# A message management system for personal computers

by L. d'Arielli

***This paper presents a design for a message management system that reduces the coding effort for the application developer, gives the user greater control over the treatment of application messages, and eliminates many problems of translation. Any message may be directed to one or more devices (screen, printer) and/or files (log, activity), with the ability to exclude message elements (e.g., date and time) from being sent to any output destination, or to exclude a destination altogether. Because the message-handling code and message texts are separated from one another and from the application code, the developer need only issue a message identifier and set the associated variable values. The message identifier contains codes for both class and severity, and the developer provides default selection criteria that the user can modify. These tables of selection criteria provide a simple yet highly flexible means of determining where each message will be sent and in what form. The simplicity of including variable information and the separation of message texts into a master repository (where an information developer or translator can work on them) tend to improve the quality of messages to the user by making them more consistent and informative.***

Since the introduction of the first personal computer, many thousands of people have started using computers who are not, and have no wish to become, experts in data processing. This fact has many implications in software design, not the least of which is the need for better messages from the computer (or, more accurately, the program) to the user. These messages give the user information ("OK, I've done it."), or warnings ("I can't do it, or the results may not be reliable."), or announce an outright error ("I tried to do it, but it went wrong."). The old generation of users, who tended to be data processing specialists, was prepared to accept cryptic and critical messages. The new generation expects a message to provide help without the need to search a manual for an explanation.

**The problems of message handling.** In the most frequent situation, in which the user is working interactively with the computer, the user does at least have the advantage of being able to see the situation at the moment the message is issued. In an event-driven situation, such as when a personal computer is receiving data from an international network, much more information is needed to tell the user what to do or what has happened.

As an example, take the case of simply copying a file from a fixed disk to a diskette on an IBM Personal Computer or Personal System/2® (PS/2®) using a command of the Personal Computer Disk Operating System (DOS) in an interactive situation:

```
copy c:daniela.dat a:emanuela.dat
```

In this situation, messages such as:

```
1 file(s) copied
```

```
File not found
```

```
Disk full
```

are, perhaps, sufficient. The context is known, the command remains on the screen, and lengthier messages could soon become tedious. The user does not *have* to be told:

```
File c:daniela.dat not found
```

In contrast, in the case in which networking is being used, a message such as:

```
Disk full
```

is totally inadequate. The user needs something like:

```
Disk A: full, unable to write file EMANUELA.DAT
received from TIMBUKTOO at 04:12:36 GMT
```

or, for example, in Italian:

```
Disco A: pieno, l'archivio EMANUELA.DAT ricevuto
alle 04:12:36 GMT da TIMBUKTOO non viene scritto
```

This example highlights two requirements: the inclusion of *variable information* and the need to provide *National Language Support* (NLS).

Developers too have their problems. For a particular application, they must decide whether messages may be displayed, printed, or recorded in a file. Then they must decide which destinations are appropriate for each individual message. And finally, they must write the code to assemble and issue each message to each destination.

All of this effort is spent to produce a totally inflexible system: The developer must dig into the code in order to implement any change in destination or format, while the user can do nothing at all to alter what the developers have provided.

**A solution to these problems.** This paper describes how the problems of providing messages were solved when designing and implementing the IBM Personal Computer/Distributed Systems Node Executive (PC/DSNX) at the IBM Telecommunications Development Center in Rome, Italy. Included with this product is a system for handling messages called the Message Management System. (The similarities with the message management of Operating System/2™ [OS/2™][1,2] are coincidental; PC/DSNX was developed totally independently.)

Separating message texts from application code is not new. It is virtually indispensable if messages are to be translated. But the Message Management System described here goes further, separating the message-handling code from the application code. It has been largely implemented as a component of Release 1 of PC/DSNX, which has been available since March 1988.

**Message texts are called skeletons.**

The application code communicates with the message-handling code through an interface control block and is itself controlled by tables defined by the customer[3] when installing the package. This approach has a number of advantages for the developers and translators of the product and also gives the customer considerable control over the selection and routing of messages. It should, perhaps, be stressed at the outset that it is *not* intended that the customer should modify the basic message texts.

The logical starting point of the description is the *master message repository* containing all of the message texts. The use of this file makes it easier:

- To keep the wording and contents of messages consistent
- To avoid duplicating messages
- To keep messages up-to-date
- To translate messages into foreign languages

The repository is used to create, more or less automatically, two other files: the text formatter, Script, source file for the messages section of the manual and the *message skeleton file.* All three files are text files that developers and translators can view and change with an ordinary personal computer editor, but clearly it is almost always easier, and invariably safer, to modify message texts in the repository file.

The message texts are called *skeletons* at this point because they contain consistent wording and the *names* of any *imbedded variables*. The *values* of the variables put the "flesh" on the "bare bones" of these texts.

**How a message is issued.** Given the existence of the message skeleton file and the Message Management System (MMS), the application code initiates a message simply by putting the module name and the values of any associated variables into an area accessible to the MMS and issuing a message request consisting of the message identifier (for example, "PXM3004E").

From this point, the MMS takes over. First, it assembles the full message text, combining the skeleton text and the values of any imbedded variables and adding the module name, the date and time, and the time zone. These last items may be required for diagnostics, tracking, and audit. The time zone is of interest only if the application involves communication across time zones.

The full message text is then made available to a user exit, permitting the customer to write a small program to make any necessary modifications. For example, the values of any variables may need to be manipulated for a language written from right to left.

On regaining control, the MMS consults the *message selection criteria* contained in the message customization file that customers can tailor if the defaults do not satisfy their needs. These criteria are the fan-out vector, the filter matrixes, and the format vector.

The *fan-out vector* simply determines what possible destinations are available. (Screen, printer, log file, and activity file are the defined destinations in PC/DSNX.) A customer who does not want an activity file at all says so by modifying the fan-out vector.

The MMS now looks at the *filter matrixes* for the remaining destinations. Every message identifier contains a class code, the meaning of which is application-dependent, and a severity code (Information, Warning, or Error). The filter matrix for a destination determines, for each combination of class and severity, whether or not the message is required.

At this point, therefore, the MMS knows that the message request affects only certain destinations, for example, the screen and the log file.

The MMS now consults the *format vector*, which provides the reference number of the output format used for each destination. An output format simply consists of a list of the message elements (for example, application module name, message identifier, message text, and its imbedded variables, date and time) to be included in all messages to the destination concerned. The output formats themselves are predefined in the application and cannot be modified by the customer.

Finally, the MMS makes a copy of the message for each destination that has not been eliminated in the previous stages, removing unwanted message elements according to the output format(s), and issues the message(s) to the destination(s).

## Message Management System rationale and overview

The first decision was to build the message skeleton file, which contains all of the messages, as a "flat" text file, written with use of any personal computer editor. In this way it is easily displayed, printed, and translated. Each message skeleton has its unique message identifier and can imbed the names of one or more variables. The variables will be resolved when the message is issued.

Another decision was to store the message in such a way that, when it has been resolved, a user program can extract whatever information is of interest, such as values of resolved variables, message identifier, module name, date, and time.

The call to a user exit routine is provided after the resolved message has been built in memory and before it is issued. This ordering allows the customer to control the message handling by extracting information for statistics in real time, by taking over the message management entirely (fanning out, filtering, and formatting), or by modifying the text of the resolved message before it is issued.[4]

The message customization file gives the customer or end user a simple and effective means of controlling how messages are handled. The message selection criteria allow different types of messages to be passed to different destinations, and there are also facilities for adding information provided by DOS (for example, date and time) or by the application. The personal computer operating system described here is DOS, but the same approach, with some modifications to the implementation, would be valid for OS/2 or any other operating system.

All of the messages are categorized into classes and severities, and both of these items of information are encoded in the message identifier. These two categories provide a basis for the logical differentiation that controls filtering. The current implementation of these message selection criteria supports two devices (screen and printer) and two files (log file and activity file).

Figure 1 shows the overall relationship between the application and the Message Management System (MMS).

It is the application developer's responsibility to build the message skeleton file, providing for each message the appropriate identification, the message text, and the names of any imbedded variables. To ensure that the name of a variable always has the same meaning, the developer also needs to provide the MMS with a message variable table (MVT) that lists all of the variables with the address in the message boundary block (MBB) in which the value of each variable will be placed. This MBB is the control block used to exchange all of the request and response information between the application and the MMS.

With the names of the variables and related information separated into the MVT and then copied into the MMS area, it becomes easier to issue an error message from the application program, because there is no need to provide both the names and the contents of the variables each time a message is issued. Only their contents are necessary.

At initialization time, at least the MVT, the name of the message skeleton file, and the name of the message customization file are passed to the MMS by the application. The MMS is thus able to build, in memory, the message skeleton index to allow keyed access to the message skeleton file, to load into memory the message selection criteria from the message customization file, and to set up appropriate values in the message control block.

From now on, in order to issue a message, any module of the application has only to provide the message identifier, its module name, and the values of the variables related to that particular message. On the basis of this information and that in the internal control areas, the MMS reads the skeleton message from the message skeleton file, obtains the values of the variables and substitutes them in the message text, and builds the resolved message to be passed to the user exit routine. When MMS regains control from the exit routine, it makes use of the message selection criteria. Only if the fan-out vector and the filter matrix let it through will the message be arranged in the user-defined format and sent to the corresponding device or file. At the end of this process, the MMS puts the return code and the condition code vectors resulting from the operation in the MBB for the use of the application program.

**A variable always has the same name in all of the messages in which it appears.**

**Data integrity.** The MMS checks the validity of the files at startup. However, direct editing of the files by the customer could introduce inadvertent errors and is therefore discouraged. To persuade the customer not to edit the message customization file directly, PC/DSNX provides an interface program for customization. This program leads the customer through a series of interactive panels and checks the validity of any changes made.

If it is felt necessary to protect the message customization file and the message skeleton file from any attempt by the naive user to edit them directly, they can easily be made read-only files.

### Message skeleton structure

The message skeleton file can be created using any personal computer editor that is capable of editing a "flat file" of variable record length. Each message skeleton occupies one or more lines in the file. The beginning of a message is marked by a new message identifier starting in column one. The file format is, therefore:

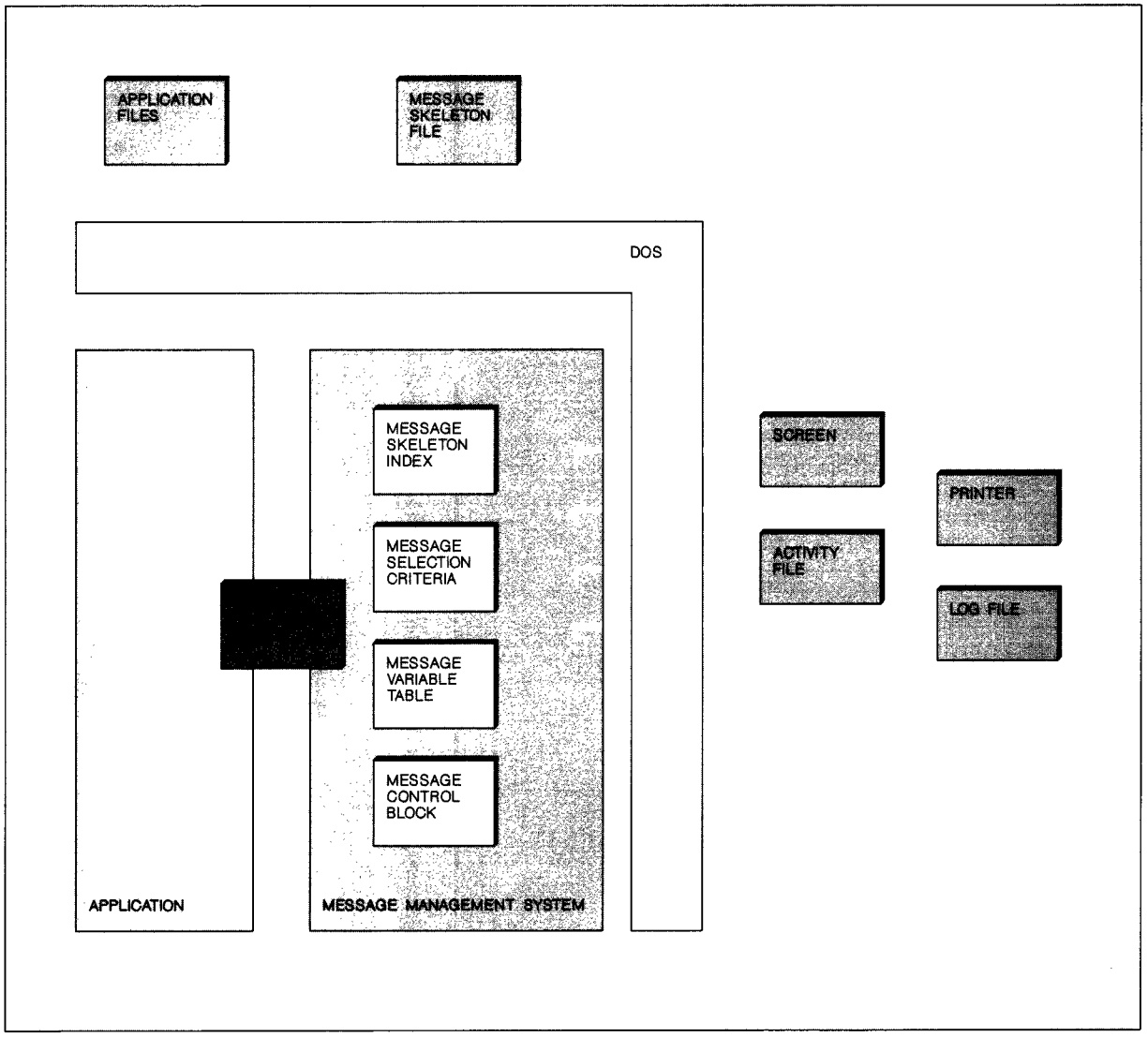
```
XXXcnnns Text of message, with KW=&VARNAME
imbedded

...

keywords and variable names

XXXcnnns Text of next message
```

**Figure 1 General structure and interfaces**




Two key points of the MMS are the structure of the message identifier "XXXcnnns" and the handling of imbedded variables.

The message identifier includes two characters that describe the nature of the message, namely the message class "c" and the message severity "s." This format imposes a discipline on the developer to categorize all of the application messages into classes and severities. The message class is a way to subdivide all of the application messages into categories that are meaningful to the end user according to the type of their content. The message severity, on the other hand, classifies the messages according to the level of intervention that the application requires from the user. On the basis of the combination of these two codes, a more granular and still simple filter grid can be built. This granularity is sufficient for most applications.

The naming of variables is the other key point. A variable always has the same name in all of the messages in which it appears (for example, &FILEID for file name), and the application program always

places the value of a variable in the same location in the MBB. This approach (as opposed to the ten numbered variables described in References 1 and 2, where the file name might be %1 in one message and %4 in another) has several effects:

- The name of the variable is meaningful.
- There is virtually no limit to the number of variables that can be imbedded in a message. The only constraint imposed by the MMS at present is the maximum message length of 32 kilobytes.
- It is much easier to translate the message skeleton file into any foreign language (which might require variables to be used in a different order).
- It is much easier to create an accurate list of messages for inclusion in the documentation.

The named variables are recognized, among the text words, through the <SOV> (Start Of Variable) character—in the example above, the ampersand (&). Once the MMS has replaced the named variable with its corresponding application value, there is no easy way to recognize the value of the variable among the words of the resolved message text.

This situation could make it difficult for a user program to extract the values of the variables. To overcome this problem, therefore, a keyword is put in front of the named variable, for instance "FN=" meaning "File Name." These keywords are part of the skeleton text and are not recognized by the MMS as having any special significance. Only the user program needs to know them, so they must be documented in the user manual. They can also be translated just like any other part of the skeleton text, whereas the message *identifiers* and the *names* of the variables must not.

Below, a detailed description of the skeleton message, with the actual values provided in the PC/DSNX application, provides a practical example that may make these concepts clearer.

XXX — Software Product Message Acronym (three characters). The capital alphabetic string "PXM" is used for messages pertaining to the PC/DSNX application.

c — Message class (one character). Allowed values are 0–9 and A–Z. The PC/DSNX application implements only the following classes:

- 0 Interactive messages, including prompts
- 1 Host request activity messages
- 2 Transmission error messages
- 3 I/O error messages
- 4 DOS or PC/DSNX program logic error messages

nnn — Message number (three characters). Messages are numbered in a separate series for each class.

s — Message severity (one character). The following values, with the corresponding meaning, are implemented:

- I Information
- W Warning
- E Error

text — Message skeleton text with imbedded variables. Message variables inside text have the following format:

<SOV> <Varname>

SOV — Start Of Variable identifier, a unique ASCII character, defined by the developer, that identifies the name of a variable. It must not have any other meaning anywhere in the message skeleton file. It may be any ASCII character in the hex range 21 through 7F except the alphanumeric characters (0–9, a–z, A–Z). The PC/DSNX default is "&."

The translator of the messages might possibly need to change the SOV, but the customer should not.

Varname — This is the name of a variable that is also defined in the MVT. It has a maximum length of eight alphanumeric characters (0–9, a–z, A–Z).

Any nonalphanumeric character marks the end of the variable name.

### Structure of files and tables

**Message skeleton file.** The message skeleton file is built up of some application-specific information in the first lines followed by all of the message skeletons.

**Figure 2 Format of the message skeleton file—examples of all the message classes are included in this selection**

```
* * * Top of File * * *
PC/DSNX  MESSAGE SKELETON FILE
/* 5669-333 (6476171) (C) COPYRIGHT IBM CORPORATION 1988              */
/* ALL RIGHTS RESERVED - LICENSED MATERIALS - PROPERTY OF IBM         */
PXM0001W Activate the printer.
PXM0002E A number of messages sent to the printer have been lost because
the printer is not active.
PXM0003W Activate drive DR=&DRIVEID.
   .
   .
PXM1007I File FN=&FILEID has been replaced.
   .
PXM1014E File FN=&FILEID was not sent because it was not found.
   .
   .
PXM2003E A SNA Conversation Failure Occurred. VERB=&APPCVERB,
PRC=&APPCPRC, SRC=&APPCSRC.
   .
PXM2006E An unrecoverable error occurred when parsing the MU.
SNAC=&SNACODE, SNAS=&SNACODE, STRID=&STRID, STRDATA=&STRXCPN,
SEGNUM=&SEGNUM, BYTENUM=&BYTENUM.
   .
   .
PXM3001E An unrecoverable error occurred while accessing the file
FN=&FILEID.
   .
PXM3003E Processing cannot continue because disk DR=&DRIVEID
is full.
   .
   .
PXM4005E An unrecoverable error occurred due to invalid data in the
PC/DSNX system file FN=&FILEID.
   .
* * * End of File * * *
```

Figure 2 illustrates an example of the message skeleton file as it is viewed when displayed by a personal computer editor. The first line is the *file descriptor*, which is checked by the MMS. This is followed by the *copyright* information in the form of comments. The remainder of the file consists of all of the *message skeletons* with the imbedded variables and keywords.

For economy of effort and to ensure consistency, the message skeleton file could be generated automatically from a master message repository that is also used to generate the source file for the messages section of the manual.

**Message skeleton index.** The highest performance would be obtained by keeping the entire message skeleton file in memory, but not every system has enough space to do this. Instead, the MMS builds a memory-resident index that gives fast random access to the disk-resident message skeleton file, thus combining good performance with economical use of memory.

As an example of relative size, the PC/DSNX message skeleton file, containing about 120 messages, is 10 200 bytes long, whereas the message skeleton index takes only 1680 bytes of memory.

At startup time, the MMS sequentially reads the whole message skeleton file and dynamically builds the message skeleton index, with one entry for each message containing its relative byte address and

**Figure 3 Example of message output formats**

| CONTAINS: | MESSAGE OUTPUT FORMAT | | | | |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |
| MESSAGE IDENTIFIER | Y | Y | Y | N | N |
| MODULE NAME | Y | N | N | N | N |
| DATE AND TIME | Y | Y | N | Y | N |
| TIME ZONE (TZ) | Y | Y | N | Y | N |
| MESSAGE TEXT | Y | Y | Y | Y | Y |

length. Thereafter, the message skeletons are read directly from disk by key (the message identifier) only when they are needed.

A small reduction in startup time could have been achieved by creating the message skeleton index once only, as can be done by a utility.[1] It was decided, however, that with the comparatively small message file of PC/DSNX, it was better to play safe in maintaining the accuracy of the index while reducing dependencies.

The user with ample memory can improve performance by copying the message skeleton file onto a virtual (RAM) disk. Performance in general can also be improved by faster reading of all files, obtainable by increasing the number of DOS buffers in the CONFIG.SYS file, and/or by making use of the IBMCACHE program on the PS/2 reference diskette.

**Message boundary block.** The message boundary block (MBB) is a control area used as the interface between the application and the MMS. The application issues all message demands via this control area. It has room, at least, for the message identifier, module name, contents of all of the variables, and return codes and condition codes.

**Message variable table.** The message variable table (MVT) is an internal table with as many entries as there are different variables named in all of the skeleton messages. Each entry contains the variable name, the address and length of the variable value, and the data transformation to be provided on that value (for example, no translation, translation from binary to decimal or hexadecimal printable format, and so on). It represents a bridge to translate the values of the variables from the MBB into the message to be issued.

**Message control block.** The message control block (MCB) is an internal area containing information to make the MMS work properly in any environment. It contains the pointers to all of the other control blocks, along with certain other information.

The screen environment type tells the MMS whether to display the resolved and formatted message or to copy it back to the application buffer, in which case it will be the responsibility of the application to display it.

For National Language Support, the skeleton text code page and the date and time format enable the MMS to represent the resolved message text and date and time in the proper way.

Finally, the time zone allows local times to be expressed on a common base in a network involving communication across time zones.

**Format of the resolved message.** After the specific message skeleton has been filled with the values of the variables, three *message-independent items* are added. One of these items, the module name (the component of the application that issued the message), may be required for diagnostic purposes. Another is the date and time, required for logging and auditing purposes. The third item is the time zone, only required for networking applications. The first two items are available only at the moment when the message is issued, whereas the time zone is recorded by the customer on installation.

The message string so generated consists of the following five items:

- Message identifier
- Module name
- Date and time
- Time zone
- Text of the message, with all variables resolved

Different message formats can be built by selecting different combinations of these five elements (but without changing their order). The message text, of course, should always be selected.

The PC/DSNX implements only the five most-used combinations, coded as message output formats 0–4 and shown in Figure 3. The format of the fully resolved message, as it is passed to the user exit routine, is given in Figure 4.

**Figure 4 Format of the fully resolved message (with lengths of fixed fields)**

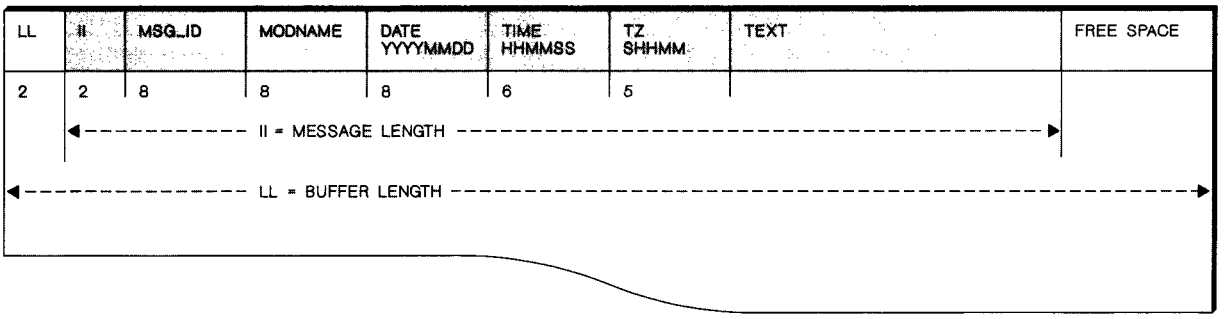



**Figure 5 The message selection criteria—data are contained (in vector format) in the message customization file**

FAN-OUT VECTOR (GENERAL)

| DEVICE/FILE | SELECTION |
|---|---|
| SCREEN | Y |
| PRINTER | N |
| LOG FILE | Y |
| ACTIVITY FILE | Y |

FILTER MATRIX (ONE FOR EACH DESTINATION)

| SEVERITY | CLASS | | | | |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |
| I | N | Y | N | N | Y |
| W | Y | N | Y | N | Y |
| E | Y | N | Y | Y | Y |

FORMAT VECTOR (GENERAL)

| DEVICE/FILE | FORMAT |
|---|---|
| SCREEN | 2 |
| PRINTER | 1 |
| LOG FILE | 0 |
| ACTIVITY FILE | 1 |

The date, time, and time-zone fields are automatically expanded in a later stage, according to instructions held in the message control block.

## Considerations for National Language Support

To minimize the design problems of the Message Management System without losing generality, it was decided to code as Single Byte Character Set (SBCS) Left-to-Right items the message identifier, module name, date, time, time zone, and variable names. The message skeleton text, with the exception of the imbedded variables, can be coded in any language, whether SBCS or Double Byte Character Set (DBCS).

Note that only items that are not translatable are used as variables. Examples are items such as file specification, return code, drive name, etc. A variable will never contain a natural-language word.

## How message selection is performed

Once the message has been resolved and the message-independent items have been added, the user-defined message selection criteria are applied to determine to which destinations the message should be sent, and in what format. As described earlier, the message selection criteria consist of three components: the fan-out vector, the filter matrixes (one for each destination), and the format vector.

Figure 5 shows, in tabular form, an example of the data that may be held. In the fan-out vector and the filter matrix, where Y stands for yes, the message may be passed to the corresponding device or file, and where N stands for no, the message must *not* be passed to the corresponding device or file. In the format vector, the numbers are the message output formats as defined in Figure 3.

The fan-out vector allows the customer to totally exclude a destination that is not required. Figure 5

shows that the customer has chosen never to direct messages to the printer. The fan-out vector is a binary vector with one position for each possible destination.

Once the message has passed the fan-out vector for a destination, it has to pass a specific filter matrix for that destination. The filter matrix allows the customer to choose, for each supported destination, whether or not the message should be passed, based on the class and severity of the message. The filter matrix is held as a two-dimensional binary matrix.

When both the fan-out vector and the appropriate filter matrix have let a message through, it will reach its destination. It only remains to format the message in accordance with the format vector.

Formatting allows the customer to choose how many of the elements of a message are required at each destination. Like the fan-out vector, the format vector has a global effect: Whatever message output format is specified in the format vector for a particular destination, that format applies to *all* messages directed to that destination.

The format vector has as many components as there are destinations defined.

**An example.** Figure 6 is a schematic representation of how a message is built and sent to devices and files. Suppose that an application module, named, for instance, APXAMAN, issues the error message whose message identification is PXM2003E, and suppose that the message skeleton text is as follows:

```
PXM2003E A SNA Conversation Failure Occurred.
VERB=&APPCVERB, PRC=&APPCPRC,
  SRC=&APPCSRC.
```

The application module has to store the following into the message boundary block:

- The message identifier PXM2003E
- The module name APXAMAN
- The values of the variables named in the skeleton, which give sufficient information for a correct diagnosis

It then calls the Message Management System.

Building of a message by the MMS can be seen in Figure 6. The numbers in parentheses here correspond to the numbered steps in the figure. MMS (1) retrieves the message identifier and the values of the variables that have been passed into the message boundary block and (2) reads the message text from the message skeleton file. Then it (3) asks the operating system to get the date and time and (4) obtains the time zone value that was passed into the message control block at initialization time. Suppose that this is minus five hours and zero minutes (U.S. Eastern time zone). In the example, the values of variables, as can be seen in (1), are the APPC verb &APPCVERB=ALOC, the primary return code &APPCPRC=0003, and the secondary return code &APPCSRC=084C0000.

At this point the MMS takes two actions. First, it resolves all the message variables, replacing the variable names &APPCVERB, &APPCPRC, and &APPCSRC with the corresponding variable values ALOC, 0003, and 084C0000 (1). Next, it inserts all of the message-independent items, APXAMAN as the module identification (1), 19880325 and 152347 (3:23 p.m. and 47 seconds on March 25, 1988) as date and time (3), and -0500 as the time zone (4).

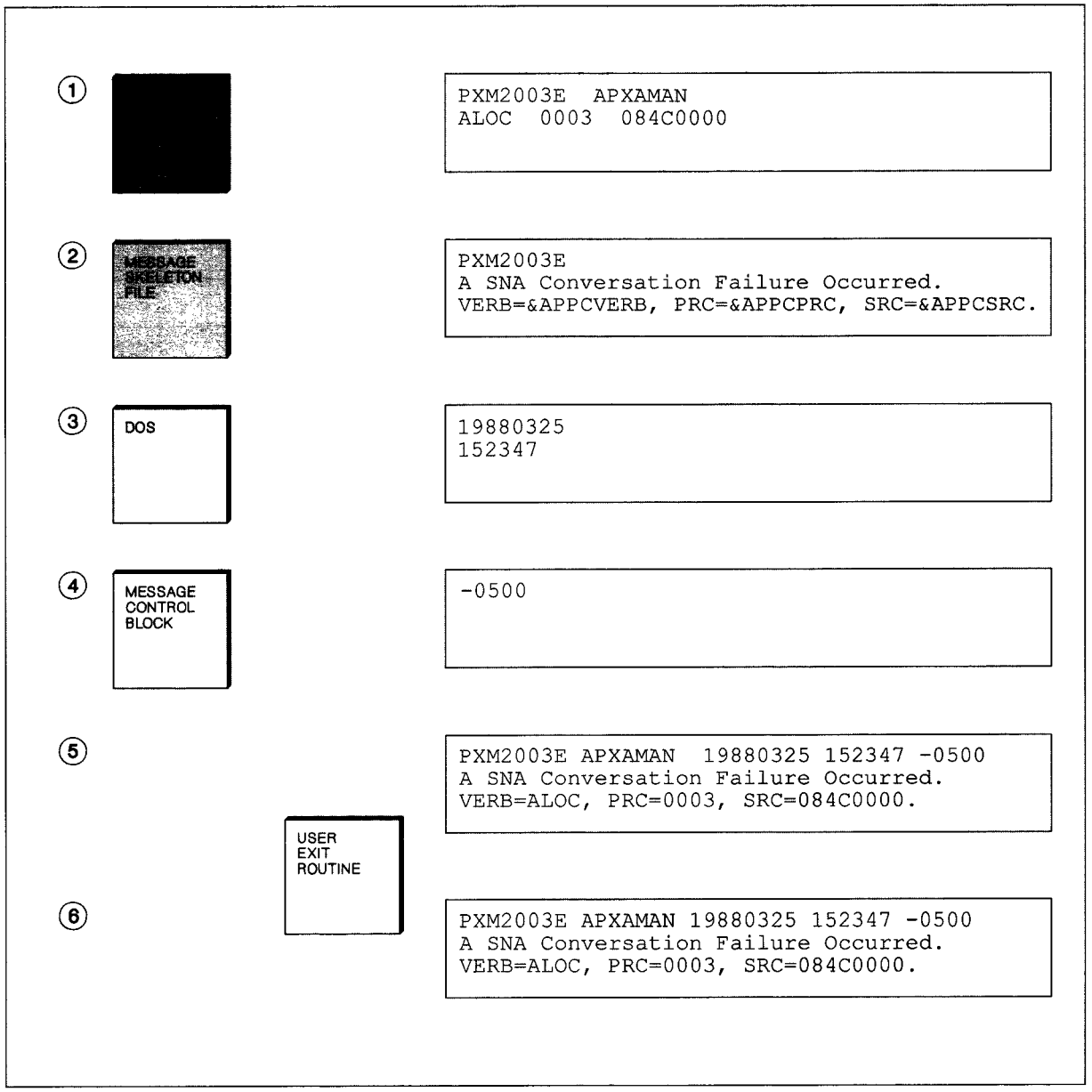
Once the message has been resolved (5), it is passed to the user exit routine if there is one. The format of the resolved message has already been shown in Figure 4. To allow for the possibility that the routine might increase the length of the text, the message is passed in a buffer with some free space. The first two fields specify the full buffer length and the length of the complete resolved message. The user exit allows the user to manipulate the message text, overwriting the original text and expanding it within the limits of the free space (and modifying the value of "ll"—message length—if necessary). It would be highly undesirable for the user exit routine to tinker with the other fields.

Suppose a user exit routine exists that simply puts the first letter of each word in uppercase. Then, on return, a slightly different message is obtained (6). (This very unrealistic example merely illustrates the fact that the user can modify the resolved message text. A realistic example would over-complicate the description.)

The MMS then applies the message selection criteria to this message, formatting it appropriately for each valid destination.

Figure 7 shows how the message is fanned out, filtered, and formatted for each device and file. Again, in the following discussion, the numbers in parentheses correspond to areas in the figure.

**Figure 6 How a message is built; processing continues with the message selection criteria in Figure 7**




In the overall picture of the cascade of vectors and matrixes, only the values relevant to class 2, severity E, are shown in the filter matrixes to make it easier to follow the example.

The fan-out vector directs the MMS to send or not to send messages to the corresponding device or file. As can be seen, the fan-out vector prevents any message from being forwarded to the printer. (The effect of this is that the filter matrix for the printer is never referred to. Note that the "Y" in the 2E cell therefore has no effect until such time as the fan-out vector might be changed.)

The message in the example is of class 2 and severity E, and this is the key information for filtering. The

**Figure 7 The message selection process**

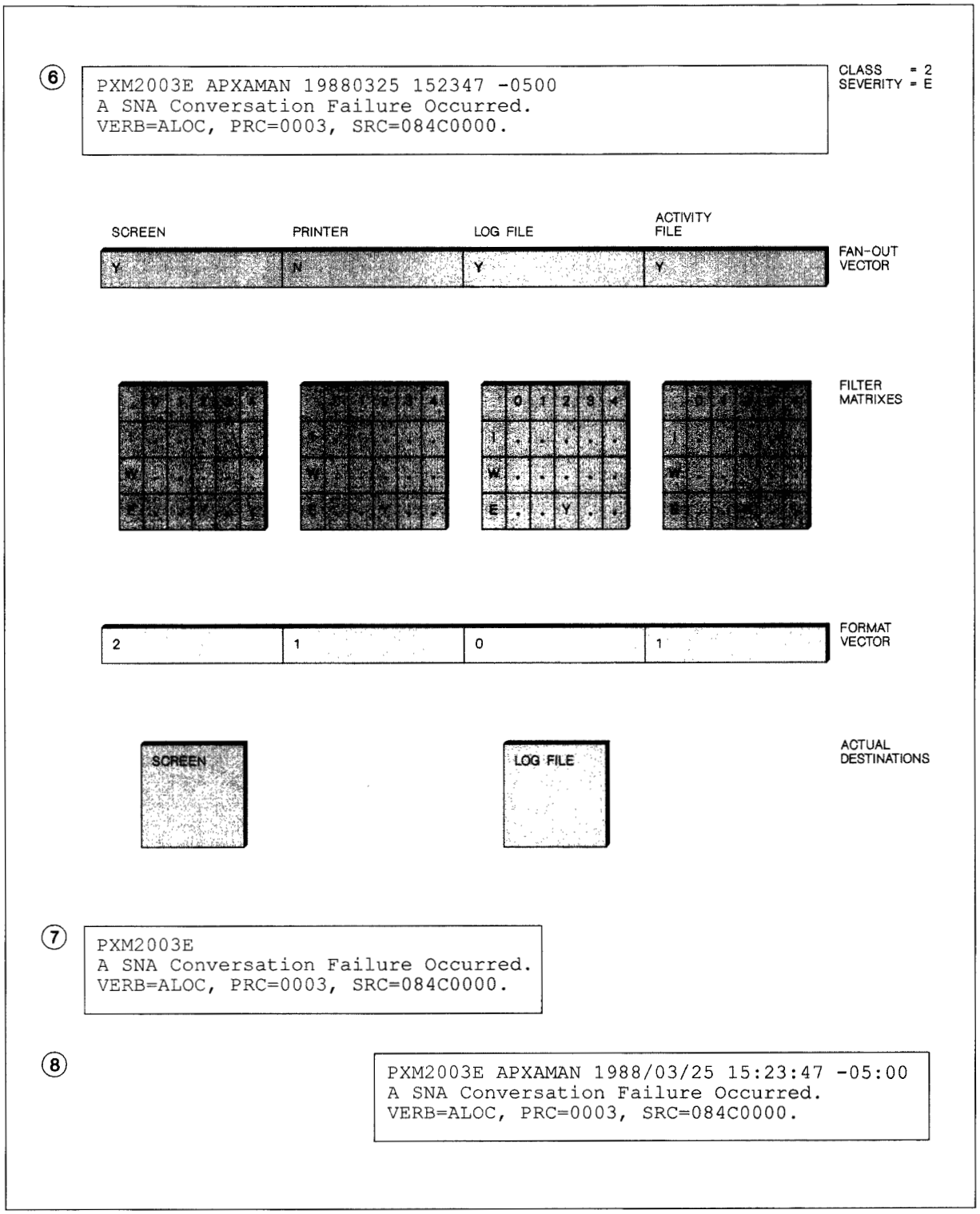
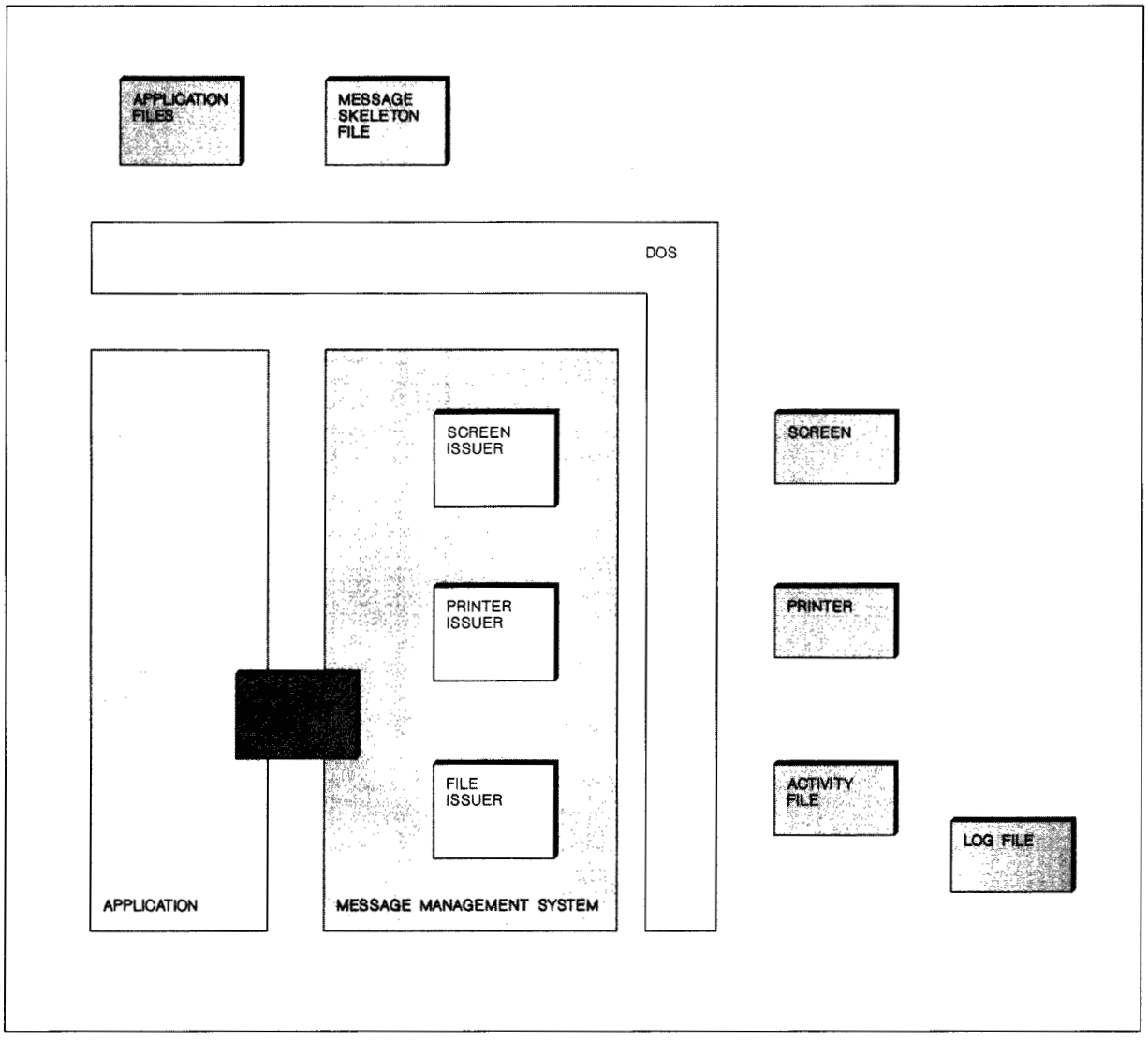


**Figure 8 General input/output interfaces**



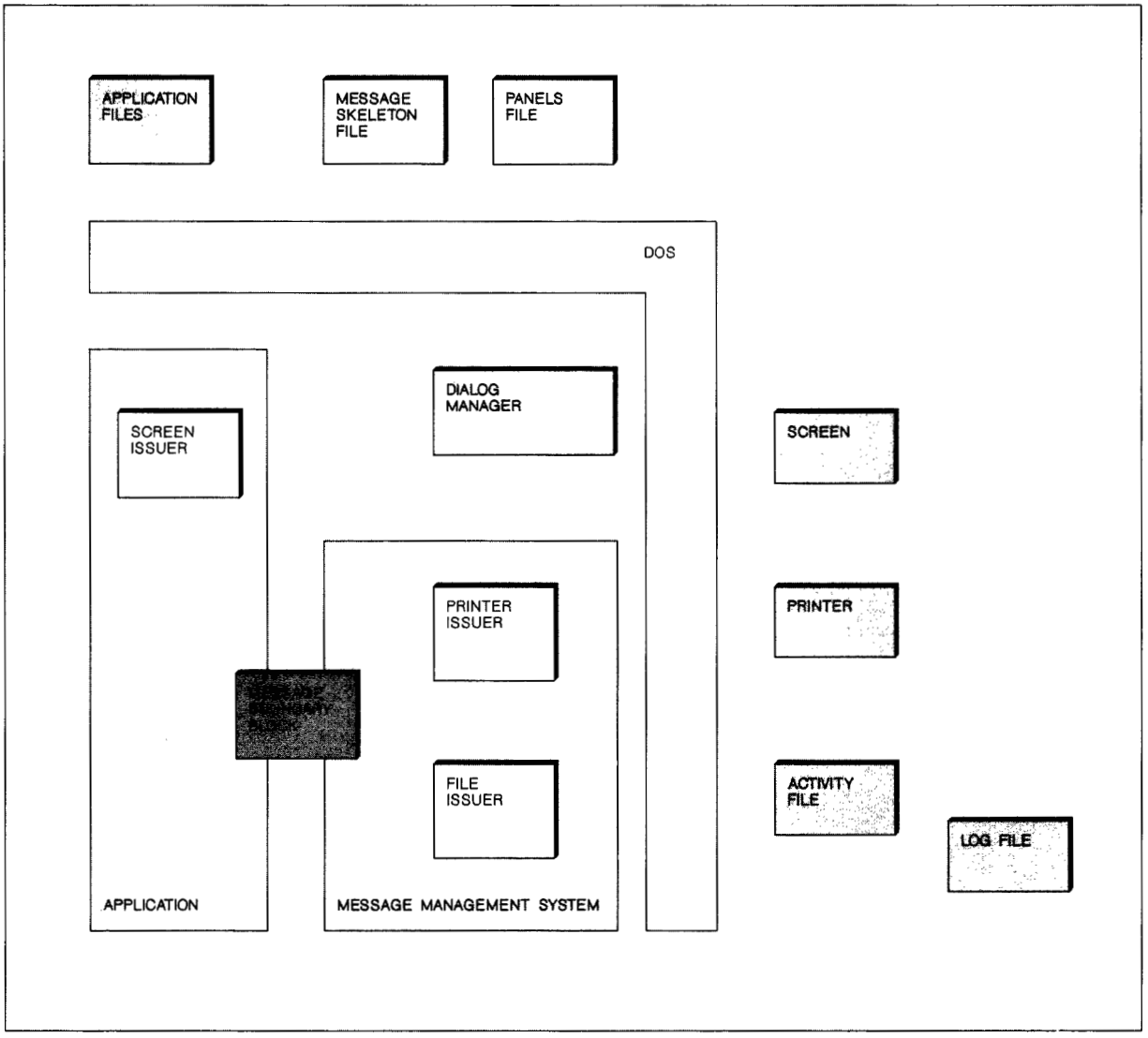

filter matrix for the activity file has an "N" in the 2E cell, which blocks any further progress of the message on that path.

The filter matrixes for the screen and the log file, in contrast, contain "Y"s in the 2E cells, so the message is passed on to be formatted as indicated by the format vector—that is, output format 2 (7) and 0 (8), respectively.

**An application using a dialog manager.** Because most interactive applications will run under a dialog manager, it was thought useful to implement the MMS in such a way that it can either interface with the display device directly or return the resolved message for the screen to a buffer provided by the application. Which of these two modes is active depends on the screen environment type value chosen.

When the messages are to be displayed by the MMS, it will use the normal DOS Function Call to manage the screen (see Figure 8). When the messages are to be displayed by the application, via the dialog manager rules, the MMS will return the resolved message

Figure 9 General input/output interfaces under a dialog manager




to an application buffer, and the application will then issue the message to the dialog manager (see Figure 9).

In the PC/DSNX, this screen interface has been implemented to comply with the IBM EZ-VU II program product.

## Concluding remarks

The Message Management System is a simple and effective way to manage application messages. The application software does not have to be concerned about where the subject message has to be sent. It has only to provide the message identifier, its own module name, and the variable values, and issue the message.

The user-defined message selection criteria direct the MMS to forward the message to the required destinations (devices and files) with the chosen formats.

A master message repository can be used to generate both the message skeleton file and the Script source

file of the messages and codes section of the user manual for the application. This repository ensures consistency between the text used by the program and the messages documented in the manual.

The MMS can support any language on the National Language Support (NLS) list. It is only necessary to provide a translated message skeleton file (probably generated from a master message repository) and the corresponding code page indication. During translation, the file descriptor, message identifiers, and the names of variables must *not* be translated. These fields will be built into the resolved message in SBCS Left-to-Right, as will the message-independent items such as date and time. In general, from an NLS point of view, the message texts will be in a mixed format.

Keywords are imbedded in the text immediately before the names of variables. This action makes the resolved message manageable by a user program, which can easily extract the variable values. A user can write a user exit routine to be used in real time, or a stand-alone program to scan the activity and log files after the application has ended.

If several message skeleton files in different languages are provided, different languages can be used in different runs. The only restriction is that the subject languages must be compatible with the personal computer hardware.

The already very good performance can be improved by copying the message skeleton file to a virtual disk in memory. A more general performance improvement can also be obtained with the use of more DOS buffers and/or of IBMCACHE.


## Acknowledgment

I would like to express my appreciation to Michael Morgan for his generous assistance in reviewing my English and making my wording and figures clearer.




## Cited references and notes


1. *IBM Operating System/2 Programmer's Toolkit*, PN 6280200, IBM Corporation; available through IBM branch offices or authorized dealers.
2. *IBM Operating System/2 Technical Reference*, Vol. 1 and Vol. 2, PN 6280200, IBM Corporation; available through IBM branch offices or authorized dealers.
3. Depending on the application, the "customer" may be the end user or a single person installing the package for a number of end users.
4. This procedure also complies with the National Language Support requirement to have in the Single Byte Character Set for Right-to-Left Subset an "Exit for Automatic Shape Determination (ASD)" which is in charge of adjusting the text encoding to be correctly displayed or printed.



**Luigi d'Arielli** *IBM Italy, Field Systems Center, P.le dell'Agricoltura 24, 00144 Rome, Italy.* Mr. d'Arielli obtained a Laurea (equivalent of a Master's degree) in nuclear engineering from the Polytechnic of Turin in 1969. He joined IBM the following year, and until 1972 he worked at Pisa University developing an administrative teleprocessing network. From 1972 to 1977, he was responsible for the development of a large network for the Italian Social Security Institute in Rome. There he was also responsible for development, with the customer, of application software to improve user-monitoring and network control. In 1977 he went to IBM Italy's Field Systems Center and was involved in distributed data processing support in the area of large networks. Mr. d'Arielli joined the Program Product Development Center in Rome in 1980 and participated in the design and development of Distributed Systems Executive (DSX), with a close involvement in its usability testing. He then transferred to the newly formed Telecommunications Development Center in 1986, where he worked in the areas of design and project management for PC Distributed Systems Node Executive (DSNX) products and in SNA/DS architecture. In February 1989 he returned to the Field Systems Center, where he is currently involved in supporting interconnected CICS systems.



Reprint Order No. G321-5371.