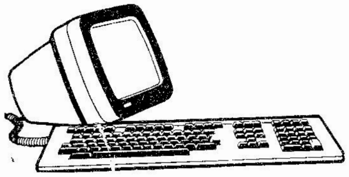
BASIC SIG DECUS U.S. CHAPTER

BASIC SIG

THE NEWSLETTER FOR THE BASIC SPECIAL INTEREST GROUP

April 1983 Vol. 4 No. 1 (Part 2)



THE NEW BASIC FOR THE DEC PROFESSIONAL 350

By Artie Alvidrez, Software Project Leader, Ross Systems, Inc.

Ross Systems, Inc., a California-based timesharing and Software producing company, was selected in 1982 as one of the "software producers" for the new PROFESSIONAL 350 microcomputer developed by DEC. The opportunity to put a major software application on this new system provided the company with one of the first exposures to the new BASIC language, and it proved to be quite an experience for us.

Our application involved a financial-based decision support system called MAPS (Management Aid for Planning Strategies), a software tool for the PDP-11 and VAX computers which has been in use since 1975. The package consists of a number of separate tasks written in both BASIC+2 and PASCAL. Development of the software to run on the PRO 350 called for the compilation and task building of the source on the VAX using DEC's PROFESSIONAL TOOL-KIT, downloading to the PRO via the unique communications interface provided for the PRO, to eventual running and testing on the micro. It was a time consuming cycle, enhanced by the availability of a new type of BASIC compiler for development called TOOL-KIT BASIC. This new BASIC appeared to us to be an interesting merging of BASIC+2 for the PDP-11 and VAX-11 BASIC. As far as we could tell, this TOOL-KIT BASIC contained the best of both worlds and much more. It allowed for the declaring of specific data types like VAX-11 BASIC, and included all built in functions of both types of BASIC. Like the VAX, it allows the use of equivalent-like functions such as the string functions POS and INSTR as well as SEG$ along with LEFT, MID, and RIGHT. But included in this BASIC was the availability of LABELS, the use of line continuations without the need for ampersand-backslash combinations, use of case-statements and other constructs such as the OTHERWISE, END-IF, and ITERATE. Built-in to TOOL-KIT BASIC is the availability of RMS-11 for handling all file I/O.

Even though we developed our application on the VAX, the TOOL-KIT compiler commands looked just like BASIC+2 on the PDP-11 rather than DCL. Modules were fetched using the OLD command, and then compiled with switches like /DOUBLE, /WORD, or /DEBUG. Compile time was very slow, much slower than it would take on a PDP-11. Since both the PRO 350 hardware and TOOL-KIT software was in field-test stage, we discovered errors in the compiler which made development a little more frustrating than we would have liked. The support team at DIGITAL suggested that we use the /MACRO switch for our basic compiles in order to generate a MACRO assembly source which could then be re-assembled into an object module using the MACRO/RSX command in DCL. In order to create a task, the TOOL-KIT included an RSX-11 Task Builder which necessitated the use of .CMD and .ODL files for the creation of a task. The .TSK file could then be downloaded to the PRO and finally tested. If errors were found here, the cycle had to be repeated. Happily, TOOL-KIT BASIC includes a BASIC+2-like debugger, but if your task image exceeded 24K you were out of luck.

After months of development, we finally succeeded in producing the first major application for the PROFESSIONAL 350, a menu-driven financial modeling tool for use in the business environment called MAPS/Pro.

It is assumed that all articles submitted to the editor of this newsletter are with the authors' permission to publish in any DECUS publication. The articles are the responsibility of the authors and, therefore, DECUS, Digital Equipment Corporation, and the editor assume no responsibility or liability for articles or information appearing in the document. The views herein expressed are those of the authors and do not necessarily express the views of DECUS or Digital Equipment Corporation.

STATIC INITALIZATION

OF

BASIC PROGRAMS

- or -

HOW TO SAVE PROGRAM SPACE

HOW TO SAVE EXECUTION TIME

By

Joe Mulvey

BASIC Language Development Manager

for RTL & SML

- CONCEPTS OF DYNAMIC AND STATIC INITIALIZATION

- HOW IT WORKS: ADVANTAGES OF STATIC INITIALIZATION

- HOW TO DO IT YOURSELF

- SUMMARY

"DYNAMIC" OR "RUN-TIME INITIALIZATION"

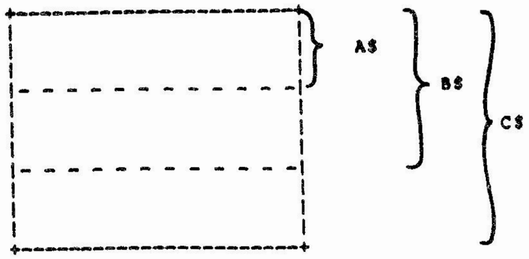
DEFINE VALUES, DATA STRUCTURES
IN PROGRAM BY ARITHMETIC
OR STRING ASSIGNMENT EXECUTION

```
                  RUNTIME                            STATIC
                INITIALIZED                        INITIALIZED

          +-------------------+              +-----------------+
          | DIM ...           |              | DIM ...         |
Tables    | COMMON/MAP ...    |              | COMMON/MAP ...  | Tables
          | DECLARE ...       |              | DECLARE ...     |
          |- - - - - - - - - -|              |- - - - - - - - -|
Init      | READ ...          |              |                 |
Code      | assignments       |              |                 | Program
          |- - - - - - - - - -|   ----->     |                 |
          |                   |              |                 |
Program   |                   |              |- - - - - - - - -|
          |                   |              |                 |
          |- - - - - - - - - -|              |                 | SAVED
DATA      |                   |              |                 | SPACE
Stmts     |                   |              |                 |
          +-------------------+              +-----------------+

          o  ALL BASIC                       o  SOME MACRO/BLISS

          o  COSTS SPACE, TIME               o  SAVES SPACE, TIME
                                                -----
```

PROGRAM SECTION
(PSECT)

- o NAMED SECTION OF TASK MEMORY
- o POSSESSES ATTRIBUTES RECOGNIZED BY LINKER
- o CONTENTS OF PSECT "CONTRIBUTED" FROM .OBJ'S AS THEY ARE ENCOUNTERED BY LINKER

PSECT "X" :



ATTRIBUTES COME FROM:

1Ø MAP (X) A$ = 12	PROG1.BAS
2Ø MAP (X) B$ = 24	PROG2.BAS
3Ø MAP (X) C$ = 36	PROG3.BAS

PSECT ATTRIBUTES:

- o USED BY LINKERS
- o LOCAL/GLOBAL
- o ABSOLUTE/RELOCATABLE
- o READ-WRITE/READ-ONLY

BASIC (AND OTHER LANGUAGES) DEFINES

DATA STRUCTURES USING PSECTS WITH ATTRIBUTES:

GLOBAL

RELOCATABLE

READ-WRITE

OVERLAY (IMPORTANT!)

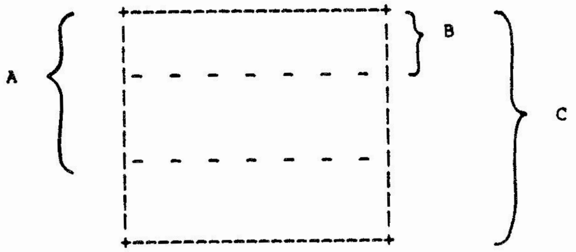
DATA

OVERLAY ATTRIBUTE:

FIRST OCCURRENCE OF PSECT DECLARATION
IN EACH .OBJ CAUSES ALLOCATION TO START
AT PSECT RELATIVE ADDRESS 0

```
         MODULE A:                  MODULE B:              MODULE C:

         .PSECT FOO   OVR,...       .PSECT FOO   OVR,...   .PSECT FOO   OVR,...
    A:   ...                   B1:  ...                C:  ...
```

WHEN MODULES A,B, AND C ABOVE ARE
ASSEMBLED AND LINKED TOGETHER,
LABELS A, B1, AND C REFERENCE THE
SAME LOCATION IN TASK MEMORY



PSECT NAME MUST CORRESPOND TO
COMMON OR MAP NAME
IN THE BP2 PROGRAM.

```
MAP (FOO) ...       ==>     .PSECT   FOO  RW,D,GBL,REL,OVR
```

PSECT MUST BE DEFINED
APPROPRIATELY IN TASK OVERLAY
STRUCTURE (.ODL) TO BE
ASSOCIATED WITH
BP2 MODULES THAT
REFERENCE SAME PSECT

CORRESPONDENCE OF BASIC DATA TYPES AND MACRO DECLARATIONS

INTEGER DATA TYPES

TYPE	MACRO DIRECTIVE	SYSTEM SPECIFIC
BYTE	.BYTE <value>	
WORD	.WORD <value>	
LONG (2 WORD)	.LONG <value> .WORD <low-order value> .WORD <high-order value>	VAX-11 MACRO-32 PDP-11 MACRO-11

CORRESPONDENCE OF BASIC DATA TYPES AND MACRO DECLARATIONS

FLOATING POINT DATA TYPES

TYPE	MACRO DIRECTIVE	SYSTEM SPECIFIC
SINGLE	.FLOAT <value> .FLT2 <value>	VAX-11 MACRO-32 PDP-11 MACRO-11
DOUBLE	.DOUBLE <value> .FLT4 <value>	VAX-11 MACRO-32 PDP-11 MACRO-11
G-FLOATING (VAX ONLY)	.G_FLOATING <value>	
H-FLOATING (VAX ONLY)	.H_FLOATING <value>	

CORRESPONDENCE OF BASIC DATA TYPES AND MACRO DECLARATIONS

MISCELLANEOUS DATA TYPES

TYPE	MACRO DIRECTIVE
PACKED DECIMAL (VAX ONLY)	.PACKED <value>,<symbol>
STATIC STRING	.ASCII /string data/ .BLKB <# of characters>

SUMMARY

- SAVE PROGRAM SPACE
  - NO DATA STATEMENTS
  - NO "TEMPORARY" STORAGE
- SAVE EXECUTION TIME
  - NO VARIABLE ASSIGNMENT
  - NO READ STATEMENT

SUMMARY

- USE MACRO/BLISS + LINKER
  - CAN BE AUTOMATED
  - REQUIRES SOME EXPERTISE
  - CAN USE BP2 BUILD COMMAND

BP2BLD for BASIC-PLUS-2:

- o Has expanded and explanatory BP2BLD dialogue
- o Allows you to take a default installation
- o Provides on-line HELP in response to a ?
- o Summarizes the options you selected
- o Allows you to change answers during the dialogue
- o Generates command file of selected options automatically
- o Installs specific BP2 utilities
- o Updates BASIC-PLUS-2

BP2BLD for version 1.6

```
+-----------------------------------------------------------+
!                                                           !
!  Basic Plus Two Build Version 01.60                       !
!  Input device <MM0:> MM1:                                 !
!  CCL/MCR Name <BP2> EIS                                   !
!  Default HISEG/LIBR for BUILD <EISCOM> NONE               !
!  Specify Location of Disk Library <NO>                    !
!  Use BP2 Resident Library <NO> YES                        !
!           Absolute Address - BP2 Library <???> 352        !
!           Specify Location BP2 Res Lib <NO>               !
!           .                                               !
!           .                                               !
!           .                                               !
!  Build BP2 Utilities <NO>                                 !
!  Specify Location of Utilities <NO>                       !
!  Customize only ? <NO>                                    !
!                                                           !
+-----------------------------------------------------------+
```

BP2BLD for Version 2

```
+-------------------------------------------------------------------+
!                                                                   !
!  What device is the distribution medium mounted on <MM0:> [S]:    !
!           .                                                       !
!           .                                                       !
!           .                                                       !
!  Do you want the default installation <YES>                       !
!  What name do you want to use to invoke BP2 <BP2>                 !
!  Here is a summary of the options you have selected:              !
!           .                                                       !
!           .                                                       !
!           .                                                       !
!  Do you wish to change any of your answers <NO>                   !
!  The BP2BLD dialogue is complete.                                 !
!  The installation will take about an 1 hour to complete.          !
!                                                                   !
+-------------------------------------------------------------------+
```

NEW BP2 INSTALLATION FEATURES

New Resident Libraries	BP2RES and BP2SML: One, both, or none
Optional Run Support	RUN, LOAD, and Immediate Mode
New Choices	Link Run support with BP2RES? Default for resident library? Install Resequencer? Install Dump Analyzer? (RSTS) Device and account for compiler work files?
New Defaults	Data tyce Data type size CROSS_REFERENCE: KEYWORDS SYNTAX_CHECK FLAG: DECLINING Listing page length and width

OLD BP2BLD

```
Use BP2 Resident Library <NO>
```

NEW BP2BLD

```
Do you want to install BP2RES <NO>
Do you want to install BP2SML <NO>
Which BP2 resident library do you want as the default <NONE>
```

OLD BP2BLD

```
Build BP2 Utilities <NO>
Specify Location of Utilities <NO>
```

NEW BP2BLD

```
Do you want to install the BP2 Resequencer  <NO>
Enter the device and account for the BP2 Resequencer <LB:[1,54]>
Do you want to install the BP2 Dump Analyzer  <NO>
Enter the device and account for the BP2 Dump Analyzer <SY:>
```

For All Systems and Specific Systems:

```
+---------+                                +---------------+           +-------+
|         |        +-----------+           |               |           |       |
|         |  ===>  | BASIC on  |  ===>     |               |           |   P   |
|         |        | VAX/VMS   |           |               |           |   o   |
|         |        +-----------+           |               |           |   c   |
| BASIC   |                                |   BASIC       |           |   k   |
|         |        +-----------+           |               |           |   e   |
| User's  |        | BASIC on  |           |   Reference   |           |   t   |
|         |  ===>  | RSX-11M/  |  ===>     |               |   ===>    |       |
| Guide   |        | M-PLUS    |           |   Manual      |           |   G   |
|         |        +-----------+           |               |           |   u   |
|         |                                |               |           |   i   |
|         |        +-----------+           |               |           |   d   |
|         |  ===>  | BASIC on  |  ===>     |               |           |   e   |
|         |        | RSTS/E    |           |               |           |       |
|         |        +-----------+           |               |           |       |
+---------+                                +---------------+           +-------+
```

The BASIC User's Guide contains language usage information common to all three systems:

- o Elements of a BASIC Program
- o Simple I/O and RMS Files
- o Program Control
- o Data Definition
- o Functions and Arrays
- o Formatting Output with PRINT USING
- o Compiler Directives
- o Handling Run-Time Errors
- o Reserved Keywords and Coding Conventions

BASIC on VAX/VMS Systems includes information on:

- o Getting started and simple DCL commands
- o Compiler commands and qualifiers
- o DCL commands and qualifiers
- o Creating and using subprograms
- o Using the VAX-11 Symbolic Debugger
- o Using libraries and shareable images
- o Using system services and RTL routines
- o Using new VAX-11 BASIC features
- o Compile-time and run-time error messages
- o ASCII codes and data definition

BASIC on RSX-11M/M-PLUS Systems and BASIC on RSTS/E Systems include information on:

- o Getting started and simple commands
- o Compiler commands and qualifiers
- o Device-specific I/O
- o Program segmentation and optimization
- o Using the BASIC-PLUS-2 Debugger
- o Using libraries and BASIC-PLUS-2 utilities
- o Compile-time and run-time error messages
- o ASCII codes and data definition

The BASIC Reference Manual describes:

- o Program elements and structure
- o Compiler commands and directives
- o Statements and functions
- o BASIC-PLUS-2 debugger commands
- o Reserved Keywords and Coding Conventions

Format (old)

```
+-----------------------------------------------------------------------------------+
|                                                                                   |
|                                  { num-vbl                                } [...] |
|                                  { str-vbl[=int-exp]                      } [...] |
| COM[MON] [(com-nam)] [data-type] { num-arr(num-cnst[,num-cnst])           } [...] |
|                                  { str-arr(num-cnst[,num-cnst])[=num-cnst]} [...] |
|                                  { FILL-item                              } [...] |
|                                                                                   |
+-----------------------------------------------------------------------------------+
```

Record Structures in VAX-11 BASIC V2

By

Tom Benson

- RECORD templates are defined by the RECORD statement.
- A RECORD name can be used wherever a BASIC data-type keyword is valid.
- RECORD data structures can be composed of variables of any valid BASIC data type or other RECORD types.

```
RECORD record-name
   {data-type  component-name,...} ,...
      .
      .
      .
END RECORD [record-name]
```

- record-name is the name of the data structure.
- data-type is a valid BASIC data-type keyword, or another record-name.
- component-name is a variable, array, or FILL item.
- Each line of a record block can have an optional line number

RECORD TEMPLATES allocate no storage; they only define the name as a data structure. Using the TEMPLATE as a data-type in a declarative statement declares a RECORD INSTANCE, for which storage is allocated.

```
100     RECORD EMPLOYEE
           LONG EMP_NUMBER
           STRING FIRST_NAME = 10
           STRING LAST_NAME = 20
        END RECORD EMPLOYEE

1000    DECLARE EMPLOYEE EMP_REC_1, EMP_REC_2
```

Only assignment and comparison (equality and inequality) operations are allowed on entire records. Elementary components may be used as normal BASIC program variables. They are specified by the RECORD instance name and the component name, separated by "::".

```
1500     DECLARE EMP_WAGE_CLASS EMP

2000      INPUT "wage Class"; EMP::WAGE_CLASS
         SELECT EMP::WAGE_CLASS
             CASE "A"
                 INPUT 'Rate';EMP::HOURLY_WAGE
                 INPUT 'Regular pay';EMP::REGULAR_PAY_YTD
                 INPUT 'Overtime pay';EMP::OVERTIME_PAY_YTD
             CASE "B"
                 INPUT 'Salary';EMP::SALARIED::YEARLY_SALARY
                 INPUT 'Pay YTD';EMP::SALARIED::PAY_YTD
             CASE "C"
                 INPUT 'Salary';EMP::EXECUTIVE::YEARLY_SALARY
                 INPUT 'Pay YTD';EMP::EXECUTIVE::PAY_YTD
                 INPUT 'Expenses';EMP::EXPENSES_YTD
         END SELECT
```

Elliptical references:

- The RECORD instance must always be specified.
- Any dimensioned GROUP name must always be specified.
- Any other intermediate component name may be omitted.
- The final component name must be specified.

```
10       RECORD RECTYPE
             GROUP GROUP_1
                 INTEGER A
                 GROUP GROUP_2
                     INTEGER B, C
                 END GROUP GROUP_2
                 GROUP GROUP_3 (10)
                     INTEGER D, E
                 END GROUP GROUP_3
             END GROUP GROUP_2
         END RECORD RECTYPE

         DECLARE RECTYPE REC
         PRINT REC::GROUP1::GROUP2::B, REC::C
         PRINT REC::GROUP1::GROUP3(1%)::D
         PRINT REC::GROUP3(1%)::E
```

```
10 RECORD COMPLEX
        REAL RE
        REAL IM
   END RECORD

20 DEF COMPLEX ADD( COMPLEX OP1, OP2)
   ADD::RE = OP1::RE + OP2::RE
   ADD::IM = OP1::IM + OP2::IM
   END DEF

30 DECLARE COMPLEX A,B,C

40 INPUT "A = ";A::RE,A::IM
   INPUT "B = ";B::RE,B::IM

   C = ADD( A, B)

   PRINT "A+B = "; C::RE; "+"; C::IM; "I'
```

```
RECORD JPI_ITEM_DESCRIPTOR
    WORD BUFFER_LENGTH
    WORD ITEM_CODE
    LONG BUFFER_ADDRESS
    LONG RETURN_LENGTH_ADDRESS
END RECORD JPI_ITEM_DESCRIPTOR

RECORD JPI_ITEM_LIST
    JPI_ITEM_DESCRIPTOR JPI_ITEM(2)
    LONG LIST_TERMINATOR
END RECORD JPI_ITEM_LIST

DECLARE JPI_ITEM_LIST ITEMS
```

```
ITEMS::JPI_ITEM(0)::ITEM_CODE             = JPI$_PRCNAM
ITEMS::JPI_ITEM(0)::BUFFER_LENGTH         = LEN (USER_NAME)
ITEMS::JPI_ITEM(0)::BUFFER_ADDRESS        = LOC (USER_NAME)
ITEMS::JPI_ITEM(0)::RETURN_LENGTH_ADDRESS = LOC (USER_NAME_LENGTH)

ITEMS::JPI_ITEM(1)::ITEM_CODE             = JPI$_ACCOUNT
ITEMS::JPI_ITEM(1)::BUFFER_LENGTH         = LEN (ACCOUNT_NAME)
ITEMS::JPI_ITEM(1)::BUFFER_ADDRESS        = LOC (ACCOUNT_NAME)
ITEMS::JPI_ITEM(1)::RETURN_LENGTH_ADDRESS = LOC (ACCOUNT_LENGTH)
```

```
ITEMS::JPI_ITEM(2)::ITEM_CODE             = JPI$_CPUTIM
ITEMS::JPI_ITEM(2)::BUFFER_LENGTH         = 4
ITEMS::JPI_ITEM(2)::BUFFER_ADDRESS        = LOC (CPU_TIME)
ITEMS::JPI_ITEM(2)::RETURN_LENGTH_ADDRESS = LOC (CPU_TIME_LENGTH)

ITEMS::LIST_TERMINATOR = 0

SYS_STATUS = SYS$GETJPI(,,, ITEMS ,,,)
```

The CDD directory hierarchy can be created and maintained using the CDD Dictionary Management Utility (DMU). It allows you to

- o Create dictionary directories and sub-dictionaries.
- o Delete dictionary directories, sub-dictionaries, and objects.
- o Rename CDD entries.
- o Set the protection of CDD entries.
- o List entries, their attributes, and history lists.
- o Make a backup copy of the dictionary.
- o Copy directories within the dictionary.

The Data Definition Language Utility (CDDL) allows you to enter record definitions and new dictionary directories into the CDD.

- o Create a CDDL source file containing the record definition, using an editor.
- o Invoke CDDL to insert the definition into the dictionary.

```
DEFINE RECORD path-name

    [DESCRIPTION [IS] /* text */].

     field-description-statement

END [ {path-name }] [RECORD].
    [ {given-name}]
```

```
DEFINE RECORD CDD$TOP.CORPORATE.ADDRESS_REC
    DESCRIPTION IS
        /* Contains standard format for addresses */.
    ADDRESS STRUCTURE.
        STREET              DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
        CITY                DATATYPE TEXT   SIZE 30.
        STATE               DATATYPE TEXT   SIZE 2.
        ZIP_CODE STRUCTURE.
            NEW             DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 4 DIGITS
                            BLANK WHEN ZERO.
            OLD             DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 5 DIGITS.
        END ZIP_CODE STRUCTURE.
    END ADDRESS STRUCTURE.
END ADDRESS_REC.
```

To access a CDD record definition from VAX-11 BASIC, use

```
        %INCLUDE %FROM %CDD cdd-path-name
```

For example,

```
  %INCLUDE %FROM %CDD "CDD$TOP.PERSONNEL.SERVICE.SALARY_REC"
```

Or, if CDD$DEFAULT = CDD$TOP.PERSONNEL,

```
  %INCLUDE %FROM %CDD "SERVICE.SALARY_REC"
```

- /SHOW:([NO]CDD_DEFINITIONS)
  - specifies whether or not to list record definitions extracted from the CDD in the listing file.

- /[NO]AUDIT[:{str-lit}]
  [:{file-spec}]
  - specifies whether or not to log audit entries in the CDD for record definitions extracted from it.

In addition, to the str-lit or file-spec you specify, BASIC includes the following information in audit entries:

- The access was in a BASIC program
- The access was an extraction (COMPILE)
- The name of the program module that requested the extraction and the date and time of the request.

```
DEFINE RECORD basicdef
   DESCRIPTION IS
           /* This is an example record containing */
           /* data-types native to VAX-11 BASIC */.
   employee STRUCTURE.
           street          DATATYPE TEXT   SIZE 30.
           city            DATATYPE TEXT   SIZE 30.
           state           DATATYPE TEXT   SIZE 2.
           zip_code STRUCTURE.
                   new     DATATYPE PACKED NUMERIC  SIZE 4 DIGITS.
                   old     DATATYPE PACKED NUMERIC  SIZE 5 DIGITS.
           END zip_code STRUCTURE.
           emp_number      DATATYPE IS SIGNED WORD.
           wage_class      DATATYPE TEXT   SIZE 2.
           salary_ytd      DATATYPE IS D_FLOATING.
   END employee STRUCTURE.
END basicdef.
```

```
define record cdd$top.basic.integers
    description is
   /* Test of selected integer data-types */.
    basicint structure.

   my_byte          datatype is signed byte.

   my_ubyte         datatype is unsigned byte.

   my_word          datatype is signed word.

   my_uword         datatype is unsigned word.

   my_long          datatype is signed longword.

   my_ulong         datatype is unsigned longword.

    end basicint structure.
end integers.
```

```
      1  1 %include %from %cdd 'integers'
C1       1          !   Test of selected integer data-types
C1       1         RECORD  BASICINT                        ! UNSPECIFIED
C1       1           BYTE    MY_BYTE                       ! SIGNED BYTE
C1       1           GROUP   MY_UBYTE                      ! UNSIGNED BYTE
C1       1             BYTE    BYTE_VALUE
C1       1           END GROUP
C1       1           WORD    MY_WORD                       ! SIGNED WORD
C1       1           GROUP   MY_UWORD                      ! UNSIGNED WORD
C1       1             WORD    WORD_VALUE
C1       1           END GROUP
C1       1           LONG    MY_LONG                       ! SIGNED LONGWORD
C1       1           GROUP   MY_ULONG                      ! UNSIGNED LONGWORD
C1       1             LONG    LONG_VALUE
C1       1           END GROUP
C1       1         END RECORD
```

- If a non-zero SCALE is specified in a CDD definition of a fixed-point (integer) or floating point field, BASIC reports the warning "CDDATTSCA, CDD specifies SCALE for <name>. Not supported".

- If a BASE other than 10 is specified in the definition of an integer or floating point field, BASIC reports the warning "CDDATTBAS, CDD attributes for <name> are other than base 10".

```
define record cdd$top.basic.funnyintegers
    description is
    /* Test of quadword and octaword integer data-types */.
    basicint structure.

   my_byte         datatype is signed byte scale 2.

   my_long         datatype is signed longword base 8.

   my_quad         datatype is signed quadword scale 5.

   my_octa         datatype is signed octaword base 16.

    end basicint structure.
end funnyintegers.
```

```
      1 1 %include %from %cdd 'funnyintegers'
C1    1       !   Test of quadword and octaword integer data-types
C1    1       RECORD  BASICINT                      ! UNSPECIFIED
C1    1         BYTE    MY_BYTE                     ! SIGNED BYTE
C1    1         LONG    MY_LONG                     ! SIGNED LONGWORD
C1    1         GROUP   MY_QUAD                     ! SIGNED QUADWORD
C1    1           STRING  STRING_VALUE  = 8
C1    1         END GROUP
C1    1         GROUP   MY_OCTA                     ! SIGNED OCTAWORD
C1    1           STRING  STRING_VALUE  = 16
C1    1         END GROUP
C1    1       END RECORD
```

- o If a field of type BIT is not a multiple of eight bits in length, BASIC signals the error "CDDBITFLD, field <name> from CDD has bit offset or length".

- o If a definition contains a field of the VIRTUAL data-type, BASIC signals the error "CDDUNSDAT, data type specified in CDD for <name> not supported".

USING USEROPEN IN V2 BASIC

By

Stepnen Reilly

INTRODUCTION

- New OPEN clause features
- What is USEROPEN ?
- How to use USEROPEN
- Useful hints and warnings
- Wrap-up

NEW OPEN CLAUSES

- The new clauses are used with RMS files only

- On RSX-11M and RSX-11M-PLUS
  - Sequential, Relative, Indexed, Virtual
  - Not Terminal format files

NEW OPEN CLAUSES

- On VMS
  - All types of OPENs

- On RSTS/E
  - Sequential, Relative, Indexed
  - Not virtual or terminal format files

NEW OPEN CLAUSES

- In BASIC-PLUS-2

  - The clauses will not affect device specific OPENs

```
10 OPEN "TI:" FOR INPUT AS FILE #1%
```

NEW OPEN CLAUSES

- BUFFER

  - Sequential files multiblock count
  - Indexed and Relative files multibuffer

- EXTENDSIZE (New for BP2 only)
  - Function of the clustersize of the media on RSTS/E

- RECORDTYPE (New for BP2 only)
  - LIST
  - NONE
  - ANY
  - FORTRAN

NEW OPEN CLAUSES

- DEFAULTNAME (New for BP2 only)

```
OPEN "ACCT.DAT" AS FILE #1%, SEQUENTIAL FIXED,
                         DEFAULTNAME "SY:[1,10]TEST"
```

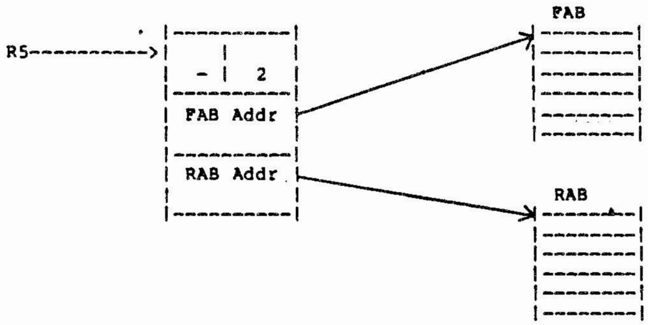
  - Resultant string is SY:[1,10]ACCT.DAT

  - Also useful because the channel is associated with the LUN. If the LUN is assigned to a different device and the file spec does not have an explicit device the OPEN will use the previous LUN assignment. (BP2 only)

HOW TO USE A USEROPEN ROUTINE

```
10      MAP ( BUF ) STRING FILE_BUFFER = 80%

        OPEN "DAT.DAT" AS FILE #1%, SEQUENTIAL VARIABLE, MAP BUF,
                USEROPEN USR

        CLOSE #1%
```

CALLING MECHANISM (-11s)



A SAMPLE USEROPEN ROUTINE FOR THE 11s

```
        .TITLE          USR
;+
;   This routine will link a protection XAB to
;   the end of a linked list of XABs so that the
;   file will be created with a protection code different
;   from the default protection code for the disk it is on.
;
; INPUT:
;   2(R5) - Address to the FAB
;   4(R5) - Address to the RAB
;
; OUTPUT:
;   R0 - the STS field of either the FAB or RAB
```

A SAMPLE USEROPEN ROUTINE FOR THE 11s

```
; EFFECT:
;   The file is created, a connect is done if no errors occured
;
;   R1 - R3 are destroyed.
;
; EXTERNALS:

        .MCALL      $GNCAL,FAB$B,RAB$B,XAB$B,NAM$B,$FBCAL,$RBCAL
        $GNCAL
        $FBCAL
        $RBCAL
;
;-
```

A SAMPLE USEROPEN ROUTINE FOR THE 11s

```
;+
;       Set the protection code for the XAB
;-

PROCOD:
        XAB$B   XB$PRO

.IF     DF      RSX

        X$PRO   60942               ; (R,RWED,R,R)
.IFF
        X$PRO   40                  ; Set protection

.ENDC
        XAB$E
```

A SAMPLE USEROPEN ROUTINE FOR THE 11s

```
USR::   MOV     2(R5),R2            ; Get FAB pointer
;+
;       Walk down through the linked list of XABs ( if any ) and
;       insert the PRO XAB at the end.
;-
        $FETCH  R3,XAB,R2           ; Get the first XAB addr if any
        BEQ     2$                  ; BR if none

1$:     $FETCH  R1,NXT,R3           ; Get the next XAB on the list
        BEQ     3$                  ; If none left BR
        MOV     R1,R3               ; R3 = current XAB address
        BR      1$                  ; Cont until done

2$:     $STORE  #PROCOD,XAB,R2      ; Store our XAB address in the FAB
        BR      4$                  ; Cont

3$:     $STORE  #PROCOD,NXT,R3      ; Store our XAB address in the Last XAB on list
```

A 2ND EXAMPLE OF USEROPEN

```
10      OPEN "ACCT.DAT" FOR INPUT AS FILE #1%, SEQUENTIAL, USEROPEN SPOOL_FILE

        CLOSE #1%                ! Spool the file

        END
```

A 2ND EXAMPLE OF USEROPEN

```
20      FUNCTION LONG SPOOL_FILE ( FAB OUR_FAB, LONG OUR_RAB, LONG CHANNEL )

        RECORD FAB
            STRING FILL = 4
            LONG   FOP
        END RECORD

        EXTERNAL LONG FUNCTION SYS$OPEN         ! open file with FAB
        EXTERNAL LONG FUNCTION SYS$CONNECT      ! connect to file with RAB
        EXTERNAL LONG CONSTANT SS$_NORMAL       ! normal return status
        EXTERNAL LONG CONSTANT FAB$M_SPL
        DECLARE LONG RMS_STATUS
```

A 2ND EXAMPLE OF USEROPEN

```
        OUR_FAB::FOP = OUR_FAB::FOP OR FAB$M_SPL        ! Set the spool bit

        !
        ! open and connect the file

        RMS_STATUS = SYS$OPEN( OUR_FAB )

        IF RMS_STATUS AND SS$_NORMAL
        THEN
            RMS_STATUS = SYS$CONNECT( OUR_RAB )
        END IF

        SPOOL_FILE = RMS_STATUS

        END FUNCTION
```

HINTS

- Don't set the locate mode bit in the ROP field of the RAB
    - Could cause incorrect data
    - Could also cause access violation on VAX
- Don't use the CTX, BKT and UBF with BP2 USEROPEN

- Access Creation and revised data
- Null keys
- File id
- Bucket sizing

WRAP-UP

- Remember the new clauses of the OPEN statement
- FORTRAN argument passing (for the -11's) and the VAX standard passing mechanism.
- Make sure that bit oriented fields for either FAB or RAB are treated as such

HOW TO USE THE REMAP STATEMENT

- All data types are allowed
  - Only strings allowed for FIELD statement
- All variables in the REMAP statement must be defined in the corresponding MAP DYNAMIC statement.

```
10      MAP DYNAMIC ( BUF ) STRING STREET, CITY, LONG ZIP
```

HOW TO USE THE REMAP STATEMENT

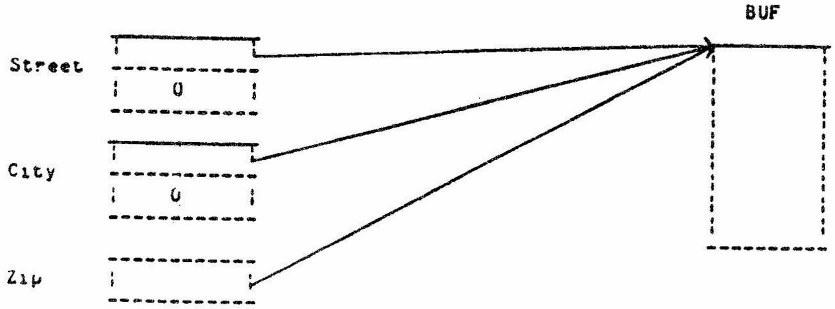
- The MAP DYNAMIC must have a corresponding static MAP

```
10      MAP ( BUF ) STRING FILL = 100%
        MAP DYNAMIC ( BUF ) STRING STREET, CITY, LONG ZIP
```

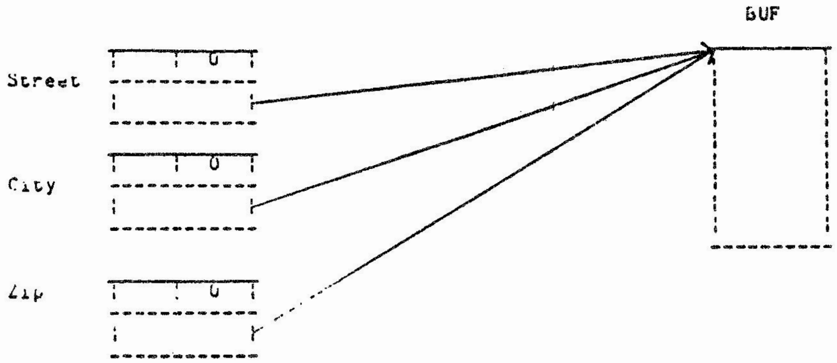
HOW TO USE THE REMAP STATEMENT

- The static MAP must be long enough to handle any element of the MAP DYNAMIC
  - Strings lengths defaulted to zero
  - All numerics are based on their data type length

BASIC-PLUS-2



VAX-11 BASIC



HOW TO USE THE REMAP STATEMENT

- At each invocation of a subprogram (ie. SUB and FUNCTIONs) all REMAP variables are pointing to the beginning of the buffer. (BASIC-PLUS 2 only)
- At each invocation of a subprogram the REMAP variables are NOT re-initialized this is a restriction in V2. (VAX-11 BASIC only)
- Execute the REMAP statement before the a REMAP variable is referenced.

REMAP VS FIELD

- Run with V2 of VAX-11 BASIC2
  - 20 times faster
- Run with V2 of BASIC-PLUS 2
  - 2 times faster (RSTS/E)

REASONS (BP2)

- BP2 OTS
  - REMAP buffer address determined at link time
  - FIELD buffer address determined at run time
- FIELDed variable must be checked
  - See if assigned to dynamic space
    - Deassign space and assign to the buffer

REASONS (VAX BASIC)

- FIELD statement
  - Must be looked up in RTL internal table
- FIELDed variable being assigned
  - Looked up in RTL internal table
    - Used so space associated to I/O buffer not deallocated

- The MAP associated with a REMAP statement
  - Be referenced by a OPEN
  - Just a buffer area
- Descriptors not be allocated if not referenced
  - Common include file

WRAP-UP

- REMAP statement
  - Dissection of a buffer
  - Less run time code
    - No table look-up
    - Buffer address determined at link time
  - No special conversion functions
    - CVT$%

USING BASIC-PLUS-2 V2 FOR THE PROFESSIONAL

By

Stephen Reilly

FEATURES

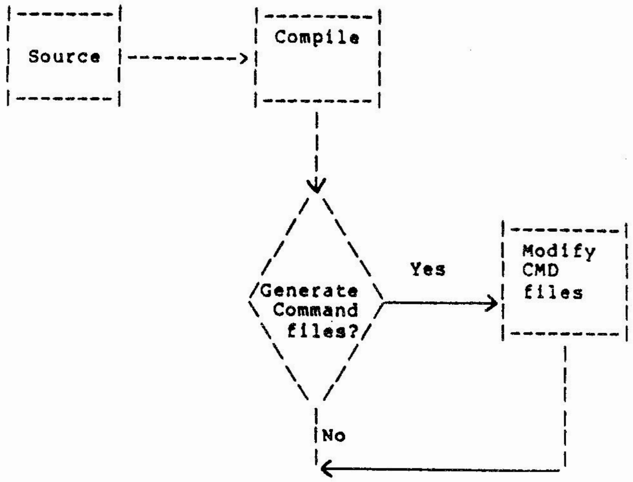
- REDIRECT command
  - New debugger command that will redirect all debugger I/O to the debugging terminal while all program I/O is unaffected
  - Good if developing forms - oriented applications

- CHAIN
  - With BP2 V2, on RSX-11M and RSX-11M+ the program to be chained t no longer needs to be installed
  - On the Professional, all tasks including those chained to must be installed.
- EDIT$
  - EDIT$(- ,1%) should not be used when processing 8 bit character set

- The user task must have RMS
  - Error messages printing on the Professional is done through common routines that require RMS.
  - Error messages will look a little different. Any error printed will be preceded by the error number.
  - After error message is printed out, the user must type the <RESUME> key. This continues program execution for continuable errors and exits applications for non-continuable errors

- The Compiler

    - Does not support the RUN and LOAD commands
    - Does not support immediate mode statements



BUILDING AN APPLICATION

- Produced .CMD file:

```
SY:CT/CP=SY:CT/MP
UNITS = 15
ASG = TI:13:15
ASG = SY:5:6:7:8:9:10:11:12
EXTTSK= 952
CLSTR=PBESML,RMSRES:RO
```

PRODUCED .ODL FILE

```
         .ROOT BASIC2-RMSROT-USER,RMSALL
USER:    .FCTR SY:CT-LIBR
LIBR:    .FCTR LB:[1,5]PBEOTS/LB
@LB:[1,5]PBEIC1
@LB:[1,5]RMSRLX
         .END
```

REQUIRED EDITING TO .CMD FILE

```
SY:CT/CP=SY:CT/MP
UNITS = 19
ASG = TI:13:15
ASG = SY:5:6:7:8:9:10:11:12
EXTTSK= 952
CLSTR=PBESML,POSRES,RMSRES:RO
```

REQUIRED EDITING TO .CMD FILE

```
                        ; DEFINE BUFFER SIZES
EXTSCT  = MN$BUF:0      ; ( 4540 ) static single choice menu
EXTSCT  = DM$BUF:0      ; ( 4540 ) dynamic single choice menu
EXTSCT  = MM$BUF:0      ; ( 1000 ) multi-screen menu
EXTSCT  = HL$BUF:0      ; ( 3410 ) help text/menu
EXTSCT  = MS$BUF:500    ; ( 3100 ) message record buffer
EXTSCT  = FL$BUF:0      ; ( 4310 ) file selection/specification
                        ; for ODFIL and NEWFIL routine
```

WRITING BASIC-PLUS-2 PROGRAMS IN A COBOL-LIKE FORMAT

Bruce K. Snyder and Lori Vanderspool
North Shore Sanitary District
Gurnee, Illinois

ABSTRACT

The primary language of most of the programmers in our area is COBOL thus making it difficult to find a programmer who has a good knowledge of BASIC. Therefore, the District has had to resort to hiring COBOL programmers and training them in BASIC. To reduce the training time, the District has experimented with writing its BASIC programs in a form that is as close as possible to the structure of COBOL.

This paper shows a sample report program written in BASIC-PLUS-2 but which is written in a COBOL-like structure. Thus all the data and report lines will have been pre-defined in a data division. Using this technique, not only has it been easier for beginning programmers to learn BASIC, but there have been other benefits as well. Foremost, programs are easier to maintain. Also, a systematic review process can be incorporated into the programming function.

Finally, data is presented showing that programs written with this technique take no additional CPU time and are roughly the same size as programs conventionally written.

REASONS FOR ADOPTING A NEW PROGRAMMING FORMAT

Even though there are many computers manufactured by Digital Equipment Corporation, it holds true that most of the programmers in the market still have COBOL as their primary language. This makes it hard to find qualified BASIC programmers. In many cases, then, the District has had to hire COBOL programmers and then teach them the syntax of BASIC. The disadvantage of doing this is the lost time in having new programmers learn another language.

Secondly, writing programs using a version of the standard DEC template always resulted in programs that appeared to take "too long" to write. Then, once completed, the programs were very difficult to verify as being correct since the code was hard to understand. This same problem has also periodically made maintaining the programs difficult.

Therefore, in addition to better design and management controls, a better format for programming had to be derived. The above circumstances led to an investigation of using some of the advantages of COBOL as a formatting technique when writing programs in BASIC.

It might be mentioned here that two reasons precluded the District from converting outright to programming in COBOL itself. First, all the previously-written programs in the District had been written in BASIC and it is easier to use only one language if at all possible. Secondly, most of the standard software from Digital is written in BASIC.

DESCRIPTION OF THE EXAMPLE PROGRAM

This section of the paper discusses the program that has been provided as an example for future reference. The discussion is presented in four sections corresponding to the four divisions of a typical COBOL program. Note that the name of each of the four divisions has been highlighted with asterisks. Each subpart of a division, such as a section or paragraph, has been highlighted with equal signs. Column nine has been reserved only for backslashes and comment indicators. Column ten is always blank. The above practices are used to help the readability of the program.

Identification Division

The Identification Division is the first part of a COBOL program. Thus our BASIC program also starts with such a division. Note that in the example program all the lines in this division are comment lines. There are separate lines to place the name of the program, the author, the name of the firm, the dates of the program, and a general description and purpose of the program.

Environment Division

The next division of the program is the Environment Division. The purpose of this division is to detail programming practices that are unique to a particular computer. Note that there are both executable and comment lines in this division. The comment lines list the computers that were used to write and compile the program.

The Special-Names statement is used to associate any special escape sequences to variable names so that if the code were to be transferred to another computer, only the lines in this part of the program would have to be changed. The variable names could remain the same.

Finally, the assignment of files to specific channels is also done in this part of the program.

Data Division

The third division in a COBOL program is called the Data Division. In this part of the program, all the files, record layouts, and other variables used in the program are "mapped out". The Data Division is divided into two sections.

The first section is called the File Section. In this section, the record layouts of all the files, except the print files, are laid out. In the program given here as an example, there are three files that are mapped. Note that the first line of each file is a comment line that starts with "FD", which stands for file description. The rest of the lines for a file are executable MAP and DIM statements. Note that, for the variables that are not strings, the length of the variable is still listed as a comment so that all the numbers in a map can very quickly be added to verify the accuracy of the program.

The second section of the Data Division is called the Working-Storage Section. The miscellaneous variables and accumulators used in the program are first logically grouped. In the example program, there are three such groups: general variables, employee accumulators, and subtotal accumulators. Each group of variables is placed in a map. After each such variable has been mapped, the same variables are assigned initial values through the use of a LET statement. The combination of MAP and LET statements thus have the same effect as the PICTURE statement in a COBOL program. Note also that even though some variables do not have to be assigned initial values, this is done anyway so that every variable format is both explicit and consistent.

After the miscellaneous variables have been mapped and assigned values, the print record layouts are then mapped and assigned values. The sample program has nine detail lines and four summary lines that are needed. Thus each one must be mapped and assigned initial values. Note that there are two differences for the print record layouts. First, each such layout has two maps. The first map is a detailed map showing each segment of a print line. The second map treats the entire print line as one variable. This is necessary since BASIC programs cannot have group fields in the same manner as a COBOL program. The purpose of having a generalized map is to reduce the coding needed whenever a given print line needs to be printed.

The second difference is that all the variables in a print record layout must be string variables. This is helpful in properly aligning the report and makes it easy to code directly from a printer layout chart. Constant variables are assigned values with the LET statements. Fields that should be blank or will have values later assigned to them are at this time assigned a blank status.

Procedure Division

The last division of a COBOL program is the Procedure Division. This division contains the logic of the program. Note that by coding a BASIC program in a COBOL-like format we have greatly reduced the length of the actual logic portion. This is the most important aspect of programming with this technique. All the layouts of files and records as well as the initial assignment of values to variables is coded apart from the logic of the program.

This simple standard has two profound benefits. It first of all enables a lower-level programmer, or a non-programmer, to do the coding of the first three divisions of the prograom directly from a program specification. After these three divisions have been reviewed for consistency with the specification, the program can then be passed along to a more senior-level programmer for the coding of the actual logic. Thus, a shop can therefore better utilize each programmer to the fullest extent of each programmer's abilities.

Secondly, the simple fact of separating the data from the logic insures that no time is wasted in coding logic for the wrong layouts. It also has a profound impact on the ease of coding and therefore the future maintainability of the program.

Note that each paragraph has only one entry point and one exit point except for where a reference is made to a lower-level subroutine (or subprogram for that matter). Except for the GOTO statement that refers the program to the END statement, the only GOTO statements allowed are ones that call the same line number as the GOTO statement itself is on. All of these coding techniques help to make the program easy to read and verify for accuracy.

When it comes time for the program to print a series of lines, the program first formats into the print variables the values from any other variables that need to be printed. Then, to print the series of lines, all that is needed is one print statement with a separate clause referencing either the generalized map for a given print line or the name of a field that was initialized in the Environment Division, for such printer control statements as line feeds and form feeds.

COMPARISON WITH CONVENTIONAL PROGRAMMING TECHNIQUES

When the District first installed its payroll system, the W-2 form printing program was written in the conventional way using a version of DEC's standard template. That version of the W-2 program used 262 CPU seconds and is 15KW in size, excluding the run-time system. The version written in the COBOL style used 271 CPU seconds and is 16KW in size. Thus, there is very little difference. This has held true for similar tests.

But, the important comparison comes in the savings in programmer time. Programs written with this technique can be written in half the time and parts of the code can be written by programmers with less experience. This can thus greatly improve the productivity of the shop.

```
        ! ************************ PROCEDURE DIVISION. ************************ &
        !                                                                         
1000    ! ======================== INITIAL PARAGRAPH. ======================== &
        !                                                                      &
        \ ON ERROR GOTO 19000                                    !                &
        \ OPEN "PR:PRI01.MST" FOR INPUT AS FILE #PM%             ! Open Payroll &
                ,INDEXED VARIABLE                                ! Master File  &
                ,ACCESS READ                                     !              &
                ,ALLOW NONE                                      !              &
                ,MAP PRI01M                                      !              &
        \ OPEN "PR:PRR06.MST" FOR INPUT AS FILE #PC%             ! Open Payroll &
                ,RELATIVE FIXED                                  ! Codes File   &
                ,ACCESS READ                                     !              &
                ,ALLOW NONE                                      !              &
                ,MAP PRR06M                                      !              &
        \ OPEN "PR:PRI10.MST" FOR INPUT AS FILE #HS%             ! Open Payroll &
                ,INDEXED FIXED                                   ! Historical   &
                ,ACCESS READ                                     ! File         &
                ,ALLOW NONE                                      !              &
                ,MAP PRI10M                                      !              &
        \ OPEN "LP:" FOR OUTPUT AS FILE #LP%                     ! Open LP:     &
                                                                 !              &
        \ GET #PC%, RECORD 4%                                    ! Get FICA max &
                                                                 ! limit        &
                                                                 !              &
        \ INPUT "ENTER YEAR ON W-2 FORMS <YY>", GEN.INPUT.YEAR$  ! Enter year   &
        \ PRINT                                                  ! for which    &
                                                                 ! W-2's are    &
                                                                 ! to be run.
1100      INPUT "ARE THE W-2 FORMS LOADED IN LP: <Y/N>", GEN.Q$  ! If forms are &
        \ GOTO 1100 IF LEFT(CVT$$(GEN.Q$,34%),1%) <> "Y"         ! not ready,   &
        \ PRINT                                                  ! ask again.   &
                                                                 !              &
        \ INPUT "IS THIS A RESTART <Y/N>", GEN.Q$                ! If restart   &
        \ IF LEFT(CVT$$(GEN.Q$,34%),1%) = "Y"                    ! do that para &
                THEN GOSUB 2000                                  ! else do      &
                ELSE GOSUB 4000                                  ! regular read &
                                                                 !
1200      GOSUB 3000 UNTIL GEN.EOF$ = "Y"                        ! Continue,    &
        \ GOSUB 5000                                             ! print last   &
        \ GOSUB 7000                                             ! W-2, subto-  &
                                                                 ! tal.
1300      CLOSE I% FOR I% = 1% TO 12%                            ! Close all    &
        \ GOTO 32767                                             ! files and    &
                                                                 ! finish.      &

2000    ! ======================== RESTART PARAGRAPH. ======================== &
                                                                                &
          PRINT                                                  ! Enter last   &
        \ PRINT "ENTER THE SSN APPEARING BEFORE THE LAST ";      ! SSN and      &
        \ INPUT "SUBTOTAL **NO DASHES**", GEN.LAST.SSN$          ! control #    &
        \ PRINT                                                  ! so that we   &
        \ INPUT "ENTER THE CONTROL NO. APPEARING ON THAT W-2",   ! can know     &
                GEN.W2CTRLNO%                                    ! where to     &
        \ GEN.W2CTRLNO% = GEN.W2CTRLNO% + 1%                     ! begin.       &
                                                                 !              &
        \ GET #HS% UNTIL HS.SSN$ = GEN.LAST.SSN$                 ! Find last    &
        \ GET #HS% UNTIL HS.SSN$ <> GEN.LAST.SSN$ AND            ! employee.    &
                (MID$(HS.CK.DATE$,1%,2%) = GEN.INPUT.YEAR$)      ! Go past that &
                                                                 ! employee     &
                                                                 ! until check  &
                                                                 ! year = input &
                                                                 ! year.        &
        \ RETURN                                                 ! Return.      &

3000    ! ====================== ACCUMULATOR PARAGRAPH. ====================== &
                                                                                &
          EMP.EIC        = EMP.EIC        + HS.EMP.EIC           ! Add employee &
        \ EMP.FED.TAX    = EMP.FED.TAX    + HS.FED.TAX           ! totals       &
        \ EMP.GROSS.PAY  = EMP.GROSS.PAY  + HS.GROSS.TOTAL       !              &
        \ EMP.DEF.COMP   = EMP.DEF.COMP   + HS.DEF.COMP          !              &
        \ EMP.FICA       = EMP.FICA       + HS.FICA              !              &
        \ EMP.STATE.TAX  = EMP.STATE.TAX  + HS.STATE.TAX         !              &
        \ EMP.SSN$       = HS.SSN$                               ! Reset ssn.   &
        \ GOSUB 4000                                             ! Read para.   &
        \ IF (CVT$$(EMP.SSN$,2%) <> HS.SSN$) AND                 ! Print W-2 if &
                                  EMP.GROSS.PAY > 0              ! not same emp &
                THEN GOSUB 5000                                  ! unless gross &
                ELSE IF (CVT$$(EMP.SSN$,2%) <> HS.SSN$) AND      ! pay <= 0     &
                                  EMP.GROSS.PAY <= 0             ! then reset   &
                        THEN GOSUB 6000                          ! para         &
                                                                 !
3900      RETURN                                                 ! Return       &

4000    ! ========================= READ PARAGRAPH. ========================= &
                                                                                &
          GET #HS% UNTIL MID$(HS.CK.DATE$,1%,2%) =               ! Get next     &
                                  GEN.INPUT.YEAR$                ! check with   &
                                                                 ! same year.   &
                                                                 !
4900      RETURN                                                 ! Return
```

```
5000    ! ================ EMPLOYEE W-2 FORM PRINT PARAGRAPH. ================ &
                                                                               &
          IF EMP.GROSS.PAY > PC.FICA.MAX                       ! Determine     &
              THEN EMP.FICA.PAY = PC.FICA.MAX                  ! FICA pay.     &
                        ELSE                                   !               &
              EMP.FICA.PAY      = EMP.GROSS.PAY                !               &
                                                               !
5100      DL.01.W2CTRLNO$       = NUM1$(GEN.W2CTRLNO%)         ! Format the    &
        \ DL.04.EIC$            = FORMAT$(EMP.EIC,GEN.F$)      ! print line.   &
        \ DL.05.EMP.SSN$        = LEFT(EMP.SSN$,3%)+"-"+       !               &
                                  MID(EMP.SSN$,4%,2%)+"-"+     !               &
                                  RIGHT(EMP.SSN$,6%)           !               &
        \ DL.05.FED.TAX$        = FORMAT$(EMP.FED.TAX,GEN.F$)  !               &
        \ DL.05.PAY.TIPS$       = FORMAT$(EMP.GROSS.PAY -      !               &
                                  HS.DEF.COMP,GEN.F$)          !               &
        \ DL.05.FICA$           = FORMAT$(EMP.FICA,GEN.F$)     !               &
        \ DL.06.EMP.NAME$       = PM.EMP.NAME$                 !               &
        \ DL.06.FICA.PAY$       = FORMAT$(EMP.FICA.PAY,GEN.F$) !               &
        \ DL.07.ADDRESS$        = PM.ADDRESS$                  !               &
        \ DL.08.CITY$           = PM.CITY$                     !               &
        \ DL.09.STATE$          = PM.STATE$                    !               &
        \ DL.09.ZIP$            = NUM1$(PM.ZIP)                !               &
        \ DL.09.ST.TAX$         = FORMAT$(EMP.STATE.TAX,GEN.F$) !              &
        \ DL.09.ST.PAY$         = FORMAT$(EMP.GROSS.PAY -      !               &
                                  HS.DEF.COMP,GEN.F$)          !               &
                                                               !               &
        \ SUB.DEF.COMP          = SUB.DEF.COMP   + EMP.DEF.COMP ! Accumulate   &
        \ SUB.EIC               = SUB.EIC        + EMP.EIC     ! values for    &
        \ SUB.FED.TAX           = SUB.FED.TAX    + EMP.FED.TAX ! subtotal W-2  &
        \ SUB.FICA              = SUB.FICA       + EMP.FICA    !               &
        \ SUB.FICA.PAY          = SUB.FICA.PAY   + EMP.FICA.PAY !              &
        \ SUB.GROSS.PAY         = SUB.GROSS.PAY  + EMP.GROSS.PAY!              &
                                                               !
5200      PM.DED.CODE%(I%),PM.DED.AMT(I%) = 0%                 ! Determine if  &
        \ GET #PM%, KEY #0 EQ CVT$$(EMP.SSN$,2%)               ! employee is   &
        \ R%     = RECOUNT                                     ! covered by a  &
        \ FOR I% = 0% TO ((R%-324%)/10%)-1%                    ! pension.      &
              \ MOVE FROM #PM%,FILL$ = 324% + (I%*10%)         !               &
                      ,PM.DED.CODE%(I%)                        !               &
                      ,PM.DED.AMT(I%)                          !               &
              \ SL.03.PENSION$ = "X" IF PM.DED.CODE%(I%) = 4%  !               &
        \ NEXT I%                                              !               &
                                                               !
5300      PRINT #LP%,                                          ! Print the     &
                LINE.FEED$;                                    ! employee W-2  &
                LINE.FEED$;                                    !               &
                DL.01.ALL$;                                    !               &
                LINE.FEED$;                                    !               &
                LINE.FEED$;                                    !               &
                DL.02.ALL$;                                    !               &
                LINE.FEED$;                                    !               &
                DL.03.ALL$;                                    !               &
                LINE.FEED$;                                    !               &
                LINE.FEED$;                                    !               &
                DL.04.ALL$;                                    !               &
                LINE.FEED$;                                    !               &
                LINE.FEED$;                                    !               &
                DL.05.ALL$;                                    !               &
                LINE.FEED$;                                    !               &
                LINE.FEED$;                                    !               &
                LINE.FEED$;                                    !               &
                DL.06.ALL$;                                    !               &
                LINE.FEED$;                                    !               &
                LINE.FEED$;                                    !               &
                DL.07.ALL$;                                    !               &
                DL.08.ALL$;                                    !               &
                DL.09.ALL$;                                    !               &
                LINE.FEED$;                                    !               &
                LINE.FEED$;                                    !               &
                LINE.FEED$;                                    !               &
                LINE.FEED$                                     !               &
                                                               ! Increment     &
        \ GEN.W2CTRLNO% = GEN.W2CTRLNO% + 1%                   ! the counter.  &
        \ FOR I% = 1% TO 10%                                   ! Do a sub W-2  &
                \ GOSUB 7000 IF GEN.W2CTRLNO% / (42% * I%) = 1% ! if count     &
        \ NEXT I%                                              ! divisible by  &
                                                               ! 42.           &
        \ GOSUB 6000                                           ! Reset para.   &
        \ RETURN                                               ! Return        &

6000    ! ========================= RESET PARAGRAPH. ========================= &
                                                                               &
          EMP.DEF.COMP,                                        ! Zero the      &
          EMP.EIC,                                             ! accumulators  &
          EMP.FED.TAX,                                         !               &
          EMP.FICA,                                            !               &
          EMP.FICA.PAY,                                        !               &
          EMP.GROSS.PAY,                                       !               &
          EMP.STATE.TAX                 =  0                   !               &
        \ EMP.SSN$ = HS.SSN$                                   ! Reset emp     &
                                                               ! ssn.          &
        \ RETURN                                               ! Return
```

DECUS

DECUS SUBSCRIPTION SERVICE ORDER FORM

RETURN TO: Subscription Service
DECUS
One Iron Way, MRO2-1/C11
Marlboro, MA 01752

- All checks payable to DECUS
- All orders MUST be paid in full
- No refunds will be made
- Prices indicated are FY'84 prices

Name ______________________________ DECUS Membership No. __________
(First) (Last)

Company/Affiliation ______________________________

Mailing Address ______________________________ Mail Stop __________

City __________ State/Country __________ Zip Code __________ Phone (   ) __________

PUBLICATIONS SUBSCRIPTION SELECTIONS

CODE	PUBLICATION	CODE	PUBLICATION
MSL	MUMPS/STRUCTURED LANGUAGES NEWSLETTER	RST	RSTS NEWSLETTER
LHS	LABS/HMS/SITE MGMT NEWSLETTER	LGS	LARGE SYSTEMS NEWSLETTER
OAD	OA/DIBOL/COBOL/GRAPH NEWSLETTER	EDU	EDUSIG NEWSLETTER
VAX	VAX/VMS NEWSLETTER	DTR	DATATRIEVE NEWSLETTER
RSX	RSX/IAS NEWSLETTER	NTW	NETWORKS NEWSLETTER
RT	RT11 NEWSLETTER	SOS	SS&OS NEWSLETTER
SPR	Spring Proceedings	BAS	BASIC NEWSLETTER
FAL	Fall Proceedings	APL	APL NEWSLETTER
		ALL	ALL PUBLICATIONS PRODUCED

	Insert Code From Above:	Check One:		
BASIC PLAN: This plan allows you to receive one (1) selection for one year	□	□	Member/DIGITAL Employee	$ 12.00
		□	Non Member	$ 24.00
STANDARD PLAN: This plan allows you to receive up to three (3) selections at one low price.	□ □ □	□	Member/DIGITAL Employee	$ 25.00
		□	Non Member	$ 50.00
DELUXE PLAN: This plan allows you to receive up to six (6) selections for one year.	□ □ □ □ □ □	□	Member/DIGITAL Employee	$ 45.00
		□	Non Member	$ 90.00
ALL: This will allow you to receive all publications listed above for one year for only one price.	ALL	□	Member/DIGITAL Employee	$120.00
		□	Non Member	$240.00

TOTAL AMOUNT OF ORDER $__________

I understand that neither DECUS nor Digital Equipment Corporation is responsible for any publication not published by a Special Interest Group or the contents of any publication published by a Special Interest Group. I also understand that there will be no refunds even if I decide to cancel my subscription.

Signature ______________________________ Date __________

DIGITAL Employees Only: Badge No. ______________________________ C.C. __________

Cost Center Manager's Signature ______________________________ C.C. __________

DECUS

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
ONE IRON WAY, MRO2-1/C11
MARLBORO, MASSACHUSETTS 01752

BULK RATE
U.S. POSTAGE
PAID
PERMIT NO. 129
NORTHBORO, MA
01532

ASSOCIATE - [redacted]
[redacted]

MOVING OR REPLACING A DELEGATE?

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

( ) Change of Address
( ) Delegate Replacement

DECUS Membership No.: ______________

Name: ______________________________

Company: ___________________________

Address: ____________________________

____

State/Country: ______________________

Zip/Postal Code: ____________________

Mail to: DECUS - ATT: Membership
One Iron Way, MR02-1/C11
Marlboro, Massachusetts 01752 USA

Affix mailing label here. If label is not available, print old address here. Include name of installation, company, university, etc.