INTRODUCTION

This paper discusses developing C language application programs that perform full-screen terminal input and output on IBM® mainframes. The architecture of IBM terminal communication and the difficulty of interfacing to this architecture is presented. A library of functions designed to be called from C programs that addresses these problems is introduced, and examples of their use are given.

Unlike some other operating systems and hardware, IBM mainframes and terminals do not support single-character unbuffered terminal I/O. The common IBM 370 terminal architecture (the 3270 Information Display System) instead maps each character on the display to a storage area known as a character buffer in the device. The application program uses commands to send or receive data from this buffer. The following overview is based on information from IBM documentation.

Overview of the 3270 Data Stream

The data transferred between the terminal and an application program is referred to as a data stream. An application program displays information on a terminal by constructing a data stream and sending it to the device. Input is obtained by reading the terminal and decoding the data stream that is returned. How the data stream is transmitted is discussed later in this paper. The data stream can be made up of commands, orders, attribute characters, and data.

Commands

Commands direct the operation to be performed and must be the first byte of the outgoing 3270 data stream. Some common commands are

- **Write** sends the data stream that follows to the terminal.
- **Erase and Write** clears the screen before writing any data.
- **Read** obtains input from the terminal.
- **Read Modified** only returns data that were altered in some way; this can reduce the amount of data being transferred.

Orders

Orders direct the formatting of the display, much like a print format control character in a programming language. More than one can appear in the data stream and indicate additional functions to be performed as part of the command. They can be present in either outbound or inbound data streams. Some orders are followed by additional information such as buffer addresses or attributes. The following are commonly used orders:

- **Set Buffer Address** an order followed by a two-byte encoded address that specifies the row and column coordinates on the display that data are written to or read from.
- **Start Field** an order always followed by a field attribute byte specifying the start of a field of data.
- **Start Field Extended** an order similar to the Start Field order but used for terminals that have extended features such as color and highlighting. This order is followed by a count field and a number of attribute pairs.
- **Repeat to Address** an order followed by an address and a byte of data to be replicated up to that address.
- **Insert Cursor** an order that specifies the coordinates on the display to position the cursor.

Attributes

Attributes specify properties for a field (and on displays that support them, for a character field or an extended field). Different fields can have different attributes. Each field attribute byte occupies a position in the display's character buffer and on the screen. (Every 3270 display has what is known as a character buffer. Each position on the display maps to one position in the character buffer. Thus, a 24-by-80 character display has a 1920-byte character buffer.) The field attribute is displayed as a blank and cannot be modified by input from the user. If ten fields are defined in the data stream, ten field attributes are required, which means there are ten fewer possible data entry positions on the display. The minimum number of field attribute bytes is one; that is, the whole display is defined as one field. The following are attributes that can be assigned to a field:

- **Protection** indicates whether or not a field can be modified by input from the user.
- **Color** indicates the color of the field (for devices that support color).
- **Visibility and Detectability** specifies if a field is displayed, highlighted, or light-pen detectable.
- **Extended Highlighting** indicates features such as blink, reverse video, trigger fill, and so on. Extended field attributes differ from field attributes in that they appear as a pair of fields in the data stream. A pair is made up of attribute type and value fields (for instance, a type of foreground color followed by a value of red).

Data Stream Format

As mentioned earlier, an outbound data stream always begins with a command and can be followed by orders, attributes, and data. For instance, a typical write data stream looks like the following:

```
write codlWCCordersidatalorders...
```

A Write Control Character (WCC) specifies (among other things) whether to unlock the keyboard or sound the audible alarm.

As a result of a READ command, an incoming data stream resembles the following:

```
AID:cursor addralSBAaddridatalSBA...
```

Data are not returned to an application program until the ENTER key or a function key is pressed. The Attention Identifier (AID) indicates what key was depressed to cause the data transmission. The Set Buffer Address orders specify the coordinates on the display the data came from.
The following is an example of a simple data stream: a full-screen stream looks like the following:

X'CO' Write control character
X'11' Set Buffer Address
X'4ED1' 12-bit coded address (row 12 column 34)
X'1D' Start Field
X'F8' Field Attribute: protected, numeric, intensified display
C'Hello, World' The text to be displayed

**Transmitting the Data Stream**

Once a data stream is constructed, an application program must ask the operating system to transmit it to the terminal. This can be done a number of ways. At the lowest level, the application program can construct the appropriate control blocks and information and use the VM Diagnose instruction or the OS EXCP access method to send the data stream. At a somewhat higher level, the application program can run under a teleprocessing monitor (CMS or TSO, for instance) and take advantage of the interfaces available at this level: the CONSOLE facility of CMS, TPUT, and TGET macros under TSO. When constructing a data stream, an application program must also take into account terminal model dependencies unless the program is restricted to running on only one terminal model. Items that can vary from model to model include:

- Command codes
- Screen sizes
- Extended features supported
- Addressing mode (12-, 14-, and 16-bit)
- Terminal access method being used.

**Other Terminal Architectures**

In 3270 data stream programming, output to the terminal is the result of a command; input to an application program occurs when the ENTER key or a program function key is pressed. In other C programming environments (for instance, a UNIX® or personal computer environment), input characters are typically available to an application program immediately as they are entered. The terminal itself may be equipped with keys that allow characters to be deleted from or inserted into an input string of characters. Often certain special characters when entered into a string of characters serve this same purpose. The application program has to process all these characters as they are input until a field is complete. There are software libraries that exist to simplify this process. For example, the CURSES library in the UNIX environment performs this low-level-field assembly and presents a final string to an application program.

The important point here is that different hardware environments present different problems to be solved. Data streams are very different from single-character unbuffered input and output. The unique architecture of the IBM environment demands an approach that is different from techniques used in other environments.

**THE SAS/C® FULL SCREEN LIBRARY**

From the preceding discussion, you can see that data stream programming is a low-level interface with plenty of function and flexibility; it is also complex, difficult to use, and potentially difficult to maintain. Even IBM does not expect this to be a commonly used interface, as evidenced by products, such as SDF, DMS, and the ISPF panel facility, among others, that offer to remove most of this burden from the application programmer. For these same reasons SAS Institute developed a library of full-screen functions to provide this capability to programs written in C.

The SAS/C Full Screen Library is a set of functions designed to be called by application programs written in C. These functions are intended to isolate the programmer from the specifics of 3270 data stream programming. The functions also allow a more generic or abstract screen manipulation. Using these functions is no different from using any other library function. The appropriate header file is included in the application program, and the function calls are coded. The compiled program must then be linked with the functions that interface with the SAS/C Full Screen Library. These functions are referred to as resident routines. During execution these resident routines call transient routines that perform low-level system-dependent tasks. The resident routines are portable between MVS and CMS; each system has its own transient library.

**SYNOPSIS OF THE FUNCTION CALLS**

The following is a summary of the typical function calls an application program might invoke. These include initializing a full-screen session; defining fields, view areas, and panels; displaying and reading panels; and terminating the session.

Every function call returns a return code to the caller. Successful completion of processing is indicated by a return code of zero. A negative return code indicates an error has occurred, and a positive return code signifies a warning. These codes are documented by #define commands in the SAS/C Full Screen Library header file.

**Terminology**

The following terms are used in this discussion of the SAS/C Full Screen Library:

- **Screen**: the physical display surface of a 3270 device.
- **Field**: the field attribute and text characters up to, but not including, the next field attribute in the character buffer.
- **Panel**: a logical screen whose dimensions are not necessarily the same as the physical dimensions of the device; a collection of fields and their characteristics.
- **View Area**: a rectangular subarea of the physical screen, or a segmentation of the physical screen into which a panel or a portion of a panel is displayed.

The relationship among these terms can be described in the following manner: there is one physical screen, one or more view areas, and one or more panels. There is a one-to-one correspondence of panel to view area. This is a difficult topic to conceptualize initially. The physical screen has a certain size, for example,
Initialization/Termination

The library includes functions to initialize the full-screen environment and to terminate it. Initialization includes acquiring storage for control blocks used internally by the library, determining terminal characteristics, and establishing the environment for full-screen operations. The function calls appear as follows:

```c
int fsinit(), rc;
int fterm();
char * error_buffer;
rc = fsinit(error_buffer);
.
rc = fterm();
```

The address of a 132-byte buffer can be passed by the caller to FSINIT. This buffer is used to return an error message text string on this and any subsequent function call. The termination function returns the terminal to the state it was in before the first full-screen library call, ending full-screen mode. Once a FSTERM call has been performed, the application program must issue an FSINIT call to resume performing full-screen functions.

Defining Fields

The define field function describes the characteristics of a field and provides the library with pointers to the application program-owned storage area for the data value and to the attributes for a field. Any of the usual attributes (color, highlighting, and so on) can be associated with a field. The value and attributes of a field are used by the library when constructing the 3270 data stream. When data are read from the terminal, the application program can examine the field value itself or the Modified attribute bit to determine which fields were returned. This topic is discussed in more detail in Reading Screens and Panels later in this paper.

Field definition function calls take the following form:

```c
rc = fsdfdfn(panel, field_id, beg_col, field_len, data_value, attributes, color, fill_char);
```

Field_id is a unique numeric identifier for the field. The beginning row and column of the field are specified. Data_value is a pointer to the application-owned storage area that contains the value for this field.

Defining Panels and Views

At least one panel and view must be defined. Once defined, they are referenced in subsequent display and read operations. Panels can be defined as any size (height and width), regardless of the dimensions of a particular terminal. To define a panel, a function call is coded as follows:

```c
char * panel;
int height, width;
rc = fsdfpvn(panel, height, width);
```

A view area definition appears as

```c
int viewarea_id, beg_row, beg_col;
rc = fsdfvw(viewarea_id, beg_row, beg_col, height, width);
```

The row and column coordinates here specify the location on the physical screen to position this view.

Return Terminal Attributes

This important function returns a structure containing all the available information about a terminal, including the terminal type, primary and alternate screen sizes, colors supported, and options such as extended highlighting, field validation, and field outlining.

Display a Panel

This function constructs a 3270 data stream from information contained in a panel and sends it to the terminal. A view area must be specified. Recall that a view area is defined as (possibly) a segment of the physical screen. The panel is displayed in the view area specified. A typical call looks like the following:

```c
rc = fsdspn(panel, viewarea_id, panel_row, panel_col, cursor_field, cursor_offset, display_flags);
```

The panel row and column are the coordinates of the panel to be displayed in the upper-left corner of the view area. In other words, if the view area is thought of as a window, you specify where to position the window on top of the panel. Cursor field and offset indicate the cursor position on the display.

Reading Screens and Panels

Reading a screen performs the physical I/O necessary to acquire the terminal user's response. None of the application program's data value fields are modified by this; a structure (tagged FS_READSCR) is returned indicating the AID character, the number of modified panels, and an array of pointers indicating which panels have been updated. A read panel function must then be called to update the data fields for each individual panel. The read panel function returns a structure (tagged FS_READPAN) with the number of modified fields and an array of field identifiers to indicate which fields were modified. The following is code sequence to read data from the terminal:

```c
struct FS_READSCR fsrcr(), screen;
struct FS_READPAN fsrcr(), panel_struct;
screen = fsrcr();
panel_struct = fsdpvn(panel);  
A SAS/C Full Screen Library "hello, world"
```

What follows is a full-screen "hello, world." For the purposes of clarity, no error return code checking is done. Instead of defining a view, the default view area (screen size equals view area size) is used.
include <stdio.h>
#define FILL-CHAR

void main()

char errmsg[132];
struct FS_TERMATTR *attr_ptr;
char *panel = "panel;"
char str = "hello, world;"
int beg_row, beg_col;
int file_no = 1;
int attr_PROTECTED = BRIGHT;
int color = RED;

fsinit(errmsg);
attr_ptr = fsrttm();
beg_row = attr_ptr->prim_row / 2;
beg_col = attr_ptr->prim_col / 2 - sizeof(str) / 2;
fsdfwp(panel, file_no, beg_row, beg_col,
       strlen(str), str, attr_ptr, color, FILL-CHAR);
fsdfpn(panel, 0, 1, 1, file_no, 1, FS_FORCE);

1. The terminal is put into full-screen mode and the SAS/C 
Full Screen Library is initialized.

2. The return terminal attributes function is called, which 
returns a pointer to a structure containing the terminal 
dimensions.

3. A panel is defined, with the same dimensions as the 
physical screen.

4. The coordinates for centering the text in the middle of 
the display are calculated.

5. A field is defined (to contain the string "hello, world") at 
the coordinates above with the attributes protected, 
bright display, and the color red. A blank fill character is 
used to pad fields.

6. The panel is displayed. The default view area (0) is used, 
and the upper-left corner of the view area is positioned 
to the coordinates(1,1) of the panel.

7. The SAS/C Full Screen Library terminates and returns 
the terminal to the state it was in before full-screen 
mode.

Notes on this program: This is just a simple example but illus­
trates some important points. It runs on any terminal type, regard­
less of the dimensions. If, for instance, the terminal does not have 
color, the request for red display is mapped to something the ter­
minal can handle (high intensity, for example). The program is 
also portable between OS and CMS with no changes.

Another Example

This program demonstrates defining more than one view. Input 
from the user directs subsequent displays. The first view is the 
first line of the terminal and is used to send and receive a panel 
containing two fields: row and column values. The second view 
is the remainder of the display (whatever its size). A huge panel 
is defined to be displayed in this second view. The panel contains 
just one field that spans the entire panel (and is initialized to 
appear as a grid). The row and column values from the first panel
2. The return terminal attributes function is called, which returns a pointer to a structure containing the terminal dimensions.

3. A view area is defined as the first line of the terminal, beginning at row one column one, as wide as the physical terminal width.

4. A panel is defined the same size as the view area.

5. Two input fields (1 & 2) are defined: the row and column of the second panel to be displayed.

6. A second view area that is the lower part of the terminal is now defined.

7. A second panel is defined. Note that its dimensions are much larger than a typical terminal.

8. A field (3) is defined on the second panel. This field is longer than the physical screen is wide. A bit is turned on in the attribute flag (FLDWRAP) to indicate that this field wraps to the next line on the terminal.

9. The first panel is displayed in the first view area.

10. The second panel is displayed in the second view area. The coordinates (input_rows, input_cols) of the panel to be displayed in the upper-left corner of the view area are variable.

11. A read screen function is performed, causing an actual physical I/O.

12. A read panel function updates the first panel's fields. In practice, the attribute flag for the fields or the structure returned by read panel (FS_READPAN) indicates if the fields were actually modified by the terminal user.

13. The updated fields are used to determine the coordinates of the second panel to be displayed at the next display.

SUMMARY

An introduction to data stream programming is given for two reasons: first, to demonstrate how it is accomplished and what is involved and second, to suggest that it is too low-level an interface to be generally useful to most application programs. The SAS/C Full Screen Library is presented as a high-level implementation suitable for applications development, and examples of its use are given.

REFERENCES


IBM Corp. (1986), 3270 Information Display System Data Stream Programmer's Reference, Kingston, NY.

SAS/C is a trademark of SAS Institute Inc., Cary, NC, USA.

IBM is a registered trademark of International Business Machines Corporation, Armonk, NY, USA.

UNIX is a registered trademark of AT&T.