Transport API C Edition 3.4

ANSI LIBRARY REFERENCE MANUAL

Document Version: 3.4
Date of issue: 15 November 2019
Document ID: ETAC340REANS.190

REFINITIV™

Contents

1 Introduction

1.1 Overview

The RDMS ANSI Library (referred to throughout this manual simply as the ANSI Library) provides functions to both interpret and generate ANSI character sequences.

The ANSI Library provides an easy to use programming interface to handle data received from the RDMS ANSI page (referred to simply as ANSI page) sources as well as an interface for constructing page data that can be published on the RDMS.

Pages on RDMS normally have 25 rows of 80 columns of ASCII characters. Each position on a page, in addition to the character displayed, may have the following values associated with it:

- Character-set indicators
- Monochrome attributes (bold, blink, underscore, inverse)
- Color attributes (foreground, background)

Pages must also support cursor-positioning controls so that updates may be applied to a page without retransmitting the entire page. Pages and updates are assembled from characters plus escape sequences and control characters in a format that conforms to the ISO 2022 or ANSI X3.64 standard. This library interprets the ANSI described in the *DEC VT220 Reference Manual* with additions for special character sets and attributes. For further detail on page data formats, refer to the C, “ANSI Page Data Formats.

The **`qa_decode()`**, **`qa_encode()`**, and **`qa_reset()`** functions, along with their associated data structures, represent the heart of the ANSI Library. The **`qa_decode()`** function accepts an ANSI string and applies it to a data structure that contains the page image. The **`qa_encode()`** function accepts a data structure containing a page image along with a list of page regions that have changed and generates an ANSI string containing the appropriate update sequence. The **`qa_reset()`** function resets a page image to its blank screen state.

Four additional functions complete the Library. These functions allow the user to set and retreive the number of rows and the number of columns comprising the user’s page of information. (The Library defaults to a 25 row by 80 column page.) The functions to set the page size attributes are **`qa_set_rows()`** and **`qa_set_columns()`**. One may retrieve the page size using the **`qa_page_rows()`** and **`qa_page_columns()`** functions.

The Library’s C-language source code is supplied along with the Library object. The header file **q_ansi.h** declares the ANSI Library’s functions and defines the data structures used to interfacie with the library.

If color and monochrome character attributes are not required by an application, the ANSI Library may be rebuilt to exclude this information. The manifest definition **ATTRIBUTES** made in the header file **q_ansi.h** controls inclusion of this information. Remove this definition to create a version without color and monochrome attributes.

1.2 ANSI Attributes and Features

1.2.1 About the ANSI Library

The ANSI Library was developed as a value-added set of functions for handling page-formated data on the RDMS.

1.2.2 Characters

This is represented internally as an ASCII value in the range 0x20 to 0x7E, together with a byte representing the graphic set.

1.2.3 Graphic Sets Attribute

The different graphic sets mean that the interpretation of characters is not trivial, except in the case of Sources which use only the US or UK ASCII character sets (Reuter Monitor, Telerate SOP etc.).

```
/*Character set defines*/
#define US_ASCII     'B'
#define UK_ASCII     'A'
#define VT100_GS     '0' /*vt100 graphics */
#define CHAP_SPC     ':' /*chapdelaine ascii */
#define RMJ_SPEC     ';' /*rmj ascii*/
#define GARBAN_S'<' /*garban ascii*/
#define MABON_SP'=' /*mabon ascii*/
#define MOSAIC_G     '>' /*mosaic graphic set */
#define FBI_ASCI   '?' /*fbi special ascii */
#define FBI_SPEC     'f' /* "     "    graphics */
#define TOPIC_CS     't' /*topic character set */
#define GENERL_G     'g' /*general graphics */
#define VIEW_MOS     'v' /*viewdata mosaics */
#define SOP_ASCI     's' /*sop ascii*/
#define SEPR_MOS'w' /*view separated mosaics */
```

1.2.4 Mono/Color Attrbitues

Each cell has “Attributes”, which represent the color, highlighting, blinking, and underline state of the character in the cell.

The prefix **F_** specifies the foreground color; the prefix **B_**, the background color.

```
/* ATTRIBUTE BYTE definitions */
#define _PLAIN     0x00
#define _BLINK     0x01
#define _REVVID       0x02
#define _DIM       0x04
#define _UNDLN        0x08
#define _BRIGHT       0x10

/* definitions for color */
#define F_BLACK      0x00
#define F_RED      0x01
#define F_GREEN      0x02
#define F_YELLOW   0x03
#define F_BLUE     0x04
#define F_MAGENTA0x05
#define F_CYAN       0x06
#define F_WHITE      0x07
#define B_BLACK     0x00
#define B_RED          0x10
#define B_GREEN    0x20
#define B_YELLOW  0x30
#define B_BLUE        0x40
#define B_MAGENTA    0x50
#define B_CYAN      0x60
#define B_WHITE     0x70
```

1.2.5 Fading Mono/Color Attributes

Each cell has “Attributes”, which represent the fading color, highlighting, blinking, and underline state of the character in the cell. The attribute values are the same as the normal attributes. The timing associated with the fading attribute must be done by the application. Fading is explained in the glossary.

1.2.6 Undefined Color Default Value

The default color and the color of erased characters is not defined by the ANSI standard. RDMS applications normally display this as black. It is represented in the data structure by the special value 0x0F. The default monochrome color follows:

```
/*define for no colors set*/
#define MON      0xff /* monochrome */
```

1.2.7 Scrolling

Only vertical scrolling of entire lines (up or down) is supported. There is provision to avoid more than one set of entries in the update list if an update causes multiple scroll operations.

There is no attempt to optimize the handling of multiple scrolling operations.

There is support for the DEC escape sequences to restrict the scrolling region.

1.2.8 Unsupported ANSI Features

Double Height and Double Width character formats are NOT supported.

1.3 Organization of Manual

This manual is organized into the following sections:

CHAPTER	CONTENTS
Chapter 2	Describes ANSI Library data structures.
Chapter 3	Presents function calls with an explanation of correct syntax and return values.
Chapter 4	Provides programming tips and examples of code using ANSI Library functions.
Appendix A	Contains a listing of the **q_ansi.h** header file.
Appendix B	Details on how to install ANSI Library software.
Appendix C	Discusses ANSI Page Data Formats.

Table 1: Manual Overview

1.4 Typographic Conventions

- Directories, filenames, parameters, and list entries appear in bold; e.g.: **q_ansi.h**
- When in text, function names, code snippets, and structures appear in orange, Lucida Console font; e.g.: **`qa_decode()`**
- Document titles appear in italics, e.g.: *Transport API C Edition Developers Guide*
- File listings or code samples appear in a typewriter font; e.g.:

```
typedef struct
{
   unsigned char  flags;
   unsigned char   code;
} SSL_STATUS_CODE;
```

1.5 Glossary

Below is a list of terms and definitions which are used frequently throughout this document.

TERM	DEFINITION
ANSI (American National Standards Institute)	This agency develops standards widely used as guidelines by US firms. Data processing standards developed from ANSI range from the definition of ASCII to the determination of overall datacom system performance.
ANSI Sequence	A string of characters which invokes a mode or status change in the display system. ANSI sequences start with an ESC character (033 octal) and then follow one of several patterns. The receiving device must recognize these patterns and act upon the supported sequences while ignoring unsupported sequences.
Consumer Application	A program, compiled and linked with the RDMS, whose main function is to retrieve market data from the Enterprise Platform.
Fading	A feature that many users and sources require, where an area of updated text must be highlighted for a short time to draw attention to it. After the end of the fade period, the text returns to its original color. Applications implement fading as follows: when a character is changed, it is displayed initially with the “Fading” set of attributes. After a fixed period of time, it is re-displayed with the default attributes.
Image	An image is the complete data representation of an item.
Item	An item is a named data entity. There are two types of items: pages and records. They are identified by the name of the source which supplies them, and the name of the individual item within that source.
Page	A page is a type of item formatted for distribution to display systems. The data includes text and control information in accordance with ANSI X3.64 and X3.41 (plus various extensions and private sequences).
Page-Oriented Source	A source that organizes information in logical pages. Logical pages are of variable dimensions, but typically displayed in 80 columns and 24 or 25 rows.
Record	A record is a type of item presented in a form which is convenient for use by computer applications. The data is presented in the Reuters Marketfeed format.
Source	Any application or server capable of supplying or transmitting information to the RDMS.
Update	An update is a modification to the contents of an item. A source sends updates asynchronously as the contents of the item change.

Table 2: Glossary of Terms

1.6 Related Documents

- *ANSI X3.64 Protocol Specification*, New York, N.Y. 1979
- *DEC VT220 Programers Reference Manual*, Digital Equipment Corporation, 1984
- The Refinitiv Developer Community

2 Data Structures

2.1 Data Structures

The following descriptions summarize the data structures used to interface with the ANSI Library. The structure definitions are taken from the **q_ansi.h** header file.

2.1.1 Character Data Structure

A data structure is used to describe each character on the page. The structure contains the character attributes described in the previous section. The **CHARTYP** data structure follows:

```
typedef struct char_type {
      char ch;                      /* the character*/
      unsigned char gs;             /* graphic set */
      unsigned char attr;           /* mono attrib */
      unsigned char fade_attr;      /* mono fading */
      unsigned char c_attr;         /* color attrib */
      unsigned char c_fade_attr;    /* color fading */
} CHARTYP, *CHARPTR;
```

2.1.2 Page Data Structure

The central data structure **PAGETYP** points to an array of character “cells” corresponding to the matrix of characters comprising the screen. The **PAGETYP** data structure follows:

```
typedef struct page_type {
      CHARTYP *page;          /*sizeof [PAGEROWS * PAGECOLS]*/
      STATUSTYP status;       /*main status*/
      STATUSTYP save;         /* for cursor save/restore */
      short scroll_top;       /* top of scrolling region */
      short scroll_bot;       /* bottom of scrolling reg */
      short last_mod;         /* qa_encode marks last entry
                               in u_list it was able to encode */
} PAGETYP, *PAGEPTR;
```

Status information is kept in the **PAGETYP** structure because the decode function may be called repeatedly for the same page.

The page structure should be initialized with the ANSI *reset* string (ESC c).

The page structure should not be altered between calls to the decode routine.

2.1.3 Update List Data Structure

The **LISTTYP** data structure provides a mechanism to communicate which character cells have changed. It contains an array of **upd_type** structures, each comprising the row, start, and end columns of cells which have been affected by an update. A description of the **upd_type** structure follows:

```
struct upd_type {                    /* structure for incoming updates*/
      short row;                     /* row on which the update appears*/
      short upd_beg;                 /* beginning column of update */
      short upd_end;                 /* ending column of update */
};
```

If more than one row was affected by the message, many of these structures must be provided for each area of change. The **LISTTYP** structure provides a length and pointer to an array of **upd_type** structures. The user must provide the space for the array. The **LISTTYP** structure follows:

```
typedef struct list_type {
      short max_updt;                /* size of upd_list */
      short index;                   /* num of entries in upd_list */
      struct upd_type upd_list[1];   /* this structure is overlaid on the space
                                     the caller provides */
} LISTTYP, *LISTPTR;
```

2.2 Using Data Structures

2.2.1 ANSI Decode

The function **qa_decode()** is used to decode an ANSI string into the **PAGETYP** structure. The function is passed an ANSI update string, plus pointers to a **PAGETYP** structure and **LISTTYP** structure. After the ANSI update string has been processed, the **LISTTYP** structure holds a list of all the fields which have been updated, and the **PAGETYP** structure contains the correct attributes for those fields.

The **qa_decode()** function allows for multiple “passes” over a single string of update information. This allows the Update List to be sized without knowledge of the maximum amount of update information that a source communicates. A call to **qa_decode()** returns the number of characters parsed from the update string. This value is less than the update string length when the Update List size was not sufficient. The application can make subsequent calls to **qa_decode()** to parse the remainder of the update string.

2.2.1.1 Interpreting the Update List

On return from **qa_decode()**, the Update List data structure contains an entry for each area updated. A descriptions of the list follows:

- **max_updt** - number of entries in the list; it is supplied by the user
- **index** - array offset of last entry (starting at 0); it is 1 less than number of entries in list and is set by the library
- **upd_type upd_list** - array of update entries of length [**max_updt - 1**]

The description of each update entry follows:

- **row** - the row (e.g. 1 to 25) in which the data has changed
- **upd_beg** - the column (e.g. 1 - 80) in which the change starts
- **upd_end** - the column (e.g. 1 - 80) where the change ended. This is 1 beyond the last character that has changed so the total number of characters that have changed in a row is (**upd_end - upd_beg**). Zero length updates should be ignored.

An entry is added to the list for the following reasons:

- For each repositioning of the cursor, an entry with zero length is generated for that point. A Carriage Return plus Line Feed is treated as a single sequence.
- Each character written increases the length of the current entry by one. If it wraps around at the end of a line, that is treated as cursor movement, and a new entry added as above.
- Scrolling generates an entry for each row scrolled, and one for the new cursor position following the scroll.

2.2.1.2 Initialization

Only one Update List data structure is required, since this is used only as a means of returning information from **qa_decode()**. One **PAGETYP** data structure is required for each page on display - it contains all of the static information for a page representation. These may be allocated dynamically or created statically.

The size of the **CHARTYP** buffer pointed to by the **PAGETYP** data structure (i.e. the page size that the Triarch ANSI Library function operates on) is communicated to the ANSI Library by the functions **qa_set_rows()** and **qa_set_columns()**. These functions define the user application's page size in rows and columns. The **CHARTYP** buffer referenced by **qa_decode()** (as well as **qa_encode()** and **qa_reset()**) must correspond to this page size.

Page size is maintained statically by the ANSI library. Once the page size has been set, one need only call **qa_set_rows()** and **qa_set_columns()** when the page size changes. The default page size is 25 rows by 80 columns. If a user's page size is always 25 rows by 80 columns, the **qa_set_rows()** and **qa_set_columns()** functions need not be called.

The update list must have the field **max_updt** initialized to the number of **upd_type** elements in the array. Note that this field is not modified by **qa_decode()**.

Each **PAGETYP** structure is best initialized by calling **qa_decode()** passing the ANSI initialize sequence (*ESC c*) which resets the entire data structure without reference to the original contents. In addition, if **qa_encode()** is to be used, the **last_mod** field must be set to the value (-1). Example initialization code follows:

```
void InitializePage( PAGEPTR page, LISTPTR u_list )
{
/* max_updt must be set and the array size > 30 */
      (void)qa_decode( page, "\033c", 2, u_list );
      page->last_mod = -1;
}
```

2.2.2 ANSI Encode

When the content of one or more regions of a page change, the application updates the page image directly, keeping track of the positions changed in the update list. The application then calls the **qa_encode()** function with the **PAGETYP** and **LISTTYP** data structures that reflect the current state of the page and the areas which were updated. **qa_encode()** generates a string containing the ANSI sequences required to communicate the changes to an RDMS consumer application.

The function permits multiple "passes" to manage updates that exceed the size of the update string buffer. It uses a field within the **PAGETYP** structure, **last_mod**, to save the state between passes. This field must be reset to the value **-1** before the function is called with a new update.

The size of the **CHARTYP** buffer pointed to by the **PAGETYP** data structure (i.e. the page size that the ANSI Library function operates on) is communicated to the ANSI Library by the functions **qa_set_rows()** and **qa_set_columns()**. These functions define the user application's page size in rows and columns. The **CHARTYP** buffer referenced by **qa_encode()** (as well as **qa_decode()** and **qa_reset()**) must correspond to this page size.

Page size is maintained statically by the ANSI library. Once the page size has been set, one need only call **qa_set_rows()** and **qa_set_columns()** when the page size changes. The default page size is 25 rows by 80 columns. If a user's page size is always 25 rows by 80 columns, the **qa_set_rows()** and **qa_set_columns()** functions need not be called.

3 Function Descriptions

3.1 Overview

This section contains the description and syntax for the Triarch ANSI LIbrary functions:

- **`qa_decode()`**: Decodes an ANSI character sequence.
- **`qa_encode()`**: Encodes an ANSI character sequence.
- **`qa_reset()`**: Resets a page image.
- **`qa_set_rows()`** and **`qa_set_columns()`**: Informs the Library of the user's page size.
- **`qa_page_rows()`** and **`qa_page_columns()`**: Retrieves the Library's user page size settings.

Data structures used by these functions are described in Chapter 2.

3.2 qa_decode()

3.2.1 Synopsis

```
#include "q_ansi.h"
int qa_decode(
      PAGEPTR page,     /*Pointer to the PAGETYP data structure*/
      char *text,       /*string (need not be null-terminated)containing
                          the ANSI sequence to apply*/
      int length,       /*length of this string (in characters)*/
      LISTPTR u_list)   /*Update list data structure*/
```

3.2.2 Description

Decoding an ANSI string is accomplished using a finite state machine. Each of the ANSI escape sequences is parsed according to the state into which the previous characters have the machine.

As the sequences are recognized, the page image is modified.

3.2.3 Return Value

`Qa_decode()` returns the number of characters in **text** it has successfully parsed. It does not return a length which would cause an escape sequence to be broken. **`Qa_decode()`** may be called as many times as necessary until all the text is parsed.

If **`qa_decode()`** has completed, the **length** which was passed to it is returned.

3.3 qa_encode()

3.3.1 Synopsis

```
#include "q_ansi.h"
int qa_encode(
      PAGEPTR page,                 /*pointer to the stored page image*/
      unsigned char *text,          /*pointer to buffer for ANSI text*/
      long maxstrlen,               /*size of text buffer*/
      long *length,                 /*offset in text after encoding*/
      short fade_enable,            /*flag to enable fading*/
      LISTPTR u_list)               /*update list pointer whose data
                                    structure contains a list of all the
                                    blocks that need to be encoded*/
```

3.3.2 Description

This routine takes a stored page image along with a list of regions that the application has changed and encodes it into an ANSI string containing the appropriate update sequence. This function is useful when creating pages to publish on the RDMS network via the platform when displaying the output of an application on an ANSI terminal.

The **`qa_encode()`** function processes each entry in the update list up to the index. For the row and start column specified in the update list up to (but not including) the end column, the corresponding characters and attributes in the page image are added to **text**.

3.3.3 Return Value

If **`qa_encode()`** is able to encode all of the data regions defined by the update list into **text**, it returns DONE.

If **`qa_encode()`** runs out of room in **text** before the image is completely encoded, it returns **NOT_DONE**, and **last_mod** in the page image is set to the entry in the update list where **`qa_encode()`** will resume encoding the next time it is called. **`qa_encode()`** may be called as many times as necessary to completely encode all the regions defined in the update list.

The memory pointed to by **length** contains the length of text in either case.

3.4 qa_reset()

3.4.1 Synopsis

```
#include "q_ansi.h"
short qa_reset(
     PAGEPTR page,   /* pointer to the page image to be cleared */
     char *not_used,/* char pointer not effected by the library */
     LISTPTR u_list)/* pointer to update list to be reset */
```

3.4.2 Description

This routine takes a pointer to a page image and an update list that the user wishes to be reset.

The **qa_reset()** function resets the page image to the blank screen state (i.e. a page of space characters with **_PLAIN** attributes. **qa_reset()** also resets the update list passed to the function.

3.4.3 Return Value

If **qa_reset()** returns 1.

3.5 qa_set_rows()

3.5.1 Synopsis

```
#include "q_ansi.h"
void qa_set_rows(short rows)/* no. of rows in user's page */
```

3.5.2 Description

This routine takes the number of rows in a user's page. **qa_set_rows()** is used in conjunction with **qa_set_columns()** to tell the library the size of the **CHARTYP** buffer being passed in subsequent **qa_encode()**, **qa_decode()**, and **qa_reset()** function calls.

The Triarch ANSI Library uses the page size information during processing of page images in **qa_encode()**, **qa_decode()**, and **qa_reset()**.

Page size is maintained statically by the Triarch ANSI library. Once the page size has been set, one need only call **qa_set_rows()** and **qa_set_columns()** when the page size changes. The default page size is 25 rows by 80 columns. If a user's page size is always 25 rows by 80 columns, the **qa_set_rows()** and **qa_set_columns()** functions need not be called.

3.5.3 Return Value

This function has no return value.

3.6 qa_set_columns()

3.6.1 Synopsis

```
#include "q_ansi.h"
void qa_set_columns(
     short cols)  /* no. of columns in user's page */
```

3.6.2 Description

This routine takes the number of columns in a user's page. **qa_set_columns()** is used in conjunction with **qa_set_rows()** to tell the library the size of the **CHARTYP** buffer being passed in subsequent **qa_encode()**, **qa_decode()**, and **qa_reset()** function calls.

The Triarch ANSI Library uses the page size information during processing of page images in **qa_encode()**, **qa_decode()**, and **qa_reset()**.

Page size is maintained statically by the Triarch ANSI library. Once the page size has been set, one need only call **qa_set_rows()** and **qa_set_columns()** when the page size changes. The default page size is 25 rows by 80 columns. If a user's page size is always 25 rows by 80 columns, the **qa_set_rows()** and **qa_set_columns()** functions need not be called.

3.6.3 Return Value

This function has no return value.

3.7 qa_page_rows()

3.7.1 Synopsis

```
#include "q_ansi.h"
short qa_page_rows(void)
```

3.7.2 Description

Page size (in rows and columns) is maintained statically by the Triarch ANSI Library. (It can be set using the **qa_set_rows()** and **qa_set_columns()** function calls.) **qa_page_rows()** returns the current number of rows in a page.

The Triarch ANSI Library uses the statically maintained number of rows in a page to process page images during the **qa_encode()**, **qa_decode()**, and **qa_reset()** function calls. The information is used to determine the size of the **CHARTYP** buffer being passed to those functions.

3.7.3 Return Value

This function returns the current Triarch ANSI Library setting for the number of rows in a page.

3.8 qa_page_columns()

3.8.1 Synopsis

```
#include "q_ansi.h"
short qa_page_columns(void)
```

3.8.2 Description

Page size (in rows and columns) is maintained statically by the Triarch ANSI library. (It can be set using the **qa_set_rows()** and **qa_set_columns()** function calls.) **qa_page_columns()** returns the current number of columns in a page.

The Triarch ANSI Library uses the statically maintained number of columns in a page to process page images during the **qa_encode()**, **qa_decode()**, and **qa_reset()** function calls. The information is used to determine the size of the **CHARTYP** buffer being passed to those functions.

3.8.3 Return Value

This function returns the current Triarch ANSI Library setting for the number of columns in a page.

4 Hints, Tips, and Examples

4.1 Programming Hints and Tips

Programmers should be aware of the following:

- The function **qa_decode()** displays an error message when it encounters an invalid sequence. This feature may be excluded by undefining the manifest parameter **QA_DISPLAY_ERR** in **q_ansi.h** and rebuilding the library.
- The **LISTTYP** structure member **index** starts at zero. Thus a value of 4 indicates 5 update entries.
- It is recommended that the update list passed to **qa_decode()** be made large enough to handle likely Triarch message updates (see the next comment).
- The number of update list entries generated by a complex update can be very large, and sufficient space should be allowed to avoid overflow. An update list buffer offering 1000 entries would not be excessive for handling Telerate pages, where the drawing of vertical columns on a row by row basis can produce hundreds of individual updates.

4.2 Examples

The following code gives examples of routines which call the ANSI **qa_decode()** and **qa_encode()** functions.

```
/*****************************************************
* NAME:        main
*
* DESCRIPTION: This example program demonstrates the
*   use of the ANSI Library functions. The program:
*   clear initializes the page and update structures
*   decodes two siple strings and checks to see if an
*   update was generated in a certain range
*   prints out a block of the modified page to stdout.

*
*****************************************************/

#include <stdio.h>
#include "q_ansi.h"


/*define initialization and example strings used in program*/
#define INIT_STRG"\033c\033(B\033[0;47;30m\033[1;1H"
#define CHG_15"\033[1;20Hwrite on line one\
          \033[5;40Hwrite on line five"
#define CHG_48 "\033[4;10Hwrite on line three \
          \033[8;30Hwrite on line eight"
#define poffset(pageptr,row,col) &(pageptr->page [((((row) - 1) * PAGECOLS)+((col) - 1))])
#define MY_ROWS 25
#define MY_COLS 80
main()
{
```

```
   void EncodeAnsi();/*declare lower routine*/
   PAGETYP qa_page; /*page structure*/
   CHARTYP pageImage[MY_ROWS*MY_COLS];
   LISTTYP *qa_ulist;/*pointer to update struc*/
   char qa_list_sp[sizeof(LISTTYP) +
      sizeof(struct upd_type) * 1000];
                  /*space for update*/
   int retv,cnt;
   struct upd_type *l_ptr;

   /* set up the page structure */
   qa_page.page = pageImage;
   qa_set_rows(MY_ROWS);
   qa_set_columns(MY_COLS);

   /* set up update list area*/
   qa_ulist = (LISTTYP *)qa_list_sp;
   qa_ulist->max_updt = 1000;

   /* clear data area */
   qa_decode(&qa_page, INIT_STRG, sizeof(INIT_STRG),qa_ulist);

   /*run decode test with samples*/
   retv = DecodeAnsi(1,80,7,10, CHG_15, qa_ulist, &qa_page);
   printf("decode test 1 returned %d\n",retv);
   retv = DecodeAnsi(1,80,7,10, CHG_48, qa_ulist, &qa_page);
   printf("decode test 2 returned %d\n",retv);

   /*print out columns 15 to 35 for rows 4 to 9
     set up update structure*/
   qa_ulist->index = -1;
   for(cnt=4,l_ptr=qa_ulist->upd_list;cnt <=9;cnt++,l_ptr++){
      l_ptr->row = cnt;
      l_ptr->upd_beg = 15;
      l_ptr->upd_end = 35;
      (qa_ulist->index)++;
   }
   EncodeAnsi(qa_ulist,&qa_page);
   exit(0);
}

/************************************************
* Name: DecodeAnsi
* Description: processes ANSI code and determines if a
* change was made in columns x1 to x2, rows y1 to y2
* it returns TRUE if an update was made in that region
* it also returns the updated PAGETYP structure and the
* last list of updates in the LISTTYP structure
* Parameters: range to check x1, x2, y1, y2
*  pointer to null terminated update string *sText
*  pointer to LISTTYP structure *u_ptr
*  pointer to PAGETYP structure *pg_ptr
************************************************/
DecodeAnsi(x1,x2,y1,y2,sText,u_ptr,pg_ptr)
int x1,x2,y1,y2;
```

```
char *sText;
LISTTYP *u_ptr;
PAGETYP *pg_ptr;
{
   long text_len,ch_p;
   int retv,ucnt;
   struct upd_type *l_ptr;

   retv = 0; ch_p = 1;
   text_len = strlen(sText);
   for(;text_len >0 && ch_p > 0;text_len -= ch_p,sText += ch_p){
      ch_p = qa_decode(pg_ptr, sText, text_len, u_ptr);
      /*check all updates for range*/
      for(ucnt = (u_ptr->index +1),l_ptr =
          u_ptr->upd_list;ucnt;ucnt--,l_ptr++){
         if(!retv && l_ptr->row >= y1 && l_ptr->row <= y2){
            if((l_ptr->upd_beg >= x1 &&
               l_ptr->upd_beg <= x2) ||
              (l_ptr->upd_end >= x1 &&
               l_ptr->upd_end <= x2))
                retv = 1;
         }
      }
   }
   return(retv);
}

/************************************************
* Name: EncodeAnsi
* Description: processes PAGETYP and LISTTYP structures
* and produce an ANSI string that will be written to
* stdout
* Parameters: pointer to LISTTYP structure *u_ptr
*  pointer to PAGETYP structure *pg_ptr
************************************************/
void EncodeAnsi(u_ptr,pg_ptr)
LISTTYP *u_ptr;
PAGETYP *pg_ptr;
{
   unsigned char tmp_buf[1000];
   int retv;
   long len;
   short last_mod;

   last_mod = pg_ptr->last_mod = 0;
   do {
      last_mod = pg_ptr->last_mod;
      retv = qa_encode(pg_ptr,tmp_buf,1000,&len,0,u_ptr);
      tmp_buf[len] = '\0';
   /*while not done and progress is being made*/
```

```
    } while(last_mod != pg_ptr->last_mod && retv != DONE);
}
```

Appendix A ANSI Include File - q_ansi

```
/*--------------------------------------------------------
 *       Copyright (C) 1995, 2019 Refinitiv,          --
 *                                                    --
 *   1111 22nd Street, Oak Brook, IL. 60521           --
 *                                                    --
 *    All rights reserved.                            --
 *     Duplication or distribution prohibited         --
 *--------------------------------------------------------
 */


#ifndef Q_ANSI_H
#define Q_ANSI_H

/* define return values for qa_encode */
#define DONE0
#define NOT_DONE1


/* define ATTRIBUTES when monochrome, color, and fading
   attributes are needed.  Leaving ATTRIBUTES undefined
   saves space.
*/
#define ATTRIBUTES


struct upd_type {/* structure for incoming updates  */

short  row;      /* row on which the update appears */
short  upd_beg;  /* beginning column of update      */
short  upd_end;  /* ending column of update */
};


typedef struct list_type {

short  max_updt;    /* size of upd_list */
short  index;    /* num of upd_list entries */
struct upd_type upd_list[1];  /* this structure is overlaid
             on the space the caller
             provides */
} LISTTYP, *LISTPTR;


typedef struct status {

short  row;
short  col;
unsigned char cur_attr;/* monochrome attribute */
unsigned char c_attr;/* color attribute */
unsigned char c_fade_attr;/* color fading attr */
unsigned char fading;/* mono fading attr */
unsigned char gr_set;/* flag to select G0 or G1 */
unsigned char G0_set;/* G0 graphics set */
unsigned char G1_set;/* G1 graphics set */
short wrap_on;/* flag for line wrap */
```

```
short vem;/* flag for vert editing mode */
short hem;/* flag for horiz edit mode */
} STATUSTYP, *STATUSPTR;


typedef struct char_type {

char ch;
unsigned char gs;/* graphic set */
#ifdef ATTRIBUTES
unsigned char attr;/* mono attrib */
unsigned char fade_attr;/* mono fading */
unsigned char c_attr;/* color attrib */
unsigned char c_fade_attr;/* color fading */
#endif
} CHARTYP, *CHARPTR;


typedef struct page_type {

CHARTYP *page;        /* size of [PAGEROWS * PAGECOLS] */
STATUSTYP status;
STATUSTYP save;/* for cursor save/restore */
short scroll_top;/* top of scrolling region */
short scroll_bot;/* bottom of scrolling reg */
short last_mod;/* qa_encode marks last entry
          in u_list it was able to
          encode */
} PAGETYP, *PAGEPTR;


#define PAGELENGTH sizeof(PAGETYP)
#define CHARLENGTHsizeof(CHARTYP)

/* ATTRIBUTE BYTE definitions */
#define _PLAIN0x00
#define _BLINK0x01
#define _REVVID0x02
#define _DIM0x04
#define _UNDLN0x08
#define _BRIGHT0x10

/* definitions for color */
#define MONO0xff/* monochrome */
#define F_BLACK0x00
#define F_RED0x01
#define F_GREEN0x02
#define F_YELLOW0x03
#define F_BLUE0x04
#define F_MAGENTA0x05
#define F_CYAN0x06
#define F_WHITE0x07
#define B_BLACK0x00
#define B_RED0x10
#define B_GREEN0x20
#define B_YELLOW0x30
#define B_BLUE0x40
#define B_MAGENTA0x50
#define B_CYAN0x60
#define B_WHITE0x70
```

```
/* definitions for character size */
#define SHSW     0x00/* single height single width */
#define DHSW_TOP 0x01/* double height single width top */
#define DHSW_BOT 0x02/* double height single width
          bottom */
#define DHDW_TOP 0x03/* double height double width top
        */
#define DHDW_BOT 0x04/* double height double width
          bottom */
#define SHDW     0x05/* single height double width */


/* Modes for encoding ansi
   UPDATE mode adds fading blink
*/
#define NO_FADE0
#define NORM  1
#define UPDATE  2
#define ISSUPDT  3


/* define character set selection parameters for use with
   encode ansi routine.  These parameters are to be
   used to indicate character set or sets being used to
   create the page image.
*/
#defineUS_ASCII'B'
#defineUK_ASCII'A'
#defineVT100_GS'0'/* vt100 graphics */
#defineCHAP_SPC':'/* chapdelaine
             special ascii */
#defineRMJ_SPEC';'/* rmj "       "   */
#defineGARBAN_S'<'/* garban " "   */
#defineMABON_SP'='/* mabon " "   */
#defineMOSAIC_G'>'/* mosaic graph set*/
#defineFBI_ASCI'?'/* fbi special
             ascii */
#defineFBI_SPEC'f'/*  "     "
              graphics */
#defineTOPIC_CS't'/* topic character
             set */
#defineGENERL_G'g'/* general graphics*/
#defineVIEW_MOS'v'/* viewdata mosaics*/



/* escape sequnce number and out sequnce table offsets */
#define DBH_TP1  /*double high top*/
#define DBH_BT2  /*double high bottom*/
#define DBHW_TP3  /*double high/wide top*/
#define DBHW_BT4  /*double high/wide bottom*/
#define DB_OFF5  /*double off*/
#define DBWD6  /*double wide single high*/
#define DB_MIN1
#define DB_MAX6

/* masks for bits in attribute sequence fddd aaaa
    where f is 1 when double is active
    where d is the double attribute info
```

```
   where a is the normal attribute info
*/
#define DB_BITS0xE0  /*mask out double flag
              bit */
#define DB_SFT_BITS0x07  /*mask out double flag
              bit after shifting
              DB_SHIFT*/
#define DB_SHIFT5     /*number shifted right to
              right justify double
              attribute bits*/
#define DB_ATT_MASK   0x1f    /*normal attribute mask*/

#define BACK_COL_MASK 0xf0
#define FORG_COL_MASK 0x0f


#ifdef __cplusplus
extern "C"
{
#endif

#if defined(__cplusplus) || defined(__STDC__)

extern int qa_decode( PAGEPTR,char*,int,LISTPTR);
extern int qa_encode( PAGEPTR,unsigned char*,long,long*,short,LISTPTR);
extern short qa_page_rows();
extern short qa_page_columns();
extern void qa_set_rows( short );
extern void qa_set_columns( short );
extern short qa_reset(PAGEPTR,char*,LISTPTR);

#else

extern int qa_decode();
extern int qa_encode();
extern short qa_page_rows();
extern short qa_page_columns();
extern void qa_set_rows();
extern void qa_set_columns();
extern short qa_reset();

#endif

#ifdef __cplusplus
}
#endif

#endif
```

Appendix B Installation Instructions

This Appendix provides the information necessary to install and set up the ANSI Library for use in application development. It lists the hardware and software requirements for ANSI Library use and gives instructions for loading theANSI Library software from the release media. Also included are explanations about incorporating the library in application code, and an installation troubleshooting guide.

B.1 Hardware Requirements

A Sun Microsystems Series 4 SPARC processor with a minimum of 8 megabytes of RAM memory is required to use this product. Approximately 70 kilobytes of hard disk space is required to load the ANSI Library software. At run time, the ANSI Library must have additional memory passed to it in the form of the **`PAGETYP`** data structure, the update list, and the ANSI string. That memory must be allocated by the user's application.

B.2 Software Requirements

The ANSI Library running on the SPARC platform requires that the Sun Operating System must be SunOS Version 4.0.3 or higher.

B.3 Installing the Software

The ANSI Library load consists of all the files needed to incorporate the ANSI Library as a set of library functions into client application programs. The Sun SPARCstation load is distributed on a Sun cartridge tape in "tar" format. To load the software, follow the instructions below.

After any step that requires you to type, press ENTER (or RETURN).

▶ To install the ANSI Library:

1. Place the ANSI Library load tape on a SPARCstation streaming tape drive.
2. At the UNIX prompt ($ in this example), change directories to the target directory in which you wish the ANSI Library to reside.
3. Type: **`$ cd /target.dir`**
4. Copy the source files from the tape into your directory.
5. Type: **`$ tar -xvf device_name`**, where ***`device_name`*** is the device name of the streaming tape drive, usually **/dev/rst0**.

   The files listed and described below are supplied on the tape and should now be in your directory.

   - **libansi.a**: This is the library file of ANSI Library functions that must be linked with any application using the ANSI Library.
   - **q_ansip.h**: You must insert this include file at the beginning of any source code using the ANSI Library functions.
   - **README**: This file describes the contents of the tape.
   - **CHECKSUM**: This file can be used to check for file transfer errors in the files that are downloaded from tape. The following command runs the standard checksum utility **sum** on the file named on the command line (shown in the example as ***`filename`***). You can use the checksum utility to check the header files listed in this section and the **libansi.a** file.
6. Load ***CHECKSUM*** from tape
7. At the UNIX prompt (**`$`** in this example), type: **`$ sum filename`**

B.4 Compiling

For application programs using the ANSI Library, you must add a path element to the developer's **.profile** that points to the directory in which you want the ANSI Library installed (***targetdirectory*** in the following example). To the developer's **.profile**, add:

```
PATH=$PATH:/targetdirectory; export PATH
```

For any program that uses the ANSI Library (**ansitest.c** in this example), create an executable file (**ansitest** in this example) using the compiling command. At the UNIX prompt (**$** in the example) enter:

```
$ cc ansitest.c libansi.a -o ansitest
```

Alternatively, if **ansitest.o** is placed in **$OBJS** of a **makefile**, the following **makefile** entries can be used to create an executable called **ansitest**:

```
ansitest:$(OBJS) libansi.a
cc $(OBJS) libansi.a $ (CFLAGS) -o ansitest
```

B.5 Troubleshooting: Copying the Library

If **libansi.a** is copied using the **cp** command before its use, then **ranlib** must be executed. At the UNIX prompt (**$** in this example) enter:

```
$ ranlib -t libansi.a
```

If you do not execute **ranlib**, you will see the following error message:

```
"ld: libppp.a: warning: table of contents for archive is out of date; rerun ranlib (1)."
```

There is no limit to the number of times **ranlib** can be applied to the library.

Appendix C ANSI Page Data Formats

C.1 Overview

Each position on a page, in addition to the character displayed, may have the following values associated with it:

- Character-set indicators
- Attributes (dim, bold, blink, underscore, reverse-video)
- Foreground and background color attributes

A complete page image consists of a full page of data and control codes. Updates can be broadcast as cursor controls followed by data or attributes. Pages must support cursor-positioning controls to make efficient use of screen-display capabilities, such as when updates to specific parts of a page are being transmitted.

The *Reuters Multilingual Text Encoding Standard* describes the encoding of graphical character sets and control functions. It gives standards for selecting character sets and switching between character sets in use. The Reuters supported character sets and control functions are defined in the text encoding standard as well. For example, the Reuters Basic Character Set is essentially ASCII. ASCII is a national variant of the ISO 646 standard for character sets. RDMS also supports the UK national variant of ISO 646 (the main difference being the national version of the 'pound' sign).

RDMS supports a subset of the Reuters standard as defined in this appendix. While an attempt is being made to conform to the international Unicode Standard, there are cases in RDMS where inheritance of private character sets clashes with the international standard for control sequences.

The private escape sequence SFR is used to control fading. Fading is a feature of RDMS display systems which allows the presentation software to enhance the display of recently modified data for a specified period of time. This is most typically used for real-time updates.

A full list of control functions and escape sequences supported by RDMS is given in this appendix. Supported commands consist of ANSI sequences for information exchange, DEC private sequences, and Rich private sequences. The definitions that follow are used in the transmission of information in the **Data** portion of the **SSL_ET_ITEM_IMAGE** and **SSL_ET_ITEM_UPDATE** events. All references to these command sequences refer explicitly to the use of 7-bit character encoding.

C.2 Introducer Sequence

The following control sequence introducer is supported.

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
CSI	ESC [	1B 5B	Control sequence introducer. The optional code sets the hex values for control sequence introducer.

Table 3: Introducer Sequences

C.3 Cursor Control Sequences

The following cursor control sequences are supported.

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
BS	^h	0/8	The cursor is moved one space to the left, but not past the beginning of the line.
HT	^i	0/9	The cursor is moved to the next tab stop (default is set to 8-column tabs) or to the right margin if no more tabs exist on the line.
LF	^j	0/10	The cursor is moved down one position, scrolling if necessary.
CR	^m	0/13	The cursor is moved to the left margin of current line.
DECSC	ESC 7		This command is a DEC private request used to store the current cursor position and attribute for later use.
DECRC	ESC 8		This command is a DEC private request used to restore cursor position and attribute from the values saved by DECSC request.
IND	ESC D		The cursor is moved to the next lower line (without changing column position), scrolling if necessary.
NEL	ESC E		The cursor is moved to the first position of the next lower line, scrolling if necessary.
RI	ESC M		The cursor is moved to the preceding line (without changing column position), scrolling down if necessary.
HVP	CSI Pl ; Pc f		The cursor is moved to the screen position on line Pl and column Pc. Lines and columns are numbered starting at 1. A parameter value of zero or null is interpreted as a value of one. An HVP command with no parameters has the same effect as a home command. Parameter values are tested to restrict the cursor location to the current screen.
CUU	CSI Pn A		The cursor is moved upwards Pn positions, stopping at the top line. The default is set to one position and moves one position if Pn = 0.
CUD	CSI Pn B		The cursor is moved down Pn positions, stopping at the last line. The default is set to one position and also moves one position if Pn = 0.
CUF	CSI Pn C		The cursor is moved forward Pn positions, stopping at right margin. The default is set to one position and also moves one position if Pn = 0.
CUB	CSI Pn D		The cursor is moved backward Pn positions, stopping at left margin. The default is set to one position and also moves one position if Pn = 0.
CUP	CSI Pl ; Pc H		The cursor is moved to position Pl,Pc. This is the same as HVP control sequence.

Table 4: Cursor Control Sequences

C.4 Scrolling Sequences

The following scrolling sequences are supported.

C.4.1 Margins

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
DECSTBM	CSI Pt ; Pb r		This command selects the top and bottom margins defining the scrolling region. Pt is the top line and Pb is the bottom line. Lines are numbered starting at 1. The default state is (1;24).

Table 5: Scrolling Sequences: Margins

C.4.2 Movement

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
SU	CSI Pn S		Scroll up Pn lines.
SD	CSI Pn T		Scroll down Pn lines.

Table 6: Scrolling Sequences: Movement

C.5 Mode Control Sequences

The following control sequences control the type or mode of the ANSI Page Data Format.

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
SM	CSI Ps h		Commands of this type set (enable) modes as requested by the parameter Ps. The ? character after the CSI defines the sequence to be a DEC private sequence
	CSI ? Ps h		
RM	CSI Ps l		Commands of this type reset (disable) modes as requested by the parameter Ps. The ? character after the CSI defines the sequence to be a DEC private sequence.
	CSI ? Ps l		**NOTE:** The last character in this sequence is a lower case L.

Table 7: Mode Control Sequence Types

The following are supported mode control sequences.

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
DECCOLM	CSI ? 3 h		These DEC private sequences control the number of columns displayed on the terminal. When this mode is set, the terminal displays 132 columns. When reset, only 80 columns are displayed.
	CSI ? 3 l		
DECSCLM	CSI ? 4 h		These DEC private sequences control the scrolling mechanism used by the terminal. When this mode is set, smooth scrolling is used. When reset, jump scrolling is used.
	CSI ? 4 l		
DECAWM	CSI ? 7 h		These DEC private sequences control the auto wrap feature. When invoked, auto wrap performs a new line upon an attempt to write past the right margin of the display area. When disabled, an attempt to write past the right margin causes the right most character of the display to be replaced with the current character.
	CSI ? 7 l		
DECTCEM	CSI ? 25 h		These DEC private sequences control the display status of the terminal's cursor. When this mode is set, the cursor is displayed. When reset, the cursor becomes invisible.
	CSI ? 25 l		

Table 8: Mode Control Sequences

C.6 Display Editing Control Sequences

The following sequences are used to alter or edit the visual display.

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
ED	CSI Ps J		This command erases some or all of the image according to value of Ps. • **Ps = 0**, from cursor to end inclusive • **Ps = 1**, from start to cursor inclusive • **Ps = 2**, all of display The value of Ps Defaults to **0**.
EL	CSI Ps K		This command erases some or all of the current line according to value of Ps. • **Ps = 0**, from cursor to end inclusive • **Ps = 1**, from start to cursor inclusive • **Ps = 2**, all of line The value of Ps Defaults to **0**.
IL	CSI Pn L		This command inserts Pn erased lines starting at the active line and working downward. The remainder of the page is moved downward Pn lines, and the last Pn lines are discarded. The default is set to one line. One line is also inserted if Pn = 0.
DL	CSI Pn M		This command deletes Pn lines starting at the active line and working downward. The remainder of the page is shifted to begin at the active line. The default is set to one line. One line is also deleted if Pn = 0.
DCH	CSI Pn P		This command deletes Pn characters starting at the active position and working to the right. The remainder of the line is moved to the left of the active position. The default is set to one character. One character is also deleted if Pn = 0. The deletion of characters is restricted to the active line exclusively.
ICH	CSI Pn @		This command inserts Pn erased positions starting at the cursor and working to the right. The remainder of the line is moved Pn positions to the right and the last Pn positions are discarded. The default is set to one erased position. One erased position is also inserted if Pn = 0. The insertion of the erased characters is limited to the active line.

Table 9: Display Editing Control Sequences

C.7 Graphic Selection Sequences

The following graphic control sequences are supported on RDMS. Please refer to the Reuters Mulitlingual Text Encoding Standard for the Reuters Basic Character Sets 1 and 2.

A character set is mapped to a "working character set" in the range G0, G1, G2, or G3.

<table>
<tr><th>MNEMONIC</th><th>ANSI SEQUENCE</th><th>OPTIONAL CODE(S)</th><th colspan="2">DESCRIPTION</th></tr>
<tr><td>SO</td><td>^n</td><td>0/14</td><td colspan="2">This command shifts out the current working graphics set and 0/14 invokes the G1 graphics set.</td></tr>
<tr><td rowspan="18">SI</td><td>^o</td><td>0/15</td><td colspan="2">This command shifts in the current G0 graphics set as the working graphics set.</td></tr>
<tr><td rowspan="16">ESC (Pf</td><td rowspan="16"></td><td colspan="2">This command designates the G0 graphics set as the primary graphics set. The physical character set associated with the selection character Pf is invoked whenever G0 is invoked.
The supported parameter values of Pf corresponding to the character set indicators are as follows:</td></tr>
<tr><td>Pf Value</td><td>Description of Parameter Value</td></tr>
<tr><td>B</td><td>ASCII (U.S. national variant of ISO 646)</td></tr>
<tr><td>A</td><td>U.K. national variant of ISO 646</td></tr>
<tr><td>0</td><td>VT100 special graphics (e.g., diamond)</td></tr>
<tr><td>:</td><td>Chapdelaine special set (e.g., fractions, superscripts, arrows)</td></tr>
<tr><td>;</td><td>RMJ special set (e.g., checkmark)</td></tr>
<tr><td><</td><td>Garban character set</td></tr>
<tr><td>m</td><td>Mabon special character set</td></tr>
<tr><td>></td><td>Mosaic graphics set (e.g., solid block)</td></tr>
<tr><td>?</td><td>FBI special character set (e.g., down arrow, subscript 6)</td></tr>
<tr><td>f</td><td>FBI special graphics set (e.g., yen, subscript 2, 4 and 8)</td></tr>
<tr><td>g</td><td>General graphics set</td></tr>
<tr><td>s</td><td>SOP Telerate ASCII set</td></tr>
<tr><td>v</td><td>Viewdata mosaic set</td></tr>
<tr><td>ESC) Pf</td><td></td><td colspan="2">This command designates the G1 graphics set as the primary graphics set. The parameter values of Pf are identical to the G0 case.</td></tr>
</table>

Table 10: Graphic Selection Sequences

C.8 Character Attribute Sequences

The following character attribute sequences are supported.

<table>
<tr><th>MNEMONIC</th><th>ANSI SEQUENCE</th><th>OPTIONAL CODE(S)</th><th>DESCRIPTION</th></tr>
<tr><td>SGR</td><td>CSI Ps m</td><td></td><td>Set Graphic Rendition (SGR) sets display attributes. Defaults to Ps = 0.
More than one parameter, separated by semicolons, can appear in this sequence.
Supported parameter values of Ps include:
<table>
<tr><th>Ps Value</th><th>Description of Parameter Value</th></tr>
<tr><td>0</td><td>All attributes off</td></tr>
<tr><td>1</td><td>High-intensity (bold)</td></tr>
<tr><td>2</td><td>Half-intensity (dim)</td></tr>
<tr><td>4</td><td>Underscore on</td></tr>
<tr><td>5</td><td>Bblink on</td></tr>
<tr><td>7</td><td>Reverse video on</td></tr>
<tr><td>8</td><td>Concealed on</td></tr>
<tr><td>22</td><td>Bold off</td></tr>
<tr><td>24</td><td>Underscore off</td></tr>
<tr><td>25</td><td>Blinking off</td></tr>
<tr><td>27</td><td>Reverse video off</td></tr>
<tr><td>30</td><td>Black foreground</td></tr>
<tr><td>31</td><td>Red foreground</td></tr>
<tr><td>32</td><td>Green foreground</td></tr>
<tr><td>33</td><td>Yellow foreground</td></tr>
<tr><td>34</td><td>Blue foreground</td></tr>
<tr><td>35</td><td>Magenta foreground</td></tr>
<tr><td>36</td><td>Cyan foreground</td></tr>
<tr><td>37</td><td>White foreground</td></tr>
<tr><td>40</td><td>Black background</td></tr>
<tr><td>41</td><td>Red background</td></tr>
<tr><td>42</td><td>Green background</td></tr>
<tr><td>43</td><td>Yellow background</td></tr>
<tr><td>44</td><td>Blue background</td></tr>
<tr><td>45</td><td>Magenta background</td></tr>
<tr><td>46</td><td>Cyan background</td></tr>
<tr><td>47</td><td>White background</td></tr>
</table></td></tr>
</table>

Table 11: Character Attribute Sequences

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
SFR	CSI > Ps m		The private sequence Set Fading Rendition (SFR) specifies that a single attribute or several attributes are invoked in addition to the base attributes defined by the standard ANSI video attribute sequence Set Graphic Rendition (SGR). The display of recently updated fields may thus be enhanced in order to bring the user's attention to the modified area of the display. The added attributes are cleared after a short period of time (usually 5 seconds). The length of application of these additional attributes is controlled by the display device. Specifically, fading allows the addition of any combination of the monochrome attributes normally supported by the display system. For color displays, fading also allows the modification of both the foreground and the background color. Attributes are only added during the fade enhanced period. For example, a base attribute of UNDERSCORE may be enhanced by a fading BLINK. The result would be to display a blinking underscored character for the enhanced period and replace it with a steady underscored character when the enhanced period ends. The foreground and the background colors may be individually replaced for the fade period. If a color is not modified by SFR, then the base color or colors is in effect. If either is replaced, even in the absence of a fading monochrome attribute, fading is invoked. Thus, if base colors are set to blue foreground on a red background and a sequence is received to fade a white foreground, the display is white on red during the enhanced period. At the end of the enhanced period, the display returns to blue on red. The supported parameter values of Ps are the same those listed in the preceding table for SGR. The default is **Ps = 0**.
			NOTE: More than one parameter may appear in this sequence, in which case they are separated by semicolons.

Table 11: Character Attribute Sequences(Continued)

C.9 Character and Line Sizing

The following escape sequences to alter character cell dimensions are supported.

C.9.1 DEC Private Sequences

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
DECDHL	ESC # 3		This command changes the current line to double-height/ double-width top half.
DECDHL	ESC # 4		This command changes the current line to double-height/ double-width bottom half.
DECSWL	ESC # 5		This command changes the current line to single-height/singlewidth.
DECDWL	ESC # 6		This command changes the current line to single-height/doublewidth.

Table 12: DEC Private Sequences

C.9.2 Rich Private Sequences

The following sequences are NOT supported by DEC compatible terminals. These sequences have historically been called Rich Private Sequences. While they may still be in use, they are not consistent with the newer Reuters text encoding standards.

SEQUENCE	DESCRIPTION
CSI > 1 Z	This Rich private sequence invokes double-height/single-width top half mode for all succeeding characters until modified by another character size command.
CSI > 2 Z	This Rich private sequence invokes double-height/single-width bottom half mode for all succeeding characters until modified by another character size command.
CSI > 3 Z	This Rich private sequence invokes double-height/double-width top half mode for all succeeding characters until modified by another character size command.
CSI > 4 Z	This Rich private sequence invokes double-height/double-width bottom half mode for all succeeding characters until modified by another character size command.
CSI > 5 Z	This Rich private sequence invokes single-height/single-width mode for all succeeding characters.
CSI > 6 Z	This Rich private sequence invokes single-height/double-width mode for all succeeding characters.

Table 13: Rich Private Sequences

C.10 Miscellaneous Sequences

The following subsections define all other RDMS supported sequences not grouped in the previous sections.

C.10.1 Control Characters

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
BELL	^g	0/7	Generates the bell tone on the terminal keyboard if the bell is enabled.
DEL		7/15	Delete characters are ignored when received.

Table 14: Control Characters

C.10.2 Controls

MNEMONIC	ANSI SEQUENCE	OPTIONAL CODE(S)	DESCRIPTION
RIS	ESC c		This command resets a device to its initial state, i.e., the state it has after it has been switched on. The screen (25 lines) and fading status is cleared, and the control structure is reset to its initial state (e.g., resets attributes, modes, etc). The scrolling region is reset to the default rows (1;24).

Table 15: Controls

Document ID: ETAC340REANS.190
Date of issue: 15 November 2019