AD-A057 320 WHARTON SCHOOL PHILADELPHIA PA DEPT OF DECISION SCIENCES F/G 9/2
DBLISP: A SEED/LISP INTERFACE USER'S MANUAL,(U)
FEB 78 R M LEE N00014-75-C-0462
UNCLASSIFIED 78-02-04 NL

| OF |
ADA
057320

END
DATE
FILMED
9-78
DDC

Wharton

Department of Decision Sciences

LEVEL II

University of
Pennsylvania
Philadelphia PA 19104

LEVEL II

(6) DBLISP: A SEED/LISP INTERFACE USER'S MANUAL (17)

(10) Ronald M. Lee

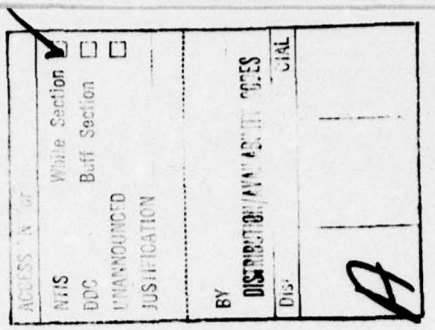
(14) 78-02-04

(11) February 1978

ABSTRACT: DBLISP consists of a set of functions which provide a direct interface between the SEED network database system (version B04) and UCI LISP. Described here are instructions for using DBLISP, a sample session and various technical comments.

(15) Contract N0014-75-C-0462

(12) 116p.



I. INTRODUCTION

Described herein is a package of functions, collectively called DBLISP, which serves to interface the SEED network data base system to UCI LISP. It may be used interactively, from top level LISP, or may be incoporated within user written LISP functions.

II. INSTRUCTIONS

A. The DBLISP Library

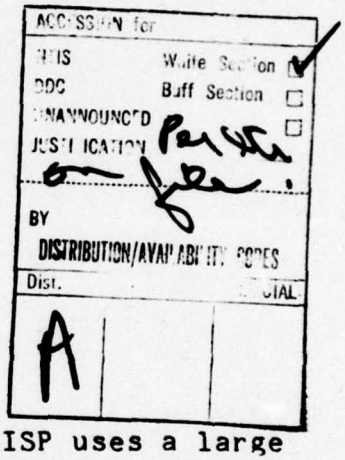
The functions used by DBLISP reside in the following files, all to be found in [4210,1].

.FILE NAME	COMMENTS
DBLMAC.REL	MACRO routines used by DBLISP. Source = DBLMAC.MAC
DBLFOR.REL	FORTRAN routines used by DBLISP. Source = DBLFOR.F4
DBLDML.REL	The SEED DML routines, excluding those FORTRAN or COBOL functions which do I/O (see the appendix for further comments).
DBLCOR.REL	Contains the DBGETC, replacing the SEED routine routine of the same name. This defines a LISP array which holds the database schema (rather than putting it in expanded core as SEED normally does). Source = DBLCOR.F4
DBLISP.LSP	LISP routines used by DBLISP.

B. Setting-Up

A LISP session utilizing DBLISP with the full SEED system requires the following set-up procedure:

```
SET VIRTUAL LIMIT 0

R LISP 60

FULL WORD SPACE = 0

BIN.  PROG.  SP.  = 36000

REG.  PDL.  = 0

SPEC.  PDL.  = 0

(LOAD  T)DBLMAC.REL[4210,1],DBLFOR.REL[4210,1],DBLDML.REL[4210,1],
DBLCOR.REL[4210,1]$

(SETQ BASE (PLUS 5 5))

(SETQ IBASE (PLUS 5 5))

(DSKIN (2184 1) (DBLISP.LSP))

(DBLINIT)
```



Several comments are in order.

Second, we must carefully caution the user that DBLISP uses a large amount of core memory. This is due primarily to the fact that DBLISP loads the entire SEED DML. Aggravating this problem is the fact that the recently introduced virtual memory facility (on the Wharton DECsystem 10) is not able to handle the core image created by LISP if the LOAD command is employed. Hence, the command 'SET VIRTUAL LIMIT 0' to shut off the VMH.

This presents a rather severe limitation since the size of the DBLISP image frequently exceeds the daytime maximum physical core allotments. A procedure for working around this problem is described in Section G of Part III. In Section E of Part III is also described how to use only part of the DML if not all of the SEED features are

required. A method for computing the amount of physical core required is also given in Section F of that part.

C. Use

Three principle functions are used for the interactions with the SEED system. The syntax of these is as follows:

(PUTUWA "itemname" value)

Puts 'value' in the UWA field 'itemname' and returns 'value'.

(GETUWA "itemname")

Returns the value of the UWA field, 'itemname'.

(DB: "dmlfunc" "arg1" "arg2" "arg3" "arg4")

Invokes the SEED DML routine 'dmlfunc' with up to four arguments. (Less than four arguments are acceptable, as per the syntax of the DML routine.) The arguments are either integer numbers (FIXNUM's) or characters. If character, they may be either a literal atom or a string (within double quotes). The user is referred to the SEED REFERENCE MANUAL, under the section "FORTRAN Data Manipulation Language" for the specifics of the various DML routines.

Exhibit 1 may help to clarify the use of these three functions. Because of incompatibilities in the data representation between LISP and the FORTRAN of the SEED system, an intermediate buffer called the UWA ("user work area") has been predefined (see Section B, Part III). While the SEED routines can access this directly, LISP must use the routines PUTUWA and GETUWA to store and retrieve data from the UWA, respectively. Likewise, since the FORTRAN subroutines of SEED aren't callable directly from LISP, the function "DB:" serves to convey the DML call to SEED.

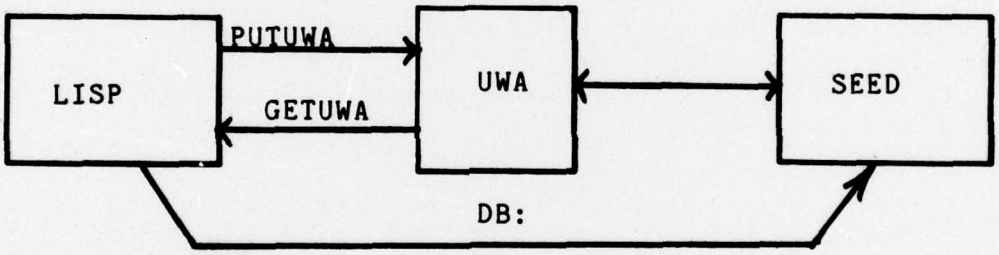


EXHIBIT 1: The Principle DBLISP Functions

It may seem curious that the functions PUTUWA and GETUWA can refer to the UWA by item name without so far having mentioned any particular data base. These two functions make use of various SEED routines which in turn access the schema from which the data base specific structure of the UWA is derived. Neither PUTUWWA nor GETUWA can be used, therefore, until a DBOPEN has been performed to identify the data base. The DBOPEN must specify a FORTRAN sub-schema name.

A sample DBLISP session is shown in Exhibit 2. A portion of a test data base with a FORTRAN sub-schema, "LSPDBF", is used, which contains a simple hierarchy of DEPT and STUDNT record types. In this example, the DBLISP functions are being used interactively from top level LISP (in much the same fashion as with DBLOOK). They can likewise be used from within other user-defined LISP functions as well.

```
.SET VIRTUAL LIMIT 0
.R LISP 60
ALLOC? (Y OR N) Y
FULL WORD SP. = 0
BIN. PROG. SP. = 36000
REG. PDL. = 0
SPEC. PDL. = 0

UCI LISP

*(LOAD T)DBLMAC.REL[4210,1],DBLFOR.REL[4210,1],DBLDML.REL[4210,1],
DBLCOR.REL[4210,1]$
LOADER 17K CORE

*(SETQ BASE (PLUS 5 5))
10.
*(SETQ IBASE (PLUS 5 5))
10.
*(DSKIN (2184 1) (DBLISP.LSP))
NIL
*(DBLINIT)
15667.
15679.
15689.
NIL

*(DB: DBOPEN TSTDBF 0 1)
0.
(PUTUWA DEPNAM @CIS)
CIS
*(DB: STORE DEPT)
0.
*(PUTUWA DEPNAM @"DECSCI")
"DECSCI"
*(DB: FINDC DEPT FIRST)
0.
*(DB: FINDPO FIRST DEPSTU)
0.
*(DB: GET STUDNT)
0.
*(GETUWA STUSSN)
"474-52-4829"
*(GETUWA STUNAM)
"LEER"
*(DB: DBCLOS)
0.
*
```

EXHIBIT 2. SAMPLE DBLISP SESSION

(user typing is underlined)

III. TECHNICAL COMMENTS

A. Interfacing with MACRO and FORTRAN Routines

The capability of directly calling MACRO and FORTRAN routines is provided by the LISP LOAD command, which dynamically loads *.REL files into the LISP LOW SEGMENT. See (2.), Appendix H, for more detail. However, care must be taken to preserve the accumulators which LISP considers "sacred". The convention maintained in DBLISP is that all accumulators are saved whenever control goes from LISP to either a MACRO or FORTRAN routine, and these are likewise restored when LISP is re-entered.

B. Limits on UWA Size

The DBLISP UWA size is pre-set to 500 words. Various attempts were made to try and make this dynamic, based on the UWA size of the specific data base, but without success. The actual size needed for the user's application program may be found by running the SEED routine SCDUMP for the user's sub-schema and looking up the value given for WRK in the second row of the output. (This may also be found as the size of the array DB in the user's file *.WRK.) If the pre-set size of 500 proves to be inadequate, the user can increase this allocation by changing 500 to the value of WRK in the statement:

```
COMMON/DBASE/DB(500)
```

in the subroutine UWA in the file DBLFOR.F4 and re-compiling.

One also needs to change the statement:

(DMLINIT (500))

within the routine DBLINIT in the file DBLISP.LSP, again replacing 500 with the value of WRK.

C. Limits on Schema Size

As indicated earlier, because LISP could not accomodate the dynamic core allocation performed by SEED, a fixed size area is reserved in LISP's binary program space to hold the database schema and buffers. This is set in the routine DBGETC (in the file DBLCOR), to a length of 1000 words. If the user's schema and buffers require more than this, an ERRSTA of 997 will be returned from the DBOPEN. The amount of space actually needed may be found by running the SEED utility program, SCDUMP on the user's sub-schema. From this, the amount of space actually needed is the sum of SSIZ plus four times the maximum value from the column labelled PSIZ. This fixed reservation may be increased by replacing this computed value for 1000 in the statement:

COMMON/EXCORE/EX(1000)

in the subroutine DBGETC in the file DBLCOR.F4 and re-compiling.

D. The DBLISP Library

As indicated in the forgoing text, the files DBLMAC.MAC, DBLFOR.F4, DBLCOR.F4 and DBLISP.LSP represent the various MACRO, FORTRAN and LISP functions which were coded specifically for the DBLISP package.

The file DBLDML.REL, on the other hand, represents a copy of the SEED DML routines (version B04 as of 20-FEB-78). The reason that a copy had to be made was that these are loaded into the LISP LOW SEG. Since the FORTRAN run time system is not available, those FORTRAN (or COBOL) routines which do I/O and thus utilize the HIGH SEG had to be removed. The routines thus deleted were: WANDA, MACROX, DBCSTO, DBDFIN, DBCDEL, DBCGET, DBCMOD, SCDUMP, DBOPOL, COBERR and ERROUT. The fact that a fixed "snapshot" of the SEED DML was made is a disadvantage in that DBLISP doesn't automatically keep pace with the ongoing development of the SEED system.

E. Using Subsets of the SEED DML.

As explained in Section II, a large amount of core is required for DBLISP due to the fact that the entire DBL is loaded. For this reason, we have collected two subsets of the DML which may be used alternatively to the file DBLDML.REL in cases where the user needs only some of the SEED capabilities.

The first of these DML subsets is contained in the file DBLRET.REL and includes the basic retrieval functions of DBOPEN, FINDC, FINDO, FINDPO, FINDAP, GET and DBCLOS. Replacing this in the LOAD command, one may reduce the binary program space to 23000.

The second subset, in the file DBLUDT.REL, adds to the previous list the fuctions STORE, MODIFY and DELETE. Using this, the binary program space needed is 30000.

F. Computing DBLISP Core Requirements

The physical core requirements needed by DBLISP can be computed as follows.

From the monitor CORE command, let P be the maximum limit of physical memory assigned. This is given as a decimal number of pages. Then the largest LISP program that can be run is approximately:

R LISP (P/2).

However, when the LISP loader is invoked, it needs to expand core by another 15K (approximately, this figure varies somewhat with the size of the files being loaded), so that the largest DBLISP that can be run is about:

R LISP (P/2 - 15).

To compute the maximum core allocations that can be used is a bit cumbersome since these are in octal. The LISP high segment takes 15K hence allowing (P/2 - 30)*1000 words for the low segment. Convert this to octal and subtract 17000 (octal), the amount of core unavailable in the low segment, to give the amount of core that can be allocated.

Of this available space, DBLISP uses octal 23000 (DBLRET), 30000 (DBLUDT) or 36000 (DBLDML) words of binary program space, as explained above. Also, another octal 5000 words of free space are used for the functions in DBLISP.LSP. These figures are all approximate, but the computation should be accurate to within a 1000 words or so.

Note: Sometimes it may occur that one is able to run LISP, but insufficient core remains outside of LISP to make the expansion needed for the loader. This will result in one or more error messages of the form:

?CORE AVAILABLE, BUT NOT TO YOU FILE <file name>

?MORE CORE NEEDED FILE <file name>

When this occurs, it is necessary to reduce n in the command 'R LISP n' to leave more room for the loader.

G. Circumventing the Virtual Memory Problem

As explained in Section II, another serious limitation with DBLISP is the fact that the virtual memory is unable to handle core images where the LISP LOAD command has been invoked. The reason for this remains a mystery, but thanks to a discovery by Jerry Kaplan of the Moore School, a way around this is available.

The basic procedure is to load the DML files into LISP using only physical core. Then break from LISP, change the core limits to include virtual memory, and re-enter LISP. Using the DML subset DMLRET.REL, this would be done as shown in Exhibit 3.

```
.SET VIRTUAL LIMIT 0
.R LISP 38
ALLOC? (Y OR N) Y

FULL WORD SP. = 0
BIN. PROG. SP. = 23000
REG. PDL. = 400
SPEC. PDL. = 400

UCI LISP
*(LOAD T)DBLMAC.REL[4210,1],DBLFOR.REL[4210,1],DBLRET.REL[4210,1],
DBLCOR.REL[4210,1]$
LOADER 11K CORE

*(SETQ BASE (PLUS 5 5))
10.
*(SETQ IBASE (PLUS 5 5))
10.

*(DSKIN (2184 1) (DBLISP.LSP))
NIL

*(DBLINIT)
10075.
10087.
10097.
NIL

*(GCGAG T)
NIL
*(GC)
2206 FREE STG,632 FULL WORDS AVAILABLE
NIL

*^C

.SET VIRTUAL LIMIT 256
.CORE 142
.START

ALLOC? (Y OR N) N

FREE STG EXHAUSTED
FULL WORD SPACE EXHAUSTED
275307 FREE STG,15632 FULL WORDS AVAILABLE

*(DB: DBOPEN etc.)
```

EXHIBIT 3. CIRCUMVENTING THE VIRTUAL MEMORY PROBLEM
(user typing underlined)

Several comments. The command (GCGAG T) causes the LISP garbage collector to print the amount of full words and free space it recovers. The command (GC), which forces a garbage collect, is not necessary but is included here to show the free and full word space originally. On re-entry to LISP, the garbage collector will collect into virtual memory and print the new amount available (in octal).

Once the user has broken from LISP, he/she sets the amount of space desired in the command:

.core n

where 'n' corresponds to the figure the user would otherwise have used in 'R LISP n'. In this example we have used the maximum amount of 142. For most applications, however, this would be much lower.

Upon re-entry, the user may change the allocations if desired. One note of caution. As explained in the LISP manual, the user may again break, increase the core, start and re-allocate. However, this may <u>not</u> be done once a DBOPEN has been performed. The reason for this is that LISP, in doing the re-allocation, also resets all unused channels. In doing so, it resets the channels then in use by SEED, resulting in the error:

?IO to unassigned channel at user PC 004763

as explained more fully in the following section.

H. Channel Allocation

LISP assigns I/O channels sequentally beginning at 0 moving up to a maximum of 15. Those channels in use are recorded in an internal table not available to the user.

SEED on the other hand assigns channels starting at 15 and moving downwards. The channels used by SEED are not included in LISP's table of channels in use. (Hence the problem noted in the previous section).

LISP (apparently) does not re-use lower number channels which again become free (until it reaches 15, when it resets them). This means that the total number of different files that may be accessed when using DBLISP is 16. This includes the four files of the LOAD command, DBLISP.LSP, the database schema, and each of the database areas. If the user has an INIT.LSP file, this too must be included in the count.

REFERENCES

(1.) SEED REFERENCE MANUAL (Version C00; B04 Draft) by Rob Gerritsen, International Data Base Systems, Inc., Philadelphia, PA. c. 1977.

(2.) STANFORD LISP 1.6 MANUAL by Lynn H. Quam and Whitfield Diffie; Stanford Artificial Intelligence Labratory, Stanford, CA. c. 1973.

(3.) UCI LISP MANUAL by Robert J. Bobrow, Richard R. Burton, Jeffrey M. Jacobs and Daryle Lewis; University of California at Irvine, Irvine, CA. c. 1974.