **Record Dependencies** list, in the **Available Records** list, right-click a record and click **Add Record**.

To add a child record to a parent record:

- In the **Record Dependencies** list, select a record to identify it as the parent record.
- In the **Available Records** list, right-click a record and click **Add Record as Child**. This action moves the record to the **Record Dependencies** list as a child record of the parent record.

For IDMS: To select the system index to use for record retrieval, right-click a record and click **Use System Index**. Then, select the index.

Record Dependencies

A list of the records that are in the complex table, with any defined hierarchical dependencies.

To remove a record dependency, right-click a record and click **Delete**.

For IDMS, to reverse the direction of the area read, right-click a record and click **Reverse Area Read**. To reverse the direction of the set read, right-click the record and click **Reverse Set Read**.

How do you want to handle multiple instances of selected records?

Select one of the following options:

- **New Row**. A new row appears or is written to the target for every instance of the record or segment.
- **Ignore**. Second and subsequent instances of a record or segment appear or are written to the target.
- **Array**. The number of records or segments specified in the **Array** list appear or are written to the target in a single row of output.

PowerExchange populates the output row until it is full, and then completes one of the following actions:

- If you clear the **New Row on Overflow** option, PowerExchange ignores subsequent records or segments.
- If you select the **New Row on Overflow** option, PowerExchange displays a new row with the overflow records or segments.

For example, for a record with five instances, if you enter 3 in the **Array** list, PowerExchange builds two output rows. The first row contains an array of three instances, and the second row contains an array of two instances.

Note: If you set a parent record or segment to **Array**, you must set all child records or segments to **Ignore**.

Column Generation

When you first define a table, PowerExchange derives column names from the field names in the record on which the table is based. However, PowerExchange uses a special naming convention for records that contain fields defined as arrays.

Select one of the following options to indicate how to refresh columns in a table after you change field definitions in the record on which the table is based:

- **Apply array format changes**. If the record on which the table is based contains changed fields that defined as arrays, those changes are reflected in the corresponding columns in the table.
- **Refresh with missing columns**. If the record on which the table is based contains new fields, corresponding columns are added to the table.
- **Reset to defaults**. PowerExchange resets column names to the corresponding fields names in the record on which the table is based. PowerExchange discards any changes that you made to column names in the table.
- **Remove Hidden Columns**. For DB2UNLD. PowerExchange generates a with the default values of **Hide from Table** for each field.

Fields

Select or clear a field to control how elements in an array or group field appear:

- To display each element in an array or group field in a single row, select the field.
- To display each element in an array or group field in a separate row, clear the field.

  Note: To display fields defined as a group field or as an array in the **Fields** list, select the **Groups and Arrays only** option. To display all fields in the record in the Fields list, clear this option. This option controls the fields that appear in the **Fields** list and in the **Table Properties** dialog box, but not the fields that appear in the **Database Row Test Output** window.

How do you want to handle multiple instances of selected records?

- **New Row**. A new row appears or is written to the target for every element in the array.
- **Ignore**. PowerExchange does not display, or write to the target, second and subsequent elements in the array.
- **Array**. PowerExchange displays, or writes to the target, the number of elements specified in the **Array** list in a single row of output.

PowerExchange populates the output row until it is full, and then completes one of the following actions:

- If you clear the **New Row on Overflow** option, PowerExchange ignores subsequent records or segments.
- If you select the **New Row on Overflow** option, PowerExchange displays a new row with the overflow records or segments.

Multiple Arrays in a Single Input Row

Generates multiple output rows from a single record that contains multiple arrays, or OCCURS clauses. PowerExchange sets the output fields to NULL when the data in the record is exhausted.

Enabled for a table with an imported COPYLIB with multiple OCCURS clauses.

Author

Diane Fleming