

U2 SELECT verb list extension

Select records based on dictionary matches to a saved list or select buffer

Skill Level: Intermediate

[Peter M. Newby \(pnewby@epicor.com\)](mailto:pnewby@epicor.com)
Principal Software Engineer
Epicor Software, Corp.

11 Jun 2009

This article provides a general solution to the problem of record selection based on a large list of non-primary-key values. It is a step-by-step guide to building a new verb which selects records based on a match between dictionary values and standard IBM® UniData® SAVE.LIST and SELECT buffer lists. For those who like to build their own tools, it offers an interesting project. For those who prefer to use tools others have built, the completed verb source code is available as a download.

Problem definition

UniData SELECT and SAVE.LIST processors provide tremendous power and flexibility for isolating sets of data. Lists can be saved, sorted, and merged according to UNION or INTERSECTION. Generally these lists are composed of primary keys to the files of interest, and are then used in conjunction with SELECT, SORT, or LIST.

However using QSELECT, or the SAVING and UNIQUE keywords of the SELECT verb, can generate a list based on any attribute or dictionary of the base file. Especially interesting are dictionaries which evaluate to foreign keys, thus converting from one file's primary key to another's. These techniques can solve a lot of knotty selection problems, but not all.

One thing that you cannot currently do is to activate a list and then SELECT items

from the file based on a foreign key match. The active SELECT list always acts on the primary key.

To make this more concrete, let's suppose we have a list of several hundred of our many, many customers and that we are interested in reporting on their order history. For one customer (or even 5 or 10), we can easily `SELECT ORDERS WITH CUST.NBR = '1010' '1013' '1021' '1044' . . .`, but not for several hundred.

We cannot activate a list of customer numbers and then `SELECT ORDERS` with those customer numbers.

To use a `SAVE.LIST` with `SELECT` we must generate a list of Item Ids from the `ORDERS` file. Sometimes that is just not possible. Perhaps the list was created manually or came from a foreign system, and there simply is no code or category in the MV database for grouping our desired customers.

The manual solution

The manual solution to this problem in a MultiValue environment has been re-invented many times, by many individuals:

1. Create a temporary file
2. Fill it with items keyed by the customer numbers of interest
3. Create a dictionary in the `ORDERS` file that translates to the temp file and returns a value if present or null if not
4. `SELECT ORDERS` records with the new dictionary value # "
5. Clean up

The user interface

We would like to automate the manual solution, and ideally, make it as easy and natural to use as the UniData `SELECT` processors:

```
LSELECT ORDERS WITH CUST.NBR IN MY.LIST
```

Yes, this could be really cool!

A more formal interface definition can be given as:

```
LSELECT filename WITH dictname [IN list_num][IN list_name] [TO list_num][TO list_name]
```

It should include the ability to work on the current active list and to deliver the results to the current active list when no other options are specified.

Key requirements

On most MultiValue platforms, you can write a program with a UI that acts like a SELECT verb by making use of the @SENTENCE intrinsic variable which will return the user's or paragraph's entire line entry.

You will need:

- A parser to sort out what the user typed in, or what the paragraph specified
- Code to create and populate a file appropriately sized for the list of keys supplied.

Since each use of your verb will temporarily create a disk file, you'll want to think about concurrency. What happens when two users try to use LSELECT at the same time?

You won't actually need to create and clean up dictionaries in your permanent file under UniData (ECLTYPE='U'), as there is a UniQuery keyword, "EVAL", which will let you select the records with an on-the-fly I-descriptor definition. But you will need code to build the appropriate translate string. For UniVerse users and UniData user not running with ECLTYPE=U, you will probably need to write some additional code and actually write out a temporary dictionary item.

LSELECT

You can break down your verb into six tasks:

1. GET.COMMAND
2. GET.SOURCE.SELECT.LIST
3. CREATE.TEMP.FILE
4. LOAD.TEMP.FILE
5. BUILD.RESULT.LIST

6. FINISH.UP

This task-list forms our mainline. Just replace the number with "GOSUB " and then add a "STOP" line.

GET.COMMAND:

Below is a simple parser implementation for this particular command structure. It includes a few nice features like case and space insensitivity, dictionary validation, and a 'usage' message to help the user out (meaning us in about 3 months).

Command parser code listing

```

GET.COMMAND:
* initialize command variables
FILE.NAME = ''
DICT.NAME = ''
OPER = ''
SOURCE.LIST = ''
TO.OP = ''
DEST.LIST = ''
SOURCE.LIST.NAMED = 1
DEST.LIST.NAMED = 1
DISPLAY.HELP = 0
*
* Gather and parse input from command line
*
INPUT.PARAMS = TRIM(FIELD(@SENTENCE, ' ', 2, 999))
SWAP ' ' WITH ';' IN INPUT.PARAMS
POS = 1
FILE.NAME = TRIM(FIELD(INPUT.PARAMS, ';', POS))
POS += 1
WITH.OP = UPCASE(TRIM(FIELD(INPUT.PARAMS, ';', POS)))
IF WITH.OP = 'WITH' THEN POS += 1
DICT.NAME = TRIM(FIELD(INPUT.PARAMS, ';', POS))
POS += 1
OPER = UPCASE(TRIM(FIELD(INPUT.PARAMS, ';', POS)))
IF OPER = 'IN' THEN
POS += 1
SOURCE.LIST= TRIM(FIELD(INPUT.PARAMS, ';', POS))
POS += 1
TO.OP = UPCASE(TRIM(FIELD(INPUT.PARAMS, ';', POS)))
IF TO.OP = 'TO' THEN POS += 1
DEST.LIST = TRIM(FIELD(INPUT.PARAMS, ';', POS))
END ELSE
IF OPER = 'TO' THEN
POS += 1
DEST.LIST = TRIM(FIELD(INPUT.PARAMS, ';', POS))
END ELSE
IF OPER # '' THEN
MSG = 'Expecting "IN" or "TO" clause. Found ':OPER
GOSUB SHOW.MESSAGE
DISPLAY.HELP = 1
END
END
END
*
IF FILE.NAME = '' OR FILE.NAME = '?' OR FILE.NAME = 'HELP'
OR DICT.NAME = '' THEN

```

```

DISPLAY.HELP = 1
END
*
* Display Usage if parameters not supplied or help is requested
*
IF DISPLAY.HELP THEN
GOSUB SHOW.HELP
STOP
END
*
* Validate Dictionary
DR.REC = XLATE('DICT ':FILE.NAME, DICT.NAME, -1, 'X')
IF DR.REC = '' THEN
MSG = 'Unable to read dictionary ':DICT.NAME:' for file ':FILE.NAME
GOSUB SHOW.MESSAGE
STOP
END ELSE
IF DR.REC<6> # 'S' THEN
MSG = 'Warning, ':DICT.NAME:' is not a single valued field.'
MSG := ' Results may be unexpected.'
GOSUB SHOW.MESSAGE
END
END
*
* Complete defaulting behavior
*
IF SOURCE.LIST = '' THEN
SOURCE.LIST = 0
END
*
IF DEST.LIST = '' THEN DEST.LIST = 0
*
IF NUM(SOURCE.LIST) AND LEN(SOURCE.LIST)=1 AND (SOURCE.LIST # 9) THEN
SOURCE.LIST.NAMED = 0
END
*
IF NUM(DEST.LIST) AND LEN(DEST.LIST)=1 AND (DEST.LIST # 9) THEN DEST.LIST.NAMED = 0
RETURN

```

SHOW.HELP:

Here is the sub for showing the help message. All of the displays are put into the MSG variable and output by the single routine SHOW.MESSAGE. This gives a single point of modification if there are output standards requirements. Otherwise you can more simply use PRINT, CRT or DISPLAY in place of MSG<-1> =.

Help display code listing

```

SHOW.HELP:
MSG<-1> = ''
MSG<-1> = 'LSELECT FILE.NAME WITH Dict_Name IN LIST1 TO LIST2'
MSG<-1> = ''
MSG<-1> = 'Example: SELECT CUSTOMERS SAMPLE 500'
MSG<-1> = ' SAVE.LIST MY.LIST'
MSG<-1> = ' LSELECT SALES.ORDERS WITH CUST.NBR IN MY.LIST TO MY.LIST.2'
MSG<-1> = 'This will create a list of SALES.ORDER keys for records whose'
MSG<-1> = 'CUST.NBR value appears in "MY.LIST".'
MSG<-1> = ''
MSG<-1> = 'If a destination list is not supplied then keys will be left'
MSG := ' as the active select list.'
MSG<-1> = ''

```

```

MSG<-1> = 'If a source list is not supplied then the current active select'
MSG := ' list will be used.'
MSG<-1> = ''
MSG<-1> = 'Lists may be specified as select buffers (0-8) or as named'
MSG := ' lists from SAVEDLISTS.'
MSG<-1> = ''
GOSUB SHOW.MESSAGE
RETURN

```

GET.SOURCE.SELECT.LIST:

I've chosen to support both lists in numerical buffers and named lists from the SAVEDLISTS file. Here is the code to get the list from either source.

Source list retrieval code listing

```

GET.SOURCE.SELECT.LIST:* get the list of values to match
SOURCE.ID.LIST = ''
*
IF SOURCE.LIST.NAMED THEN
GETLIST SOURCE.LIST TO 9 ELSE
MSG = 'Error reading saved list ':SOURCE.LIST
GOSUB SHOW.MESSAGE
STOP
END
READLIST SOURCE.ID.LIST FROM 9 ELSE SOURCE.ID.LIST = ''
END ELSE
READLIST SOURCE.ID.LIST FROM SOURCE.LIST ELSE SOURCE.ID.LIST = ''
END
RETURN

```

CREATE.TEMP.FILE:

If we want our verb to scale well, then a little care on file sizing is in order. We can quickly calculate the size of our id list using the multi-value version of the LEN function LENS, and then SUMming the results.

I've also included a little bit of code to guard against work file name collisions (to support our concurrency requirement) by making the work file name process number dependent; just in case LSELECT is used simultaneously by multiple users.

Create work file code listing

```

CREATE.TEMP.FILE:* create temporary file to support TRANS selection
* determine appropriate file size based on size of list of values to match
TBYTES=SUM(LENS(SOURCE.ID.LIST))
TEMP.MOD = INT(TBYTES/1024)
IF TEMP.MOD < 1 THEN
TEMP.MOD = 5
END
PROCESS.NBR = @UDTNO + 0
PROCESS.ID = ('0000':PROCESS.NBR:@LEVEL) 'R#4'

```

```

TEMP.NAME = 'LSELECT':PROCESS.ID
UDT.COMMAND = 'CREATE.FILE ':TEMP.NAME:' ':TEMP.MOD
PERFORM UDT.COMMAND CAPTURING UDT.RESPONSE
*
OPEN '', TEMP.NAME TO TEMP.HANDLE ELSE
* Message 420:
MSG = 'Error opening ':TEMP.NAME:' file'
GOSUB SHOW.MESSAGE
STOP
END
OPEN 'DICT', TEMP.NAME TO TEMP.DICT ELSE
* Message 420: Error opening %1 file
MSG = 'Error opening DICT ':TEMP.NAME:' file'
GOSUB SHOW.MESSAGE
STOP
END
RETURN

```

LOAD.TEMP.FILE:

After our temp file is created, we need to load it as efficiently as possible. So here I suggest using a REMOVE command to traverse the key list as efficiently as possible. Alternately you could activate the list and use the READNEXT command which also scales quite well. In any case, I would avoid direct attribute number references on potentially large arrays (for example, ID = SOURCE.ID.LIST<ATB.IDX>).

REMOVE does have its quirks. You must be careful to avoid missing the last attribute or value. I am using the SETTING variable simply as a flag to signal the end of the list, but it will go to 0 as the last id is removed. So the WRITEVU must occur above the WHILE DO to make sure this last id is written to the temp file.

Work file loading code listing

```

LOAD.TEMP.FILE:* put selection values into temp file
EMPTY.STRING = ''
MORE.IDS = 1
LOOP
REMOVE ID FROM SOURCE.ID.LIST SETTING MORE.IDS
WRITEVU EMPTY.STRING ON TEMP.HANDLE, ID, 0
WHILE MORE.IDS DO
REPEAT
* Reset internal udt REMOVE pointer
SOURCE.ID.LIST = SOURCE.ID.LIST
RETURN

```

BUILD.RESULT.LIST:

In order to implement the selection process we can use the EVAL function to carry out a TRANSLATE based on the dictionary name that is provided by the user. This alleviates a number of problems related to the creation and subsequent clean-up of

a permanent dictionary entry in the dictionary of our real file.

Also, by using a dictionary-based TRANS statement instead of a hard-coded attribute number technique, the user can take advantage of I-Descriptors to specify the matching values, which greatly generalizes the selection capability.

We just build the TRANS statement as we would an I-Descriptor in a dictionary and then include it in the SELECT statement after the keyword EVAL. The trickiest part is getting a working combination of single and double quotes.

Result List Build code listing

```
BUILD.RESULT.LIST:* create the resulting list
TRANS.STMT = "TRANS('":TEMP.NAME:"',"":DICT.NAME:"",'F0','X')"
IF DEST.LIST.NAMED THEN
UDT.COMMAND = 'SELECT ':FILE.NAME:' WITH EVAL "':TRANS.STMT:'" GT "" '
PERFORM UDT.COMMAND CAPTURING UDT.RESPONSE
PERFORM 'SAVE.LIST ':DEST.LIST
END ELSE
UDT.COMMAND = 'SELECT ':FILE.NAME:' WITH EVAL "':TRANS.STMT
UDT.COMMAND := '" GT "" TO ':DEST.LIST
PERFORM UDT.COMMAND
IF DEST.LIST = 0 THEN
HUSH ON
PERFORM 'SAVE.LIST ':TEMP.NAME
HUSH OFF
END
END
RETURN
```

FINISH.UP:

With our work done, we now want our verb to clean up after itself. We need to delete the temporary file. If we are creating an active list result, we do so using a temporary list, and so there is a little extra code to activate the list and clean up the disk copy.

Housekeeping clean-up code listing

```
FINISH.UP:* remove temporary key list file
CLOSE TEMP.HANDLE
CLOSE TEMP.DICT
UDT.COMMAND = 'DELETE.FILE ':TEMP.NAME
DATA 'Y'
PERFORM UDT.COMMAND CAPTURING UDT.RESPONSE
IF DEST.LIST = 0 THEN
GETLIST TEMP.NAME TO DEST.LIST ELSE
MSG = 'Error deleting temporary saved list ':DEST.LIST
GOSUB SHOW.MESSAGE
STOP
END
HUSH ON
DELETELIST TEMP.NAME
HUSH OFF
END
RETURN
```

SHOW.MESSAGE:

To finish it all off, add a simple display routine for displaying messages.

Message display code listing

```
SHOW.MESSAGE:* display messages
MSG.CNT = DCOUNT(MSG,@AM)
FOR MSG.IDX = 1 TO MSG.CNT
PRINT MSG<MSG.IDX>
NEXT
PRINT 'Press <cr> To Continue':
INPUT DUMMY
RETURN
```

Conclusion

The UniData SELECT processor provides great support for accessing information based on primary key lists. LSELECT lets you go farther selecting based on foreign keys or even non-key dictionary matches to lists. Whether you assemble these parts with your own improvements or use the accompanying download source file, LSELECT will provide you with easier access to a broader range of information sets.

Downloads

Description	Name	Size	Download method
Completed UniBasic source code for this article ¹	LSELECT.txt	7KB	HTTP

[Information about download methods](#)

Note

1. Save this file as an item in your UniData account's BP file. Make sure to remove the ".txt" extension. You will need to compile and catalog this program to make it operational.

This code was tested against UniData 6.1 with ECLTYPE=U. It has NOT been tested on UniVerse or other MV environments by the author, but will probably require at least some additional development to cope with the lack of EVAL functionality.

It will not work as-is on Manage 2000 installations due to conflicts with the Manage 2000 toolset. A 'Toolized' version named RSELECT is included as a standard feature of Manage 2000 in release 7.2sp2 and above.

Resources

Learn

- In the [U2 page on developerWorks](#), read technical articles and tutorials, and link to the resources you need to advance your skills on U2.
- Browse the [technology bookstore](#) for books on these and other technical topics.

Get products and technologies

- Download [trial versions of U2](#).
- Download [IBM product evaluation versions](#) or [explore the online trials in the IBM SOA Sandbox](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- [Participate in the discussion forum for this content.](#)
- Check out the [Author's blog](#).
- Check out [developerWorks blogs](#) and get involved in the [developerWorks community](#).

About the author

Peter M. Newby

Peter Newby is a graduate of Drury University with a double major in Philosophy and Mathematics. He worked in computer operations and programming for Northrup-King Seed Co., Minnesota Toro Inc., and ROI Systems Inc. before joining EPICOR. Peter has over 25 years of experience working in PICK, Microdata, PRIME, and UniData environments.