Data Documentation and Retrieval Using Unity in a UniVerse® Environment

Progress report – Summer 2002

July 28, 2002

Jose JIMENEZ

University of Iowa, Iowa City, Iowa

jose_jimenez73@hotmail.com

Goals

The goal of the second phase of this project was to increase the amount of information available to Unity from the target database.[1] In order to accomplish this, a program was planned to examine dictionary items and populate the @select item of the data files' dictionaries.[2]

Details

Environment

Phase two of the project was implemented on the same system as the host database. The host database is a UniVerse® version 9.5.1.1.fr1-1 database with multiple files. It runs on an IBM model P620 RS6000 system. The operating system is AIX version 4.3. The native programming language of UniVerse® is UniBASIC, a compiled version of BASIC with extensions used to work with the databases directly. UniVerse® files use companion files called dictionaries to provide access to a variety of query methods. These methods include a built in reporting language called Retrieve which is a variant of SQL, locally executed SQL queries, and ODBC clients. Both SQL and the ODBC clients rely on an entry in the dictionary of the file to determine which fields, generally referred to as columns in ODBC and SQL, are available to return. This entry is always labeled "@select" and is generally updated manually. Entries may be in the dictionary file for a data file without being in the @select entry of its dictionary file. When using ODBC or SQL to access a UniVerse® file, each data file is seen as a table and each entry in the @select dictionary item labels a column of that table.

Specification

The program that would populate the @select dictionary item for the data files would have to add items to a simple list separated by spaces. The challenge for the program would be to select only items that would work with ODBC and SQL. Some items that would not work with ODBC and SQL are items that are right justified and have both numeric and alphabetic data. Other items that would pose problems are items that don't execute correctly to bring back the appropriate information. In UniVerse®, a dictionary item can implement joins to another data file. Dictionary items may also implement complicated decision matrices by executing programs to return a value for the column. If a dictionary item refers to a program that cannot be executed or refers to a file that is not available, it is invalid. Invalid dictionary items and items that refer to other items that are invalid should not be added to the @select entry.

Description

The task of selecting valid entries from those available in the dictionary file is made more complicated by the fact that dictionaries can be any of four different types. The structure of dictionary items varies by type. The program written, ODBC.DICT.CHECK, includes code that differentiates between the different possible types and parses each dictionary into its component pieces. The type is determined by the first attribute of the file. Attributes in UniVerse® correspond to columns of a table. Within the UniVerse® host environment, even dictionary items are separated into separate attributes that can be accessed individually by UniBASIC programs. Unfortunately, this information is not accessible using ODBC or SQL statements outside of UniVerse®. Some of the other available dictionary attributes are the base field and the method used to manipulate the data in order to arrive at a value. Each dictionary item is in turn checked for consistency, and validity. Right-justified fields merit further investigation since it is impossible for the program to tell simply from looking at the dictionary item what kind of information is actually in the data file. In order to determine whether the field suffers from having mixed alpha and numeric data, a sampling of the data is taken to determine if there are any non-numeric values in it.

The program can validate the entries in the dictionary file and generate a report or update the @select entry when it executes. When the program works on a given file in update mode, it makes a backup of the previously used data. It also writes a signature to the @select entry of the dictionary file including the name of the program that updated the file, the date, and the time of update. When adding dictionary items to the @select entry, it copies the entries to new names that have more information than would be available otherwise. The naming convention that it uses is made up of several pieces of information separated by a "@" which is converted by some ODBC clients to an underscore. These pieces of information are the base field number if available, the original name of the field and the text label used for reporting purposes. This results in fairly long names, but it is usually easier to tell what the field contains using these new names as compared to using the original names.

Implementation

ODBC.DICT.CHECK is available in the appendix.

Results

Before executing ODBC.DICT.CHECK, the sample file used had fifty seven (57) fields available for queries via ODBC and SQL. Once ODBC.DICT.CHECK was executed, five hundred and twenty one (521) fields were available for queries. The additional fields were those accepted by ODBC.DICT.CHECK as valid fields. A simple increase in the number of fields available is not necessarily an improvement if there is no improvement in the quality of the information available. The benefit of the additional columns and the improved naming methodology became apparent when accessing the fields using Unity. Deciding on semantic names and whether to add these fields to the specification of the data source was much easier with the new data compared to the old data. Quantifying this improvement was not attempted at this time, however, going from a label such as "4" on a dictionary item to a label of "O@4@F4@TAXCODE" on the column, the improvement is quickly visible. The first may give a hint at what column number the data comes from. The second gives an indication the data in the column is a TAXCODE, which is much easier to work with and assign a semantic name to. The example given is an actual data field from the sample table.

Further work for the project

The next phase of the project is to create a source, specification and schema for the database once more. This would entail a repeat of some of the work done during spring 2002, but would not have to overcome the same problems. This would give a baseline for the amount of documentation possible manually using Unity. This had been tentatively scheduled for summer of 2002, but was not completed. Once the source, specification and schema are set up, queries will be executed against the host database with Unity. The results will be compared to doing similar queries against the host database on the host system.

The fall of 2002 may also be used to increase the amount of automated documentation available. Some of this work has already been done in ODBC.DICT.CHECK. Simply adding the column label to the name of the field increases the amount of information available to anyone using ODBC or SQL to connect to the UniVerse® database. Additional work may be done to generate X-Specs for use in Unity using programs written in the UniVerse® environment.

Summary

Simple programs and programming languages can be used to much advantage when it comes to automated documentation. This phase of the project involved creating a program in a language similar to BASIC that increased the number of columns and the quality of the names of columns available in a target database. Deciphering a name like "F79" from outside of the host system requires significantly more work than "O@79@F79@SHIP_DAYS", so long as the "@" is known to be a separator. The next phase of the project will focus on modifying ODBC.DICT.CHECK to optionally produce an X-Spec file describing the table/tables verified or updated with information about the columns and fields available.

Appendix

```
*
* V:1.0
* U:???
*** Copyright 2002, etc., Blooming Prairie Cooperative Warehouse (BPW)
*** Confidential and proprietary information of BPW,
*** Iowa City, IA.  Use or dissemination allowed only with
*** prior written consent of BPW
*
*  PROGRAM NAME : ODBC.DICT.CHECK
*  WRITTEN BY   : JMJ
*  DATE WRITTEN : 6/1/02
*  DESCRIPTION  : CHECK DICTIONARY ITEMS FOR ODBC COMPATIBILITY
*
* MODIFICATIONS:
* ??? ???               ??-??-??        ?????????????????????????
*

*
*** COMMONS
*


*** EQUATES ***
      EQU TRUE TO 1
      EQU FALSE TO 0

*** DEFINE MISC VARIABLES *
      AM=CHAR(254)
      VM=CHAR(253)
      SM=CHAR(252)
      CLR=@(-1)
      BELL=CHAR(7)
      CLL=@(-4)
      CEOS=@(-3)
      EL=@(0,23):BELL:CLL
      EEL=@(0,23):CLL
      PL=@(0,22):CLL
      ULON=@(-15)
      ULOFF=@(-16)
      LF=CHAR(10)
      FF=CHAR(12)
      CR=CHAR(13)
      TAB=CHAR(9)
      PROMPT ''
      TODAY = DATE()
```

```
*
*** OPEN FILES
*

      OPEN 'VOC' TO VOC ELSE
         PRINT 'UNABLE TO OPEN VOC.  PRESS RETURN TO CONTINUE'
      END

*
*** MAIN
*


      PGM.NAME = "ODBC.DICT.CHECK"
      PGM.TITLE = "ODBC DICTIONARY CHECK"

      DEFFUN F.TRIMLOW(I.STRING)         ; * FUNCTION TO REMOVE CHARACTERS BELOW HEX 20

* THIS PROGRAM ASSUMES IT WILL BE WORKING WITH A LIST OF FILES TO
* VERIFY OR UPDATE.  UniVerse SUPPORTS A NAMED LIST OF FILES.

GET.LIST.NAME:
      PRINT 'ENTER LIST NAME TO USE E=EXIT : ':
      INPUT LIST.NAME

      IF LIST.NAME = 'E' THEN GO PRGXIT
      IF LIST.NAME = '' THEN GO GET.LIST.NAME

* VERIFY THE FILE LIST
      GETLIST LIST.NAME ELSE
         PRINT 'UNABLE TO GET LIST.  PRESS RETURN TO CONTINUE.':
         INPUT JUNK
         GO GET.LIST.NAME
      END

      READLIST FILES.TO.CHECK ELSE
         PRINT 'UNABLE TO GET LIST.  PRESS RETURN TO CONTINUE.':
         INPUT JUNK
         GO GET.LIST.NAME
      END

      FILE.COUNT = DCOUNT(FILES.TO.CHECK,@AM)      ; * FIND THE NUMBER OF FILES TO REPORT ON

* USE @SELECT OR FULL DICTIONARY?
GET.DICT.SEL.FLAG:
      PRINT 'PRESS RETURN TO USE @SELECT ENTRIES, ANY OTHER KEY TO USE FULL DICTIONARY : ':
```

```
      INPUT DICT.SEL.FLAG

* GET THE RESPONSE TO UPDATE OR SIMPLY VERIFY
      PRINT 'ENTER "Y" TO CREATE NEW DICTIONARY ENTRIES WITH EXPANDED NAMES.  ANY OTHER KEY TO SKIP STEP.'
      INPUT DICT.UPDATE

* GET THE RESPONSE TO ADD OLD ENTRIES
      PRINT 'ENTER "Y" TO CREATE KEEP OLD ENTRIES AS WELL AS NEW ONES.  ANY OTHER KEY TO SKIP STEP.'
      INPUT KEEP.OLD.FLAG


      FOR FL = 1 TO FILE.COUNT
         FILE.ASSOC = ''                  ; * USED TO KEEP TRACK OF FILE ASSOCIATIONS
* OPEN THE DICTIONARY OF THE FILE
         THIS.FILE = FILES.TO.CHECK<FL>
         IF DICT.UPDATE = 'Y' THEN
            PRINT @(0,10):CLL:"ON ":FL:"/":FILE.COUNT:"   ":THIS.FILE  ; * DISPLAY A COUNTER
         END
         CLOSE FILE.DICT
         OPEN 'DICT ':THIS.FILE TO FILE.DICT ELSE
            PRINT @(0,15):CLL:'DICT ':THIS.FILE:' NOT FOUND. UNABLE TO PROCESS.'
            GO SKIP.TO.NEXT.DICT
         END
         IF DICT.SEL.FLAG = '' THEN
            *   IF USING THE @SELECT ENTRY AND THERE IS NOT ONE,
            READ SEL.DICT FROM FILE.DICT, '@SELECT' ELSE
               PRINT @(0,15):CLL:'DICT ':THIS.FILE:' @SELECT RECORD NOT FOUND.  UNABLE TO PROCESS.'
               GO SKIP.TO.NEXT.DICT
            END
         END ELSE
            DICT.STMT = 'SELECT DICT ':THIS.FILE
            EXECUTE DICT.STMT CAPTURING JUNK
            READLIST SEL.DICT ELSE
               PRINT @(0,15):CLL:'DICT ':THIS.FILE:' NO RECORDS FOUND. UNABLE TO PROCESS.'
               GO SKIP.TO.NEXT.DICT
            END
            SEL.DICT = @AM:CHANGE(SEL.DICT,@AM,' ')
         END
         WORD.COUNT = DCOUNT(SEL.DICT<2>,' ')      ; * THE NUMBER OF ITEMS TO VERIFY
         NEW.SEL.DICT = ''                ; *  USED TO STORE NEW NAMES TO ADD TO @SELECT ENTRY
         OLD.SEL.DICT = ''                ; *  USED TO STORE OLD NAMES TO ADD TO @SELECT ENTRY
         DEP.LIST = ''                    ; *  USED TO TRACK DEPENDENCIES
         FOR WL = 1 TO WORD.COUNT
            THIS.WORD = SEL.DICT<2>[' ',WL,1]
            PRINT @(0,20):CLL:"ON ":WL:"/":WORD.COUNT:"   ":THIS.WORD
            READ THIS.DICT FROM FILE.DICT, THIS.WORD ELSE
               PRINT @(0,21):CLL:'DICT ':THIS.FILE:' ENTRY ':THIS.WORD:' NOT FOUND. UNABLE TO PROCESS.'
               GO SKIP.THIS.DICT
            END
            THIS.TYPE = THIS.DICT<1>[1,1]          ; * DICTIONARY TYPE
            THIS.ASSOC = ''              ; * USED TO STORE ASSOCIATION WITH OTHER MULTIVALUED FIELDS
            THIS.CORR = ''               ; * USED TO STORE INFO ABOUT DATA MANIPULATION FOR CERTAIN TYPES
            THIS.FIELD = 0
           THIS.EVAL = ''                ; * USED TO STORE INFOR ABOUT CODE THAT IS EXECUTED
            BEGIN CASE
               CASE THIS.TYPE = 'D' OR THIS.TYPE = 'I'
                  THIS.LABEL = THIS.DICT<4>
                  THIS.JUST = THIS.DICT<5>
                  THIS.JUST = THIS.JUST[1]
               CASE THIS.TYPE = 'A' OR THIS.TYPE = 'S'
                  THIS.LABEL = THIS.DICT<3>
                  THIS.ASSOC = THIS.DICT<4>
                  THIS.JUST = THIS.DICT<9>
                  THIS.CORR = THIS.DICT<8>:" ":THIS.DICT<7>
               CASE 1
                  PRINT @(0,21):CLL:'DICT ':THIS.FILE:' ENTRY ':THIS.WORD:' HAS TYPE ':THIS.TYPE:'. UNABLE TO PROCESS.'
                  GO SKIP.THIS.DICT
            END CASE

* LOOK FOR INVALID SUBROUTINE CALLS IN 'I' TYPE DICTIONARIES
            IF THIS.TYPE = 'I' THEN
               THIS.EVAL = THIS.DICT<2>  ; * USUALLY CODE THAT IS EXECUTED
               THIS.OCC = 0
               THIS.SEARCH = 'SUBR('     ; * MARKER FOR CALLED SUBROUTINE
               PASS = 1
               LOOP
I.L.TOP:
                  THIS.OCC += 1
                  SUB.IND = INDEX(THIS.EVAL,THIS.SEARCH,THIS.OCC)
                  IF PASS = 1 AND SUB.IND = 0 THEN
                     THIS.SEARCH = 'TRANS('        ; * MARKER FOR JOIN WITH ANOTHER FILE
                     THIS.OCC = 0
                     PASS = 2
                     GO I.L.TOP
                  END
               WHILE SUB.IND > 0
                  C.IND = SUB.IND + INDEX(THIS.EVAL[SUB.IND,99999],',',1)
                  THIS.ROUTINE = THIS.EVAL[SUB.IND+6,C.IND-2]
                  * WHETHER IT IS A SUBROUTINE OR A JOIN, THE VALUE OF THIS.ROUTINE
                  * MUST APPEAR IN THE VOC (MASTER LISTING OF PROGRAMS/FILES)
                  * IN ORDER FOR THE FIELD TO BE VALID
                  READ JUNK FROM VOC,THIS.ROUTINE THEN
                     NULL
                  END ELSE
```

```
                     PRINT @(0,21):CLL:'DICT ':THIS.FILE:' ENTRY ':THIS.WORD:' HAS INVALID CALL TO ':THIS.ROUTINE:'. UNABLE TO
        PROCESS.'
                     GO SKIP.THIS.DICT
                  END
               REPEAT
            END

* LOOK FOR REFERENCES TO FILES NOT AVAILABLE
* THIS IS ONLY EXECUTED FOR DICTIONARY TYPES OTHER THAN "I" SINCE
* THE ABOVE CODE HANDLES THOSE.  THIS IS CONTROLLED BY THE VALUE OF
* THIS.CORR BEING SET TO NULL ABOVE.

            T.OCC = 0
            PASS = 1
            T.SEARCH = '(T'                ; * MARKER FOR FILE JOINS
            LOOP
T.L.TOP:
               T.OCC +=1
               TIND = INDEX(THIS.CORR,T.SEARCH,T.OCC)
               IF TIND = 0 AND PASS = 1 THEN
                  T.SEARCH = @VM:'T'      ; * THE SECOND PASS LOOKS FOR
                  * occurrences in multivalued correlatives.
                  T.OCC = 0
                  PASS = 2
                  GO T.L.TOP
               END
            WHILE TIND > 0
               SIND = TIND + INDEX(THIS.CORR[TIND,99999],';',1)
               T.FILE = TRIM(THIS.CORR[TIND+2,SIND-1])
               OPEN T.FILE TO T.JUNK THEN
                  CLOSE T.JUNK
               END ELSE
                  GO SKIP.THIS.DICT
               END
            REPEAT

            THIS.LOC = THIS.DICT<2>
            IF THIS.TYPE = "I" THEN THIS.LOC = "I"           ; * I TYPE FIELDS DO NOT HAVE
            * BASE FIELDS. THEY GENERALLY EXECUTE
            * CODE THAT CAN PULL DATA FROM
            * ANY FIELD/FILE OR DECISION MATRIX
            * THE NEW NAME OF THE FILE WOULD BE MULTIPLE PARTS
            * PART 1: TAG FOR ODBC DICT = "O"
            * PART 2: THE BASE FIELD NUMBER FOR THE DICTIONARY ITEM
            * PART 3: THE ORIGINAL NAME OF THE COLUMN
            * PART 4: THE TEXT LABEL FOR THE COLUMN WHEN PRINTING REPORTS FROM RETREIVE
            * CONVERT TILDA'S, COMMAS, AND PERIODS TO SPACE IN THE ORIGINAL TO AVOID CONFLICTS
            * CONVERT SPACES TO UNDERSCORES IN THE RESULT


            NEW.NAME = "O@":THIS.LOC:"@":CONVERT("@"," ",THIS.WORD):"@":TRIM(CONVERT("."," ",CONVERT(@VM," ",CONVERT("@","
        ",THIS.LABEL))))
            I.STRING = NEW.NAME
            NEW.NAME = CONVERT(\"' \,"_",F.TRIMLOW(I.STRING))
            IF THIS.JUST = 'R' THEN
               * IF THE FIELD IS RIGHT JUSTIFIED, SAMPLE THE DATA TO VERFIY
               THIS.STMT = 'SELECT ':THIS.FILE:' WITH ':THIS.WORD:' SAVING ':THIS.WORD:' SAMPLE 100'
               EXECUTE THIS.STMT CAPTURING JUNK
               READLIST SAMPLES ELSE
                  PRINT @(0,21):CLL:'DICT ':THIS.FILE:' ENTRY ':THIS.WORD:' NO SAMPLES. UNABLE TO PROCESS.'
                  GO SKIP.THIS.DICT
               END
               IF NOT(NUMS(SAMPLES)) THEN
                  PRINT @(0,21):CLL:'DICT ':THIS.FILE:' ENTRY ':THIS.WORD:' HAS INVALID DATA.  RIGHT JUSTIFIED ALPHA DATA.'
               END ELSE
                  LOCATE(NEW.NAME,NEW.SEL.DICT;NEW.POS) ELSE
                     NEW.SEL.DICT = INSERT(NEW.SEL.DICT,NEW.POS;NEW.NAME)         ; * ADD NEW NAME TO LIST
                     OLD.SEL.DICT = INSERT(OLD.SEL.DICT,NEW.POS;THIS.WORD)        ; * ADD ORIGINAL NAME
                     * TO LIST FOR ASSOCIATIONS
                  END
                  IF DICT.UPDATE = 'Y' THEN
                     WRITE THIS.DICT ON FILE.DICT, NEW.NAME ELSE
                        PRINT @(0,21):CLL:'UNABLE TO CREATE NEW DICT FOR ':THIS.WORD:' - ':NEW.NAME
                     END
                  END
               END
            END ELSE
               LOCATE(NEW.NAME,NEW.SEL.DICT;NEW.POS) ELSE
                  NEW.SEL.DICT = INSERT(NEW.SEL.DICT,NEW.POS;NEW.NAME)            ; * ADD NEW NAME TO LIST
                  OLD.SEL.DICT = INSERT(OLD.SEL.DICT,NEW.POS;THIS.WORD)           ; * ADD OLD NAME TO LIST
               END
               IF DICT.UPDATE = 'Y' THEN
                  WRITE THIS.DICT ON FILE.DICT, NEW.NAME ELSE
                     PRINT @(0,21):CLL:'UNABLE TO CREATE NEW DICT FOR ':THIS.WORD:' - ':NEW.NAME
                  END
               END
            END
* ADD INFO ABOUT ASSOCIATIONS TO FILE.ASSOC
            IF NUM(THIS.LOC) THEN
               THIS.FIELD = THIS.LOC+0
               IF THIS.ASSOC[1,1]='D' THEN
                  THIS.ASSOC.TO = TRIM(THIS.ASSOC[3,99999])
                  IF NUM(THIS.ASSOC.TO) THEN
                     IF FILE.ASSOC<2,THIS.ASSOC.TO> = '' THEN
```

```
                        FILE.ASSOC<2,THIS.ASSOC.TO,1> = THIS.FIELD
                        FILE.ASSOC<3,THIS.ASSOC.TO,1> = NEW.NAME
                     END ELSE
                        LOCATE(THIS.FIELD,FILE.ASSOC,2,THIS.ASSOC.TO;F.POS) ELSE
                           FILE.ASSOC = INSERT(FILE.ASSOC,2,THIS.ASSOC.TO,F.POS;THIS.FIELD)
                           FILE.ASSOC = INSERT(FILE.ASSOC,3,THIS.ASSOC.TO,F.POS;NEW.NAME)
                        END
                     END
                  END
               END ELSE
                  THIS.ASSOC = TRIM(THIS.ASSOC[3,9999])
                  THIS.ASSOC.LEN = LEN(THIS.ASSOC)
                  IF THIS.ASSOC[THIS.ASSOC.LEN,1] = ';' THEN
                     THIS.ASSOC = THIS.ASSOC[1,THIS.ASSOC.LEN]
                  END
                  ASS.CNT = DCOUNT(THIS.ASSOC,';')
                  FOR A.L = 1 TO ASS.CNT
                     A.L.ASSOC = THIS.ASSOC[';',A.L,1]
                     IF NUM(A.L.ASSOC) THEN
                        A.L.ASSOC +=0
                        IF FILE.ASSOC<1,A.L.ASSOC> = '' THEN
                           FILE.ASSOC<1,A.L.ASSOC,1> = THIS.FIELD
                           FILE.ASSOC<4,A.L.ASSOC,1> = NEW.NAME
                        END ELSE
                           LOCATE(A.L.ASSOC,FILE.ASSOC,1,THIS.FIELD;F.POS) ELSE
                              FILE.ASSOC = INSERT(FILE.ASSOC,1,A.L.ASSOC,F.POS;THIS.FIELD)
                              FILE.ASSOC = INSERT(FILE.ASSOC,4,A.L.ASSOC,F.POS;NEW.NAME)
                           END
                        END
                     END
                  NEXT A.L
               END
            END
* TRACK DEPENDENCIES
            IF THIS.EVAL MATCH "...<0N,0N>..." THEN
               W.START = 1
               W.END = W.START
               E.LEN = LEN(THIS.EVAL)
               C.IND = W.START
               LOOP
                  IF THIS.EVAL[C.IND,1] = ' ' THEN
                     W.START = C.IND +1
                  END ELSE
                     IF THIS.EVAL[C.IND,1] = '<' THEN
                        POSS.WORD = THIS.EVAL[W.START,C.IND-1]
                        G.MARK = INDEX(THIS.EVAL[C.IND,9999],'>',1)
                        IF G.MARK > 0 THEN
```

```
                           IND.NUMS = THIS.EVAL[C.IND+1,G.MARK]
                           IF DCOUNT(IND.NUMS,',') < 3 AND NUMS(CHANGE(IND.NUMS,',',@AM)) THEN
                              IF DEP.LIST<1> = '' THEN
                                 DEP.LIST<1,1> = NEW.NAME
                                 DEP.LIST<2,1> = POSS.WORD
                              END ELSE
                                 DEP.LIST<1,-1> = NEW.NAME
                                 DEP.LIST<2,-1> = POSS.WORD
                              END
                           END
                        END
                     END
                  END
                  C.IND +=1
               WHILE C.IND < E.LEN
               REPEAT
            END
SKIP.THIS.DICT:
         NEXT WL
         IF DICT.UPDATE = 'Y' THEN
* REMOVE DICTIONARIES WITH INVALID ASSOCIATIONS
            ASS.CNT = MAXIMUM(DCOUNT(FILE.ASSOC<2>,@VM):@AM:DCOUNT(FILE.ASSOC<1>,@VM))
            FOR A.L = 1 TO ASS.CNT
               A2.CNT = MAXIMUM(DCOUNT(FILE.ASSOC<2,A.L>,@SM):@AM:DCOUNT(FILE.ASSOC<4,A.L>,@SM))
               FOR A2.L = 1 TO A2.CNT
                  A2.L.ASSOC = FILE.ASSOC<2,A.L,A2.L>
                  IF A2.L.ASSOC = '' THEN GO SKIP.A2.ASSOC
                  LOCATE(A2.L.ASSOC,FILE.ASSOC,1;JUNK) ELSE
                     A2.DICT.NAME = FILE.ASSOC<3,A.L,A2.L>
                    PRINT @(0,21):CLL:"DELETING DICT ":A2.DICT.NAME:" ASSOCIATION PROBLEM"
                     DELETE FILE.DICT, A2.DICT.NAME ELSE
                        NULL
                     END
                     LOCATE(A2.DICT.NAME,NEW.SEL.DICT;A2.L.POS) THEN
                        DEL NEW.SEL.DICT<A2.L.POS>
                        DEL OLD.SEL.DICT<A2.L.POS>
                     END
                  END
SKIP.A2.ASSOC:
                  A2.L.ASSOC = FILE.ASSOC<1,A.L,A2.L>
                  IF A2.L.ASSOC = '' THEN GO SKIP.A2.ASSOC2
                  LOCATE(A2.L.ASSOC,FILE.ASSOC,2;JUNK) ELSE
                     A2.DICT.NAME = FILE.ASSOC<4,A.L,A2.L>
                    PRINT @(0,21):CLL:"DELETING DICT ":A2.DICT.NAME:" ASSOCIATION PROBLEM"
                     DELETE FILE.DICT, A2.DICT.NAME ELSE
                        NULL
                     END
```

```
                      LOCATE(A2.DICT.NAME,NEW.SEL.DICT;A2.L.POS) THEN
                         DEL NEW.SEL.DICT<A2.L.POS>
                         DEL OLD.SEL.DICT<A2.L.POS>
                      END
                   END
SKIP.A2.ASSOC2:
                NEXT A2.L
             NEXT A.L
             D.COUNT = DCOUNT(DEP.LIST<1>,@VM)
             FOR D.L = 1 TO D.COUNT
                THIS.DEP = DEP.LIST<1,D.L>
                LOCATE(THIS.DEP,NEW.SEL.DICT;D.POS) THEN
                   LOCATE(DEP.LIST<2,D.L>,OLD.SEL.DICT;D2.POS) ELSE
                     PRINT @(0,21):CLL:"DELETING DICT ":THIS.DEP:" DEPENDENCY PROBLEM"
                      DELETE FILE.DICT, THIS.DEP ELSE
                         NULL
                      END
                      LOCATE(THIS.DEP,NEW.SEL.DICT;D3.POS) THEN
                         DEL NEW.SEL.DICT<D3.POS>
                         DEL OLD.SEL.DICT<D3.POS>
                      END
                   END
                END
             NEXT D.L
* WRITE OUT NEW @SELECT ENTRY
             READU ORIG.SEL.DICT FROM FILE.DICT,"@SELECT" ELSE
                ORIG.SEL.DICT = ''
             END
             READU SAVE.SEL.DICT FROM FILE.DICT,"ODBC.SAVE.SEL.DICT" ELSE
                SAVE.SEL.DICT = ''
             END
             SAVE.SEL.DICT = ORIG.SEL.DICT:@AM:"------SEPARATOR------":@AM:SAVE.SEL.DICT
             WRITE SAVE.SEL.DICT ON FILE.DICT,"ODBC.SAVE.SEL.DICT" ELSE
                PRINT @(0,21):CLL:'UNABLE TO WRITE SAVED SELECT DICT INFO.'
             END
            IF KEEP.OLD.FLAG = 'Y' THEN
               TEMP.SEL.DICT = CHANGE(SAVE.SEL.DICT<2>," ",@AM)
               O.CNT = DCOUNT(TEMP.SEL.DICT,@AM)
               FOR O.L = 1 TO O.CNT
                  OLD.NAME = TEMP.SEL.DICT<O.L>
                   LOCATE(OLD.NAME,NEW.SEL.DICT;NEW.POS) ELSE
                      NEW.SEL.DICT = INSERT(NEW.SEL.DICT,NEW.POS;OLD.NAME)           ; * ADD NEW NAME TO LIST
                   END
               NEXT
            END
             NEW.SEL.DICT = CHANGE(NEW.SEL.DICT,@AM,' ')
             NEW.SEL.DICT<2> = NEW.SEL.DICT
            NEW.SEL.DICT<1> = 'PH Updated by ODBC.DICT.CHECK ':OCONV(DATE(),'D2/'):" @ ":OCONV(TIME(),'MTH')
            WRITE NEW.SEL.DICT ON FILE.DICT,"@SELECT" ELSE
               PRINT @(0,21):CLL'UNABLE TO WRITE NEW @SELECT DICTIONARY FOR FILE'
            END
         END
         PRINT 'PRESS RETURN TO CONTINUE': ; INPUT JUNK
SKIP.TO.NEXT.DICT:
      NEXT FL
*
*** EXIT PROGRAM
*
PRGXIT:


   END
```

References

[1] J. Jimenez: Data Documentation and Retrieval Using Unity in a UniVerse® Environment. February 21, 2002.

[2] J. Jimenez: Data Documentation and Retrieval Using Unity in a UniVerse® Environment – Progress Report – Spring 2002. May 15, 2002.