Writing Applications for Uniform Operation on a Mainframe or PC: A Metric Conversion Program

Charles A. Schulz
Lockheed Missiles & Space Company, Inc.
Dept. 19-74, Bldg. 102
1111 Lockheed Way
Sunnyvale, California 94089-3504, USA

ABSTRACT

The metric system of measurement is the primary standard in all countries except the USA and two others. Use of the metric system is becoming more important to the USA for trade and commerce in the world economy. A metric conversion program was developed to convert 350 measurement units between inch-pound (or USA customary) and metric systems for engineering design and documentation. The program follows the primary national metric standard with its conversion factors and special rules for arithmetic, rounding, accuracy, formatting, and terminology. It was first developed in APL on a VM CMS mainframe system, but subsequent demand warranted a PC version. The program has been presented at national metric meetings and is briefly described here.

The portability of APL2 under VM CMS and PC DOS was exploited in moving the mainframe program to the PC, maintaining much of the code in identical form, thus giving advantages in testing, verification, and user experience. Differences between the environments fell mainly in the user and file interface areas, which led to a matched set of cover functions for system dependent operations. These functions and differences between the two systems are described for file operations, pop-up windows, printing, stack and host system commands.

The capability and portability of APL enabled this application to become the first internally written example at this company of IBM's new user interface guidelines working on both a central mainframe and a PC.

INTRODUCTION

The Metric System

The metric system of measurement is the primary standard in all countries except the USA, Burma, and Liberia. Even in the USA, industries such as medical, computer, and automobile, among others, already make significant use of metrics.

International standardization of measurement began in 1875 with the establishment by 17 nations of the International Bureau of Weights and Measures in France near Paris. This bureau is controlled by the General Conference on Weights and Measures that meets every few years in Paris with representatives from member nations. The internationally standard measurement system now in use, maintained by the Bureau, is the International System of Units (SI) 1 which incorporates the metric system.

The metric system has several advantages long recognized by scientific and technical circles. Use of the decimal system for scaling (multiples and submultiples) of units simplifies arithmetic with measurements. The coherence of units is based on unique units for each physical quantity (length, mass, time, etc.), clarifying relationships between fundamental and derived units. The use of unique units produces unique symbols that prevent confusions such as use of "b" for *bar* (pressure) and *barn* (cross section). Also, use of the predominant international system permits easier participation in international cooperation and trade.

Due to historically large trade interests with England in inch-pound products, the USA retained the inch-pound system while other countries implemented metrics. For this reason the inch-pound system of units is sometimes called the "English" system. Inertia and nationalism have continued the delay in adoption of metrics by much of USA industry and commerce until recently. Also, USA reliance on the inch-pound system of measurement has caused the confusion between weight and mass to persist from education into industry, whereas that problem has disappeared in other countries. It is interesting to note, however, that since 1893 the USA customary foot and pound have actually been defined in terms of the international metre [2] and kilogram.

Metric Need in the USA

Use of SI is becoming more important to the USA for trade and commerce in the world economy. This fact has been recognized in recent national developments. The Department of Commerce funds an Office of Metric Programs, and specific government department and agency policies have recently been appearing. For example, a policy which signaled the importance of metrics to the aerospace/defense industry was the 1987 Strategic Defense Initiative Office (SDIO) directive to use metrics in all new designs, or to use dual indication with metric units first, followed by USA customary units in parentheses[3]. The importance of metrics was finally recognized in legislative requirements for all government departments and agencies to use the metric system[4]. This legislation makes it the "policy of the United States to designate the metric system of measurement as the preferred system of weights and measures for United States trade and commerce." It requires government departments and agencies to use metrics in procurements, grants, and other business-related activities by the end of fiscal year 1992 to the extent that it is economically feasible.

Metric Need at LMSC

SI requirements are already showing up in Lockheed Missiles & Space Company (LMSC) contracts. A major consideration in the SDIO policy was interchangeability and interoperability with allies' systems. SI's advantages may also apply to LMSC's work on non-defense aerospace projects such as the Space Station *Freedom*, for which international participation by Europe, Japan, and Canada is already planned.

METRIC CONVERSION PROGRAM

Purpose

To address LMSC's growing needs for conversion of existing designs and comparison of USA customary and SI unit measurements, the COMET (COmputerized METrication) program was developed. The program uses standard conversion factors and special rules for arithmetic, rounding, accuracy, formatting (style), and terminology to convert 350 measurement units between USA customary and SI systems for engineering design and documentation. These factors and rules follow the primary national metric standard [5] and an aerospace standard for preferred units[6].

COMET's primary intention is to provide existing inch-pound engineering designs a standard metric *soft* conversion, where equivalent metric units do not change the physical configuration beyond original tolerances. This conversion allows existing inch-pound tooling to be used during production from metric converted engineering designs. While it is important to fall within the original tolerances when calculating a soft conversion, it is just as important to not exaggerate the required accuracy in order to avoid resulting increased production costs.

COMET's secondary intentions are to assist metric *hard* conversion or design through testing of tolerances, and conversion verification and education through testing of conversions. Metric hard conversion changes the existing design to fit metric tooling so that the physical configuration can change beyond original tolerances. The program also converts from SI to USA customary units for verification of USA customary unit to SI conversions and special situations such as using metric tooling for production of inch-pound designs.

The use of COMET should preclude common mistakes such as incorrect application of significant digits rules, forgetting to square or cube the prefix on area or volume units, confusing use of the kilogram standard factors when other scalings of grams are needed, or incorrect use of standard symbols and abbreviations.

LMSC currently uses COMET to standardize metric documents such as design handbook sections, design standards, and material and process specifications. These uses have in turn revealed technical and user requirements that have been used to refine the program.

LMSC is considering interfacing COMET to CADAM, the computer aided design and manufacturing system used extensively at LMSC, to provide rounding when converting inch-pound designs. CADAM handles hard metric designs or direct metric conversion from inch designs, but it does not provide necessary rounding features. There are thousands of existing inch-pound unit designs, stored in data bases from which components and modifications are often reused for new contracts, that could be applied on metric contracts requiring soft conversion.

COMET is available as a public utility to all 6,000 users of the central LMSC VM/CMS system. A PC version has also been developed. Versions are being considered as utilities for other computing environments.

Presentations

Although COMET was first envisioned as a simple straightforward application for a common need, unexpected complexities were encountered that made it a more difficult application. Work in progress has been discussed at conferences and in newsletters to compare with similar efforts at other companies.

COMET was presented at the U.S. Metric Association 1988 annual conference, the first forum after the 1988 legislation to discuss government responsibilities for implementing metric provisions of the legislation. The conference included attendees from government, industry, and academia, including members of the American Society for Testing and Materials (ASTM) Committee E-43 on Metric Practice, which is responsible for the primary national standard. As a result of this presentation, LMSC was invited to participate on the E-43 committee. COMET work was described further in a manufacturing newsletter[7], and a metric newsletter[8]. In 1989, the PC version of

COMET was demonstrated and discussed at the annual meeting of the full E-43 committee. Refinements to the program are continuing as a result of these presentations.

Usage Features

COMET is basically an input to output converter. The program handles single values, toleranced values, and ranges of values with several rounding methods and percent error limiting. It uses rules of significant digits for calculations, as specified in the ASTM national standard, rather than simple computer or calculator arithmetic. Results are handled up to 16 digits – more than a calculator, important for some conversions. 16 SI prefixes can be used to modify scaling of conversions. Results are direct conversions and recommended significant digit roundings within standard error percentages. Input is accepted in decimal, exponential (scientific), or fractional form. Output includes formulas, long names, accepted abbreviations (SI symbols), and optional comments.

The usage goal is to provide a converted recommended number without much effort, and to provide extra information about that conversion for reference. This involves some basic features, perhaps easily foreseen, including:

- Optional sort so the original order of inputs can be preserved to follow the appearance of those measurements on engineering drawings
- Labeling, displaying, printing, or saving of outputs
- Executing a file of inputs as a procedure batch mode for large amounts or repetitive sets of data.

COMET uses a series of pop-up window menus, prompts, and displays to provide instructions and request conversion selections, input, and output options. This user interface attempts to follow guidelines from IBM's Systems Application Architecture™ (SAA™) Common User Access (CUA) concept [9] to maintain usability and familiarity in programs between mainframes and workstations. CUA is a style of user interface that IBM is using in its new products and applications. It consists of many guidelines based on past experience and thorough consideration of computer terminal and workstation features. General flavors are the use of overlapping windows to illustrate the paths through a program, and well-defined actions from standard kinds of windows.

In the case of COMET, CUA is implemented in both the CMS and DOS environments at the non-programmable terminal level. Graphical features, icons, mouse handling, and multi-tasking windowing, as discussed in CUA for programmable workstations, are not handled. These features are aimed more towards the OS 2™ environment. Actions in COMET consist of entering answers and inputs with the Enter key and using programmable function (PF or F) keys shown on the windows. The user dialog consists of primary windows with pop-up windows within them. Pop-up windows display actions that must be completed before returning to underlaying windows. Progress messages are displayed at the bottom of the screen or in pop-up displays.

A few more minor style choices were made. An action bar with pull down menus was not needed for the small number of screens in this application. Screen color recommendations were mostly followed, and differences in recommendations for the VM terminals and PC displays did take advantage of differences in hardware color between them. The "Cancel" function, recommended for all screens as canceling the current action, is labeled as "Return" for compatibility with many current mainframe applications.

Standard actions are assigned to standard keys as shown in Table 1.

All screens have the same general layout from top to bottom:

- Window title
- Scroll information showing if more display is available than will fit in the window
- Data display or instructions, selections and input fields
- Message line for progress messages or special instructions
- Action keys.

Specific types of screens are as follows:

Single selection menus: An item is selected by entering its number in the field next to the item numbers or by moving the cursor next to an item number and using Enter. In some cases, a default item number may be displayed in the selection field; it must be erased before making a selection with the cursor. Figure 1 shows a pop-up menu for a particular input field on the underlaying window with a selection typed in the selection field next to the item numbers.

Multiple selection menus: This is a variation of the single selection menu where one or more items are selected by typing a slash (/) next to each item.

Entry panels: Inputs are typed as instructed. A question mark (?) typed in an input field provides a pop-up menu of available choices for a specific input field. The "Prompt" action key provides pop-up menus of available choices for all fields. A sample is shown in Figure 2.

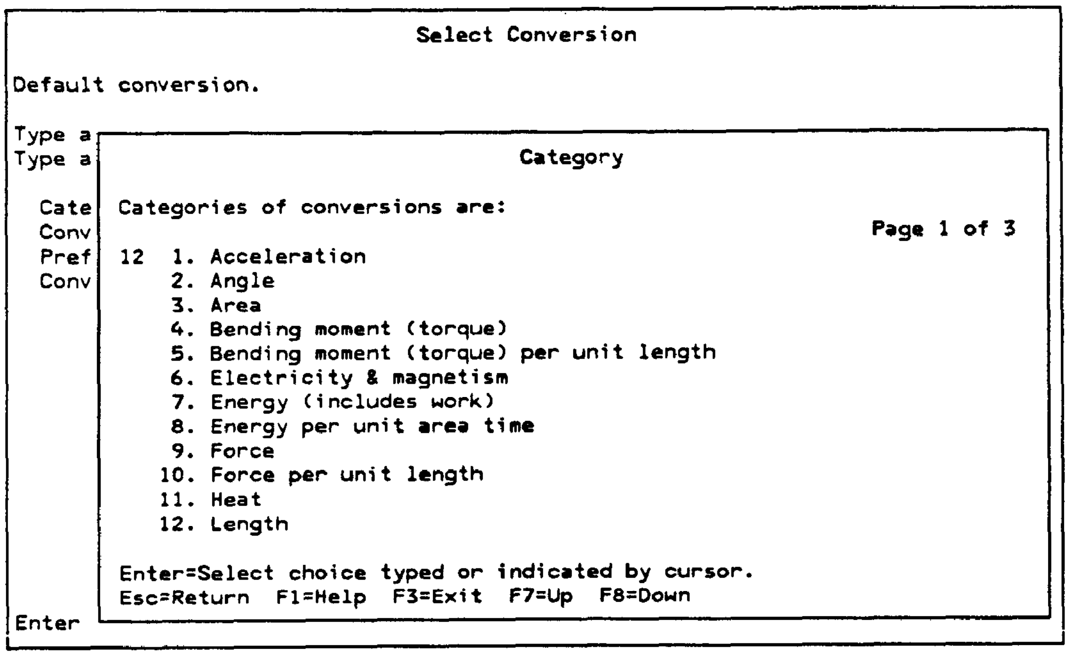
Data displays: Data is scrolled through the window with the action keys. Scrolling can also be performed by modifying the starting line number in the scroll information line. A sample is shown in Figure 3.

Pop-up messages: Progress messages may use pop-up message displays to emphasize activity. A sample is shown in Figure 4.

In COMET, each pop-up window overlaps the previous window's action keys displayed at the bottom, so that it is clear which window is active, as shown in Figure 5. Also, input fields on underlaying windows are reset so that tab keys will not move to them.

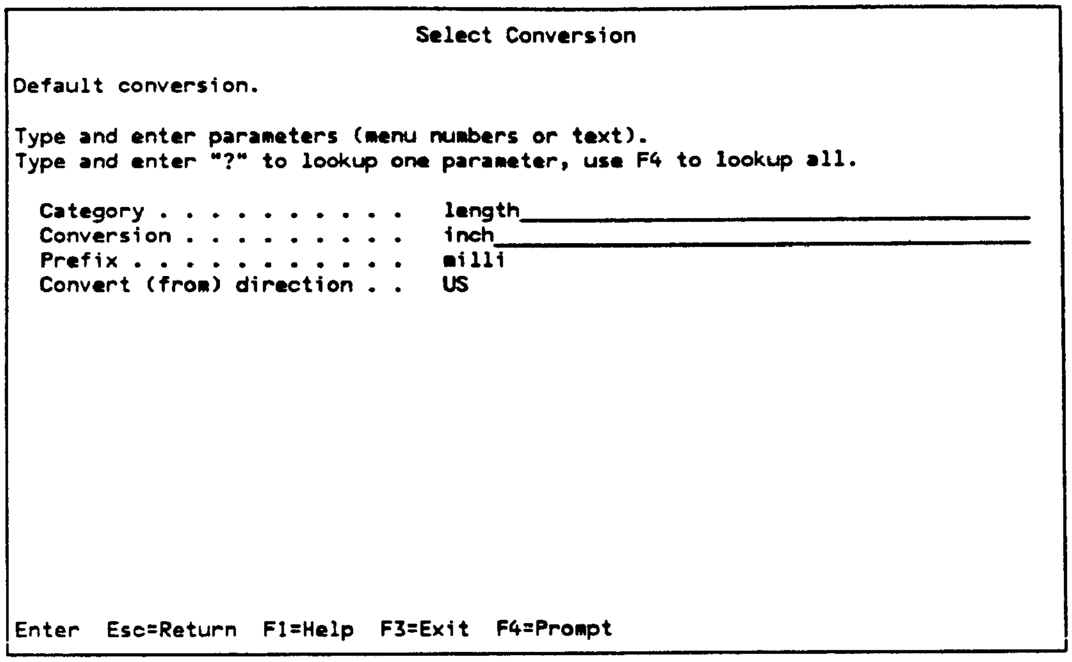
Key	*Action*	*Description*
Enter	Select or proceed	Act on information typed on the screen and proceed through the program.
F1	Help	Display a pop-up menu of help sections to select and scroll through.
F3	Exit	Display a pop-up menu of yes/no to exit the program.
F4	Prompt	Provide pop-up selection menus for each entry field of possible choices.
F7 and Page Up	Up	Scroll backward 1 window.
F8 and Page Down	Down	Scroll forward 1 window.
F10	Actions	If there is more than one possible action from a screen, display a pop-up menu of actions for selection.
F12 and Esc	Return	Cancels or returns from the current action window to the previous action or display.

Table 1. Standard action keys



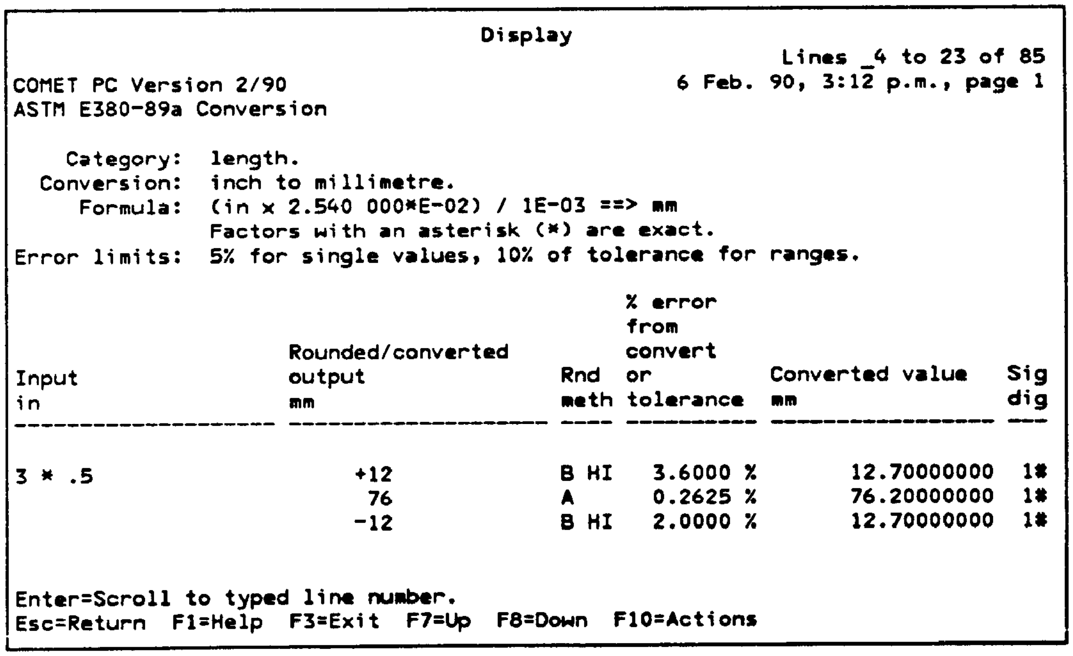
```
                              Select Conversion

Default conversion.

Type a
Type a                               Category

  Cate  Categories of conversions are:
  Conv                                                    Page 1 of 3
  Pref  12  1. Acceleration
  Conv      2. Angle
            3. Area
            4. Bending moment (torque)
            5. Bending moment (torque) per unit length
            6. Electricity & magnetism
            7. Energy (includes work)
            8. Energy per unit area time
            9. Force
           10. Force per unit length
           11. Heat
           12. Length

        Enter=Select choice typed or indicated by cursor.
        Esc=Return  F1=Help  F3=Exit  F7=Up  F8=Down
Enter
```

Figure 1. Single selection pop-up menu



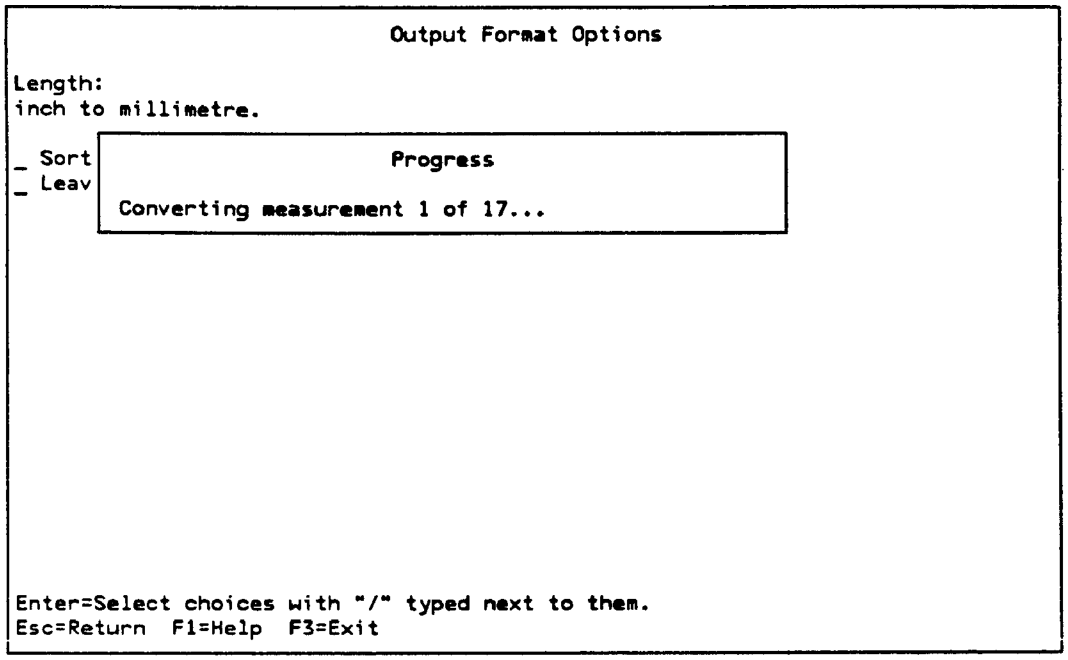
```
                              Select Conversion

Default conversion.

Type and enter parameters (menu numbers or text).
Type and enter "?" to lookup one parameter, use F4 to lookup all.

  Category . . . . . . . . . .   length_______________________
  Conversion . . . . . . . . .   inch_________________________
  Prefix . . . . . . . . . . .   milli
  Convert (from) direction . .   US










Enter   Esc=Return   F1=Help   F3=Exit   F4=Prompt
```

Figure 2. Entry panel



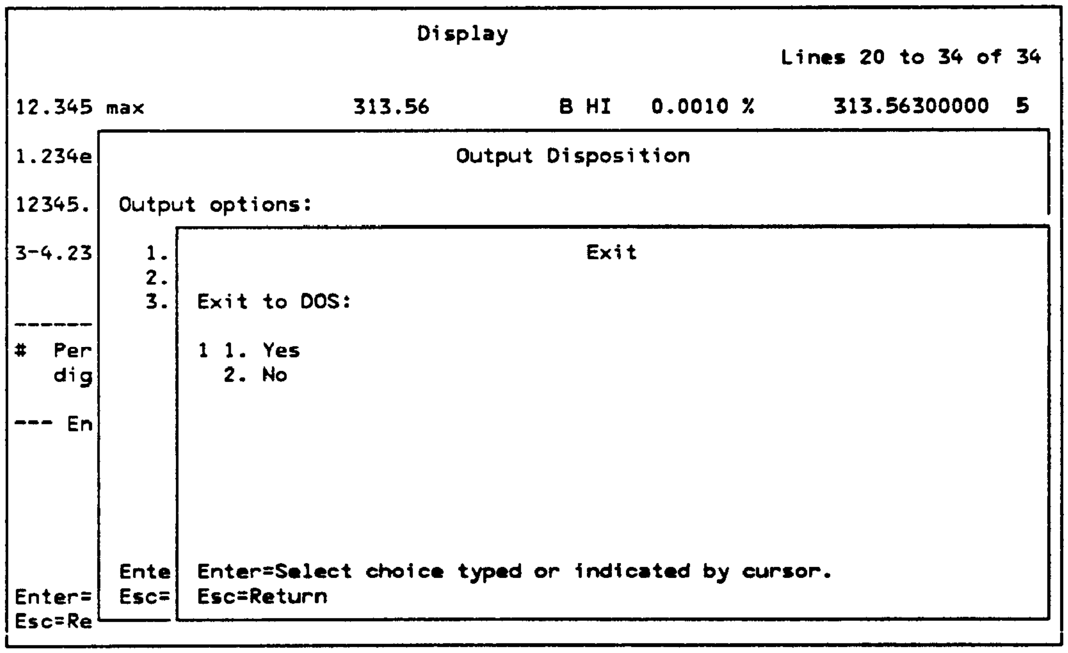
```
                                  Display
                                                      Lines _4 to 23 of 85
COMET PC Version 2/90                             6 Feb. 90, 3:12 p.m., page 1
ASTM E380-89a Conversion

    Category:  length.
  Conversion:  inch to millimetre.
     Formula:  (in x 2.540 000*E-02) / 1E-03 ==> mm
               Factors with an asterisk (*) are exact.
Error limits:  5% for single values, 10% of tolerance for ranges.

                                                % error
                                                from
                      Rounded/converted         convert
Input                 output               Rnd  or          Converted value   Sig
in                    mm                   meth tolerance   mm                dig
--------------------  -------------------- ---- ----------  ----------------  ---

3 * .5                        +12          B HI   3.6000 %       12.70000000   1*
                               76          A      0.2625 %       76.20000000   1*
                              -12          B HI   2.0000 %       12.70000000   1*


Enter=Scroll to typed line number.
Esc=Return   F1=Help   F3=Exit   F7=Up   F8=Down   F10=Actions
```

Figure 3. Scrollable data display



```
                              Output Format Options

Length:
inch to millimetre.

_ Sort                       Progress
_ Leav
        Converting measurement 1 of 17...










Enter=Select choices with "/" typed next to them.
Esc=Return  F1=Help  F3=Exit
```

Figure 4. Pop-up message overlapping multiple selection menu



```
                              Display
                                                         Lines 20 to 34 of 34

12.345 max              313.56          B HI    0.0010 %      313.56300000  5

1.234e                           Output Disposition

12345.  Output options:

3-4.23    1.                                  Exit
          2.
          3.  Exit to DOS:

------
#  Per        1 1. Yes
   dig          2. No

--- En







        Ente  Enter=Select choice typed or indicated by cursor.
Enter=  Esc=  Esc=Return
Esc=Re
```

Figure 5. Action keys only in active window

Program Design

Although COMET started out as a simple program, unexpected difficulties were encountered that made it a more difficult application continuing off and on over a couple of years. Significant digits arithmetic alone was complex. This involved determining significant digits in input measurements, following rules for arithmetic with significant digits in conversion and rounding calculations, and displaying significant digits properly in output. Occasional edge condition examples of incorrect roundings produced the need to pre-round numbers in order to precisely follow the rounding rules. Various error checking conditions were added such as checking for input of a temperature that would convert to a negative Kelvin temperature. SI terminology and style rules complicated proper handling of prefixes in compound units and capitalization, and in number formatting.

Unexpected requirements appeared from actual usage. For example, input error checking for zero value tolerances had to be changed when engineers encountered real situations using a bilateral tolerance, one of which was indeed zero, e.g.:

```
    1.25+.05-0
```

Finally, providing a modern user interface without any prewritten SAA CUA tools or utilities turned out to be more than trivial.

COMET was first developed in VS APL on a VM/CMS mainframe system. While the choice of APL originally seemed suited to this application for handling arithmetic simply, it eventually was quite necessary in actually understanding the arithmetic, which was of course essential to programming it correctly. APL's ability to handle character data as easily as numeric data also suited it to manipulation of the character information, conversion data tables, and user interface features such as menus.

After an early version of the mainframe program was in use, demand warranted a PC version for parts of LMSC where the central VM system was not used. About this time IBM made APL2/PC available — an attractive product for this situation due to its compatibility with mainframe APL (APL2) under VM/CMS and IBM's permission to package programs for distribution without licensing obligation.

PORTING TO THE PC

Transfer Files

Some extra work was necessary without a copy of APL2 on the mainframe with which APL2/PC was easily compatible, and to which VS APL code could be migrated automatically with)MCOPY. IBM provided a printout of the INX̲ and OUTX̲ programs from APL2/VM's 2 TRANSFER workspace that were typed in manually to VS APL. These read and write VS APL code in transfer form files readable by APL2's)IN and)OUT.

Transfer files produced by OUTX̲ in VS APL and downloaded in binary form to the PC are translated and read by APL2/PC's)IN just as those produced by)OUT in APL2/370 (mainframe for VM/CMS or MVS/TSO) would be.

Transfer files produced by)OUT in APL2/PC can be uploaded in binary form to be read by APL2/370's)IN. Transferring code from APL2/PC back to VS APL had some extra difficulties:

- APL2/370 from Release 3 performs the appropriate translation from ASCII to EBCDIC character representation. The character translation for VS APL was figured out by trial and error in passing a variable of ⎕AV back and forth between VS APL and APL2/PC.
- The APL2)OUT command might use either migration or extended transfer forms for objects. The INX̲ program typed in for COMET was unable to read a transfer file created with)OUT by APL2/PC, but was able to read a transfer file created with the OUT function from the APL2/PC FILE workspace, which uses only the extended transfer form via ⎕TF.
- INX̲ in VS APL also seemed to require that the transfer file be uploaded with a CR/LF option to retain record breaks which APL2/370 would otherwise recognize.
- And, of course, APL2 features not compatible with VS APL had to be avoided.

Isolating System Dependencies

From this point, the challenge in moving the mainframe program to the PC was to preserve as much as possible of the program between environments, in order to maintain the integrity of testing and verification, as well as the users' experience, for both versions. Since the VS APL code was upward compatible with APL2/PC with very few exceptions, the differences would fall in system dependent areas such as files, screens, and operating system commands. By appropriately isolating these types of operations, a parallel set of cover functions could be used so that the calculation and reporting parts of the code would remain identical. In a few cases dual environment functions were appropriate, such as character case translation.

Unfortunately, the first attempt at replacing the obvious system dependent functions showed how poorly these had been isolated from the other parts of the program code or were dependent upon special features of the environment. It took two weeks to get the APL2/PC version to successfully limp through. More optimistically, this exercise identified areas in the code where improvement would make future maintenance and enhancements much easier. These improvements could also open the way for consideration of porting to yet other systems, since the main calculation code would already be independent of two operating systems.

Areas of the code were rewritten in bits and pieces as the application progressed, and have now resulted in a nice set of functions for system dependencies that can fit in an application workspace on either the mainframe or the PC. These functions deal almost exclusively with the differences in auxiliary processors (APs) between the mainframe and PC APLs.

APL Versions

APL2 has recently become available on LMSC's VM/CMS mainframe. But initial work was done with VS APL Release 4 for VM/CMS and APL2/PC Version 1 (and later Version 1.01). Extra care had to be taken on the PC side to use APL2 style code only in the PC specific functions and to continue using VS APL style in parts of the code that also run on the mainframe. This was difficult and confusing at times because it was tempting to use new APL2 features. But new APL2 features really fell into just a few areas to think twice about before using:

- Nested arrays in variables and as arguments to functions
- New primitives related to nested arrays, such as enclose, disclose, depth
- New primitives match, each, find, dyadic grade, index
- New system functions and variables for event handling
- Comments at the end of code lines.

For compatibility of VS APL style with APL2 PC code, there were also only a few problems. The general trap for any new user of APL2 is, of course, the possibility of accidentally using vector notation by leaving out a primitive, such as ravel, during edit of a function: rather than the editing error being identified with a SYNTAX ERROR, the line of code actually executes and produces a nested result which then leads to other troubles in following lines. Other differences that arose included:

- VS APL's evaluation of brackets allows selection from a constant in vector notation, whereas APL2 requires parenthesis around vector constants
- ⎕AV uses different character orderings in VS APL, APL2 PC, and APL2 370
- Monadic format of rank 2 or greater arrays includes a column of leading blanks in VS APL, but not in APL2.

In the monadic format case, and in a few functions that handle dual environment operations, the environment checking function shown in Figure 6 was useful. Specifically for the format case, the APL function was used with format as a phrase:

```
(0,1=APL)↓⍕
```

For the differences between VS APL and APL2, the IBM migration guide [10] was quite useful. Interestingly, in this manual, execute alternative ⎕EA is noted as a feature new with APL2, although it is available in VS APL and was used in COMET. Apparently it is an undocumented or test feature of VS APL.

Files

Both VS APL and APL2 for VM use AP 110 for sequential or direct access to CMS files of fixed or variable length records. APL2/PC uses AP 210 for sequential or direct access to DOS files of fixed or variable length records. Both APs use a control and a data shared variable. The principal differences between the two APs are the use of the shared variables and the availability of file size information.

In AP 110, after a file is opened, the control variable contains the most recent return code, the record position of the read pointer, the record position of the write pointer, and the blocking factor for multiple fixed length records. Records are written by setting the data variable, and read by referencing the data variable. The control variable pointers are updated automatically by reads and writes, and can be altered to control which and how many records are accessed. The file size, in terms of records, is determined by inspecting the control variable write pointer at file open time.

In AP 210, after a file is opened, the control variable contains the most recent return code and is used to set numeric operation codes and optional record numbers and sizes. Records are written by setting the data variable, then setting the control variable with a write operation code. Records are read by setting the control variable with a read operation code, then referencing the data variable. The AP maintains pointers internally for sequential accesses, but these are not available to the user. The file size, in terms of bytes, is shown in the control variable at file open time.

These differences and the capabilities of the APs have several effects on the PC side:

- To determine the number of records in a file, you must know the record length for fixed length records or read the entire file to count CR/LF sequences for variable length records.
- Direct access to variable length records is much less efficient than sequential access because the file is scanned from the beginning in search of the requested record by CR/LF sequences.
- A copy of the last record accessed is kept in the data variable until it is reset, i.e. in the active workspace.

```
      ∇
[0]    R←APL
[1]   ⍝ Returns 1, 2, or 3 for VSAPL, APL2/PC, or APL2/VM.
[2]   ⍝ (In order of COMET development.)
[3]    R←⎕AV[39 98 130]⍳'a'
      ∇
```

Figure 6. Environment checking function

- In contrast to AP 110, the internal write pointer always starts at the beginning of the file rather than at the end.
- The size of fixed length records cannot be determined by reading one, because the length defaults to 128 unless specified.
- Blocked multiple record access is not directly available. Several fixed length records can be read by a multiplied record size, but this requires a record number which is also figured by the multiplied record size.
- Fixed length record files are not as easily transferred to the mainframe because there are no CR LFs to indicate record breaks.

To handle these effects, accommodations were made in use of files:

- Where scrolling display of file contents was expected, files were written with fixed length records, using a known length, for more efficient read access.
- All read and write functions immediately reset the data variable with a null value to release workspace taken by occasionally large file records.
- The first sequential write to a file would use 32767, the largest record number possible, to force the pointer to the end.
- Blocked read and write were simulated by multiple accesses for variable length and fixed length records.
- Where fixed length record files were to be made available for transfer to the mainframe, they were re-written to variable length format. This avoids the need for a utility on the mainframe side to break the file into proper size records.

The common denominator file operations used by COMET and adjusted for the AP differences are described in Figure 7. The APL2/VM and APL2/PC versions of these functions handle files with file ID numbers and create numbered sets of shared variables. In the case of AP 210, a numbered variable is maintained with the shared variables to keep track of records and blocking.

Screens

Enough code had to be changed in order to separate system dependencies that there was the chance of trying out the new SAA CUA guidelines. Use of CUA would take advantage of wider experience in user interface considerations and consistency between non-programmable (mainframe) terminal and workstation (PC) techniques. "Trying out" in this case meant attempting to follow them without any previously developed utilities that implemented the guidelines.

VS APL and APL2 for VM use AP 126 for management of full-screen panels. AP 126 provides access to several hundred routines from the Graphical Data Display Manager (GDDM™) product and a few service requests for dealing with GDDM. GDDM includes calls for graphic and full-screen character field services.

APL2 PC uses AP 124 for management of full-screen panels. This is similar to the original AP 124 provided with VS APL before GDDM was available. It handles full-screen character field services but no graphics.

FILECAPLCLOSE	Closes file number opened by FILECAPLOPEN.
FILECAPLOPEN	Opens/creates file for read or sequential write of APL objects. Identifies file ID with a file number for the shared variables.
FILECAPLREAD	Reads records from file number opened by FILECAPLOPEN, null at end. Can take an optional record number to change read pointer.
FILECAPLWRITE	Writes record sequentially to file number opened by FILECAPLOPEN.
FILECLOSE	Closes file number opened by FILEOPEN.
FILEMAT	Returns a file as a matrix.
FILEOPEN	Opens/creates file for read/write, with optional blocking factor. Identifies file ID with a file number for the shared variables. Blocking specified on new file creates for fixed length records.
FILEREAD	Reads records from file number opened by FILEOPEN, null at end. If blocking used in FILEOPEN then returns matrix block, else 1 record. Can take optional record number from which to start reading.
FILESIZE	Returns number of records in a file, assuming fixed length format.
FILEWRITE	Writes record(s) sequentially to file number opened by FILEOPEN. Records are rank ≤ 2; may have more rows than the blocking factor.
NEXTFILE	Returns next suffixed file name not currently in use.

Figure 7. File functions

Both APs use a control and a data shared variable and deal with full-screen services in terms of rectangular fields on the screen described by numeric field attribute matrices. The principal differences between the two APs' full-screen services are the use of shared variables, identification of fields, and handling of overlapping windows.

In AP 126 the control variable is used to make GDDM and AP 126 requests as numeric call codes, and to report return codes and numeric parameters for the service requests. The data variable is used to pass character parameters and data between GDDM and the workspace. The return codes in the control variable also indicate if character data is returned in the data variable. The separation of numeric and character parameters between control and data variables predates the availability of mixed type arrays in APL2. Fields are identified by user specified numbers. New fields are added to a screen by a reformat operation with a new field number. Overlapping of fields is handled by GDDM *partitions*, which permit definition of new fields within rectangular sub-portions of the main screen, using separate format matrices.

In AP 124 for APL2 PC, the control variable is used to request numeric command codes and report return codes. The data variable is used to pass numeric or character parameters and data between the AP and the workspace. Fields are identified by their order in the original format matrix. New fields are added to a screen only by reformatting fields in the original format matrix, although 0 rows can be included in the original matrix for this purpose. Overlapping fields are permitted field by field, where the last definition of a location on the screen overlaps previous definitions.

There are a few other minor differences in operation between the two APs:

- In AP 126, the physical display size can be queried at any time with a GDDM call, but in AP 124 it can be queried only before a format matrix is in effect by querying the default format matrix.
- GDDM does not report fields modified with the read action request. However, separate GDDM status calls or an AP 126 service request can accomplish this.
- Unlike GDDM, AP 124 handles read or write with multiple fields on a single AP call.

These differences and the capabilities of the APs have several effects on the PC side:

- AP 124 works with one pair of shared variables at a time and the supplied editors in the EDIT workspace also use AP 124. Development of AP 124 code can be complicated by conflicts between the editor and pending states.
- New fields must use pre-allocated format matrix field rows. The maximum number of possible fields must be known ahead of time.
- Underlaying field contents and attributes, if altered from the original format, must be saved in the workspace for future queries, because queries will only show new contents and attributes from overlaying fields.
- Underlaying fields cannot be re-used simply by removing overlaying fields with reformat to 0 requests. Latest display and input/output attributes remain on the screen.
- Underlaying input fields cannot be re-used for input simply by reformatting them with their original attributes and overlaying non-input fields with 0s. A format screen request or new overlaying input field reformat is needed.
- Individual field description attributes can be inspected only by requesting a get of the whole format matrix. This can affect performance in handling of large matrices.
- Management of which fields in the single format matrix correspond to which windows must be handled in the workspace rather than by the separate format matrices available for each partition in GDDM.
- Windows must use a background field covering the whole window to overlay previous fields.
- The cursor position in a window is not remembered as it is in AP 126 partitions.
- The cursor does not show in invisible fields. Selection fields to receive tab key movements must use visible fields that can look invisible with blank characters in them, but which thereby also permit visible typing.
- Color displays have 255 possible combinations of foreground and background colors and highlight attributes versus 30 for AP 126.

These differences and effects were handled in several ways on the PC side:

- A standard calling function, FSCALL, handles shared variable sets and queries. A null option to retract the variables was handy when the APL2/PC full-screen editor needed to be used.
- Global variables are used to keep track of display size and active window information handled by GDDM.
- Underlaying window format matrices and contents that are preserved by GDDM are kept in global variables for quick restore of underlaying windows and query of particular field characteristics for reading and writing contents.
- The special control information request (command 8 3) and peek/poke ⎕PK were used to retain and refresh whole screen contents without having to reconstruct their contents and use field level writes.
- A small function SCR∆ALL was used to display all possible field color and highlight combinations in order to simplify making choices between them.

All screens in COMET are built dynamically for flexibility in maintenance. However, a set of FSA∆-prefixed variables, keeping full-screen attributes for standard field types (title, instructions, choices, input, etc.), are prepared in advance to avoid lookups at each usage.

The basic screen operation functions and variables used by COMET and adjusted for AP differences are prefixed with FS. A few of these are particular to AP 126 or 124, described in Figure 8. The rest, the common denominator functions and variables with a few related functions, are described in Figure 9.

Other screen functions that build on the FS functions for SAA CUA operations and are identical for both APs fall into two categories:

- SCR-prefixed functions for defining standard components of the SAA CUA panel types, shown in Figure 10
- Functions for generating and operating the standard SAA CUA menu and data display panel types, also shown in Figure 10.

Printing

In COMET, printing is handled by VM CMS facilities for the mainframe version and by AP 80 for the APL2/PC version.

In VM CMS, the PRINT command is used to print a file to a printer, reading the first column as standard carriage control characters (single space, double space, etc.). Related commands direct the printing to particular printers.

AP 80, the printer AP, is quite simple to use. One shared variable accepts numeric command codes or character lines for the printer with imbedded control codes, and reports return codes. Control codes are available to duplicate those to which the PRINT command in VM CMS responds. The sole difficulty seems to be in querying whether the printer is turned on, because the status query code seems to hang until the user turns it on or interrupts the program.

A single function PRINTFILE was used in both environments for printing an existing disk file.

Stacking Commands

VS APL and APL2 for VM and APL2/PC all use an AP 101 for stacking commands. The VM and PC APs both use one shared variable to accept character lines to stack and to report return codes. The only important difference between them is that the PC AP 101 stacks only in first-in-first-out (FIFO) order, whereas the VM AP can stack lines in either FIFO or last-in-first-out (LIFO) order.

A single function STACK was used in both environments for stacking lines.

Operating System Commands

VS APL and APL2 for VM and APL2/PC all provide an AP 100 for host operating system commands. The VM and PC APs both use a shared variable to accept character commands and report return codes.

The PC AP has two differences worth noting:

1. DOS commands that update the file directory, such as COPY or ERASE, will not execute if other files are still open via the file AP 210.
2. It always reports a successful command with return code 0, regardless of the command. While this is probably due to lack of information from DOS, the programmer must be careful to ensure that commands really are executed.

Due to these differences, PC operating system commands were handled in two different ways:

1. COPY and ERASE functions were written as dual environment functions. In CMS, a CMS function was called for these file operations. In DOS, erase was performed via a file AP 210 request, and copy was performed by reading and writing a new file via the AP 210 functions shown previously in Figure 7.
2. Other PC commands such as directory creates and queries were handled via AP 103 for BIOS DOS interrupts and function calls. Some of the more useful DOS commands are handled by functions in the provided DOSFNS workspace.

For AP 126 only:

FSQMOD	Returns vector of screen fields modified.

For AP 124 only:

FSAΔCOLOR	Switches FSAΔ— variables to a palette of colors.
FSCOLOR	Switches window to a palette of field colors.
FSPALETTE	Variable of current color palette.
FSPALETTES	Lookup table of color palette choices.
FSTERMINAL	Variable of terminal size.

Figure 8. Basic fullscreen functions and variables for particular APs

FKEYΔFMT	Returns formatted line of selected Enter and function keys.
FSAΔPREP	Prepares FSAΔ-- field attribute variables by lookups in FSFIELDS.
FSALARM	Sounds the terminal alarm at next screen write.
FSBORDER	Formats and writes border on pop-up windows.
FSCALL	Executes fullscreen call and returns results from data variable for query calls, from control variable for others; checks return codes. Shares with AP and makes initial settings if not already shared. Takes numeric call codes or ⍳0 to retract share, optional data for data variable.
FSCLEAN	Maintenance clean up of pending windows.
FSCLOSE	Closes current window, re-establishes previous windows, if any.
FSCURSOR	Places the cursor in a field, row and column.
FSFORCE	Immediately displays fullscreen updates made so far.
FSFORMAT	Formats fields in numeric matrix, adding offsets for windows.
FSGET	Returns contents of a field as vector or array.
FSMGET	Reads multiple screen fields into rows of an array.
FSMWRITE	Writes data to multiple fields, row by row.
FSOPEN	Opens new window; overlays and saves previous window if any.
FSQSIZE	Queries row/column size for device or window.
FSREAD	Updates the screen display; waits for and reads input actions with special keys; returns key actions, cursor position, fields modified.
FSREDEF	Redefines attributes for fields, prepared from FSFIELDS.
FSSAVE	Saves pending window features in sequential number suffixed variables.
FSWRITE	Writes data to a screen field, as delayed write.
KEY	Writes optional information or error message, waits for user action, reads action, checks specified keys, acts on standard keys if not specified. Returns good F key number or 0 for Enter, cursor position, modified fields.
MESSAGE	Writes user message with attributes for error or information.
ENTERS	Lookup table of Enter key definitions to show.
FKEYS	Lookup table of function key definitions to show.
FSAΔ--	Variables of attribute for standard field types.
FSFIELDS	Lookup table for standard field types, attributes.
FSFILL	Flag to fill input fields with underscores.
FSHOLD	Flag to hold a window open for use of its message line until the next window is ready.
FSWINDOW	Variable of current window ID and position data.

Figure 9. Basic full-screen functions and variables for AP 124 and 126

SCR∆PROMPT	Produces screen format for multiple one-line prompts or one multiple input line prompt.
SCR∆SCROLL	Produces fields for scroll information line in: - line format: lines xx to yy of zz (with xx input) - page format: page x of y (output only).
SCR∆SHOW	Produces screen format to show text, with optional attributes.
SCR∆STANDARD	Produces screen format for standard title, message, F key lines; optionally for heading lines, instruction lines.
SCREEN	Write standard fields on special screens: title top line, F keys bottom line(s), message line just above F key line.
SCREEN∆MENU	Displays menu of numbered choices on full screen with scrolling if necessary; returns choice number input or indicated by cursor. Uses controls for panel title, default choice, help name, info or error message, PF numbers to prevent standard screen actions, etc.
SCREEN∆MULT	Displays menu of choices on full screen for multiple selection, returns boolean vector of choices selected. Uses arguments similar to SCREEN∆MENU's.
SCROLL	Displays a matrix or file with F keys to scroll and return. Uses controls for panel title, action options (F10) to execute, help available, etc.

Figure 10. Standard panel component and panel functions

CONCLUSIONS

The portability of APL under VM CMS and PC DOS was successfully exploited to move the COMET application from VM CMS to PC DOS. Because the initial VM CMS version had not been originally planned for porting to the PC, system dependent operations had not been completely isolated. However, once porting was attempted, the differences between the environments quickly identified those spots needing revision.

There were very few differences in the operation of the APL language itself either in APL2 or VS APL, aside from availability of new features in APL2. Instead, the differences fell in those areas handled by auxiliary processors, illustrating the natural separation of language and environment for which the AP concept was designed.

The ability to isolate system dependencies in functions dealing with APs was successfully used to keep the user interface, calculation and reporting parts of the code absolutely identical between the mainframe and PC environments. This enabled consistent testing, verification, user experience, and documentation for both versions.

The capability of APL2 provided the possibility for a single programmer to implement IBM's SAA CUA guidelines at the application level, without CUA operating system facilities or utilities. In fact, this capability enabled COMET to become the first LMSC internally written example of CUA interface guidelines working on both a central mainframe and a PC. This is especially significant given the erratic availability of development time experienced with this application, and gradual refinement over a couple of years. APL2 code was easily re-explorable and maintainable over long periods between work on the application.

Another advantage to the portability of APL2 was the usefulness of passing APL2 PC code back to the mainframe during continuing refinement of COMET. No change had to be made twice, it was simply passed up or down between versions. Also, development on the PC escaped potentially expensive mainframe billing charges.

A measure of the success for the matched set of system dependent functions and the CUA interface could be complexity of the code. As an indication, the top CUA operating functions shown in Figure 10 are less than 100 lines of code each. The code for files, screens, printing, stack and host commands, with comments removed, takes about 50 kB of workspace on the PC, and 70 kB on VM. Speed of windowing is also an important measure for these kinds of functions. When COMET runs on a current generation basic PC such as a 386 IBM PS/2® Model 70, 16 MHz, no math coprocessor, VGA color display, most windows pop up within 1-3 seconds. Considering that these functions were developed by one programmer at the application rather than system level, and fit comfortably in a workspace, this performance is quite good.

Finally, COMET users have reacted positively to the user interface and the availability of a PC version. This reflects favorably on both the CUA concept, and the portability and capability of APL used in this application.

TRADEMARKS

CITED REFERENCES AND NOTES

1. "Le Système Internationale d'Unités (SI)." Bureau Internationale des Poids et Mesures (BIPM), Paris, France.
2. Lockheed Missiles & Space Company, Inc. has adopted the Department of Defense permitted policy exception of using the international spelling of metre and litre with "re."
3. "Metrics Policy for the Strategic Defense Initiative." Memorandum from USAF Lieutenant General Abrahamson, Director, Strategic Defense Initiative Office, Department of Defense, to Under Secretaries of Defense, et al. (3 November 1987).
4. "Omnibus Trade and Competitiveness Act of 1988." Public Law 100-576 (signed 23 August 1988 by President Reagan), Section 5164, "Metric Usage."
5. "Practice for Use of the International System of Units (SI) (the Modernized Metric System)," ASTM Standard E 380. American Society for Testing and Materials, Philadelphia, Pennsylvania. This standard has been approved for use by agencies of the Department of Defense, and is specified by "Military Standard Engineering Drawing Practices," standard DOD-STD-100, Department of Defense, Washington, D.C.
6. "Preferred Metric Units for Aerospace," National Aerospace Standard NAS 10001. Aerospace Industries of America, Inc., Washington, D.C.
7. Richardson, H.L. "DOD Strategic Defense Initiative Programs Go Metric; SI Metric System at LMSC." Issue topic "Metric, Standardization, and Global Markets: The Implications for U.S. Industry." *MAPI Economic Report*, Manufacturer's Alliance for Productivity and Innovation, Washington, D.C., No. 144 (August 1989).
8. "COmputerized METrication (COMET) program eases metric conversion at Lockheed." *Metric Reporter*, American National Metric Council, Washington, D.C., Vol. 17, No. 5 (August/September 1989).
9. *Systems Application Architecture: Common User Access Panel Design and User Interaction.* IBM manual SC26-4351 (first edition December 1987).
10. *APL2 Migration Guide.* IBM manual SH20-9215 (third edition November 1987).