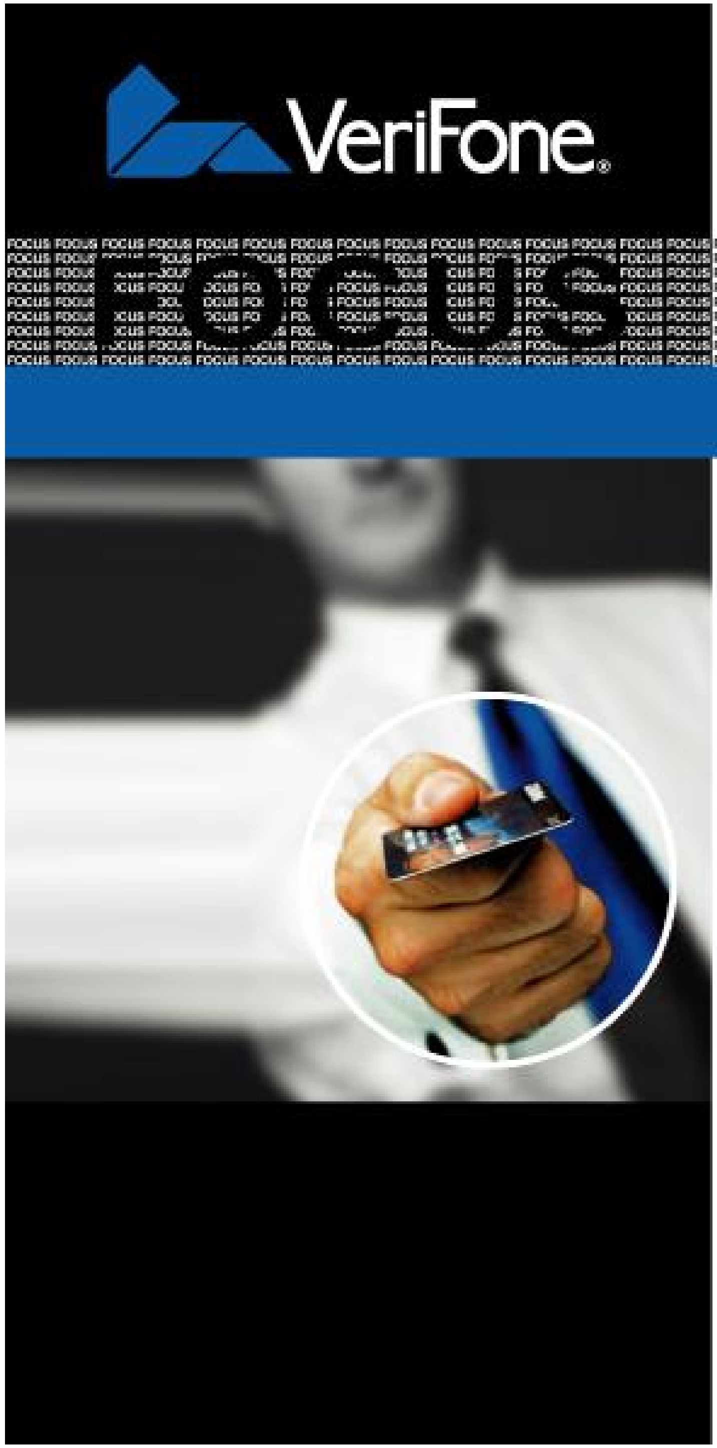

Verix Developer Training

User Interface

The Console

- Controls the display and the keypad
- The console must be opened prior to any console, keypad, or display function call
- `hConsole = open(DEV_CONSOLE, 0);`
  - Clears key buffer
  - Sets font to the default
  - Clears display
  - Turns cursor off
  - Sets contrast to default

Console Ownership

§ Verix mediates sharing of the console among tasks

- A task that successfully opens the console become its owner, preventing other tasks from using it

- The owner task can relinquish the console to allow other tasks to use it

Transfer Console Ownership

- Press the hotkey
- Press the system mode keys ([F2]+[F4]
- Call activate_task() (console owner only)

Basic Console Functions

- `open`
- `close`
- `read`
- `write`

The Display

§ Display size is based on the font

• 6x8 font:	8 lines x 21 characters (Default size for ASCII)
• 8x16 font:	4 lines x 16 characters
• 16x16 font:	4 lines x 8 characters

§ Other fonts may be downloaded as files

§ Graphic images may be displayed by creating custom font files or through direct pixel writes

Display Functions

- `write(CONSOLE, buffer, len)`
- `write_at (buffer, len, x, y)`

- `clrscr(void)`
- `clreol(void)`

- `gotoxy(x, y);`

- `display(buffer)`
- `display_at(column, line, buffer, clr_option);`

Display Functions

- `window(x1, y1, x2, y2)`
- `wherecur(*x, *y)`
- `wherewin(*x1, *y1, *x2, *y2)`
- `wherewincur(*x, *y)`

- `set_cursor(flag)`
- `set_backlight(flag)`

- `SVC_INFO_DISPLAY(buf_6)`

Prompts

- `int prompt(hClock, prompt_str, wait, options);`

  Displays null-terminated string at current line and column position for specified period of time or keypress.

- `int prompt_at(hClock, col, line, prompt_str, wait, opt);`

  Displays null-terminated string at specified line and column for

  specified time or keypress.

Using Fonts

- Font files have a .vft or .fon extension

  Each .VFT font includes 128bytes（fixed）

  – The console driver supports font files up to 65,536 characters.

  – If the font file is 256 characters or less, the console driver can retrieve characters with a one-byte index

  – the specified font file size is greater than 256 characters,the console driver uses two bytes for every character displayed:

Using Fonts

- Font files may be created using the Font Generation Tool
- Font files are located in:
  - <install dir>\font generation tool\samples
- Download font files as a binary image
  - e.g. ddl appl.out –iK2_16x16.vft

- set_font(font_name)
- get_font(*font_name)
- get_font_mode(void)

LAB Exercise

LAB3： Count down application using the display

Write a simple procedure that displays the counting down of a number from a fixed start value to zero.

Keypad

- 12-key Telco-style keypad
  ([0] – [9], [*], [#])
- Four function keys:
  - [ALPHA]
  - [X] Cancel
  - [←] Backspace
  - [↵] Enter
- Eight screen-addressable keys:
  - Programmable function keys **[PF1] – [PF4]**
  - ATM-Style function keys **[F1] – [F4]**

Key Beeps

- By default, key presses are accompanied with a normal beep (key beep).
- To disable key beeps:
  - *`disable_key_beeps()`*
- To enable key beeps:
  - *`enable_key_beeps()`*
  - *`key_beeps(int flag)`*
    - **Flag = 1: beeps enabled**
    - **Flag = 0: beeps disabled**

Data Read Functions

There are two functions to read value of key entry

- read() //non-blocked

-get_char() //blocked

Key Value												
Key Value	**F1**	**F2**	**F3**	**F4**	**X1**	**X2**	**X3**	**X4**	**X**	**<**	**Alpha**	**Enter**
read() HEX	0xfa	0xfb	0xfc	0xfd	0xe1	0xe2	0xe3	0xe4	0x9b	0x88	0x8f	0x8d
get_char() HEX	0x01	0x02	0x03	0x04	0x61	0x62	0x63	0x64	0x1b	0x08	0x0f	0x0d
get_char() ASCII					**a**	**b**	**c**	**d**				**\r**
KEY	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**0**	*****	**#**
read() HEX	0xb1	0xb2	0xb3	0xb4	0xb5	0xb6	0xb7	0xb8	0xb9	0xb0	0xaa	0xa3
get_char() HEX	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x30	0x2a	0x23
get_char() ASCII	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**0**	*****	**#**

Data Entry Functions

- `int SVC_KEY_NUM(dest_buff, max, frac, punctuate);`

  Gets formatted decimal number from keypad. (Counted String)

- `int SVC_KEY_TXT(dest_buff, type, max, min,keymap, keymap_len);`

  Gets formatted input from keypad. (Counted String)

Data Entry Functions

§ `int getkbd_entry(hClock, msg, buffer, wait, type, keymap, keymap_len, va_list);`

Provides display and keypad entry functions.
Returns a Null-Terminated String

```
char szKeyMap [MAX_ALPNUM_KEYS][CHAR_PER_KEY] =
   {"0- +%", "1QZ.\\", "2ABC&", "3DEF!", "4GHI@", "5JKL/",
   "6MNO_",  "7PRS^", "8TUV{", "9WXY}", "*,'\":", "#=;$?"};
```

Data Entry Functions

- `int keyin_amt_range(dest_buff, amt_fmt, max, min, frac);`

  Uses SVC_KEY_NUM() to accept amount entry within a maximum and minimum amount range.

- `int keyin_mapped (key_map);`

  Limits the keys that can be entered. Useful for Y/N responses. key_map is created by ORing the values of all allowed keys as defined in aclconio.h

Keypad Functions

- `int kbd_pending_count(void);`

  Returns the number of keystrokes available for reading. Max 20, Min 0.

- `int KBHIT(void);`

  Determine the number of unprocessed keys in the buffer. Uses kbd_pending_count()

- `int get_char(void)`

  Wait for a key to be pressed.

Keypad Functions

- `int SVC_INFO_KBD (char *buf);`

  Fills the buffer with a 1-byte keyboard type

- `int kbd_pending_test(int t);`

- `int act_kbd_pending_test(int t);`

  Checks if the target character is present in the keyboard buffer.

The Beeper

- Generates audible tones
- Two types of sounds are defined

• Normal Tone	1200 Hz for 50 msec
• Error Tone	889 Hz for 100 msec

- By default, a key press will generate a normal tone
  - Key press tones can be disabled and enabled
- The use of normal and error tones are used within the application appropriately

The Beeper

`sound(note, millisecond)` API has been introduced

- 96 distinctive tones can be played
- No need to settle with error_tone and normal_tone

The Beeper

- The beeper is a shared device and is implicitly opened.
- The beeper may be explicitly opened in order to prevent other applications from using it. Likewise, if the beeper is explicitly opened it must be explicitly closed.

```
Open         hBeeper = open(DEV_BEEPER, 0);

Close        status = close(hBeeper);
```

- To immediately squelch the beeper use:

```
             beeper_off()
```

Emitting Tones

```
normal_tone()

error_tone()

beep(type)

NORM_BEEP()

ERR_BEEP()
```

LAB Exercise

LAB4： Beeper and Keypad

Obtain the start count from the user via the key pad. Make use of normal and error beeper tones to indicate progress and completion of the countdown.

The Message Engine

- Designed to retrieve prompts and messages from a message file
- All static prompts and messages:
  - Are removed from the application
  - Stored in separate “message files”

The Message Engine

§ Advantages of using message files:

- Code size is reduced because data is stored to a file in the file system
- Data is not compiled as part of the application
- Messages/prompts are easily maintained (grouped in to a file)
- If a data file is modified, the file can be updated and downloaded rather than the entire application
- Makes it easy to switch languages for display and printing

Using The Message Engine

- Create an ASCII message file that contains the prompts/messages
- Run TXOFILE.EXE on the message file to generate files useable by the terminal
- Download the message file created by txofile to the terminal
- Use the APIs to access the messages

Message File Format

- The message file is an ASCII file that contains one #define directive for each message.
- Each line in the message file should conform to the following format:

```
    #define [name] [value]  /* Message number for
                               "<message>" */
```

- When TXOFILE process the input file, the 'value' becomes the record number and the 'quoted string' within the comments becomes the message text
- For example:

```
#define STARTVAL 1/*"Enter Starting Value"*/
#define BLAST       2 /*"BLAST OFF"*/
#define COUNT       3 /*"COUNTDOWN" */
```

Converting the Message File

- TXOFILE syntax for converting a message file:
  - `txofile <input file name> <output file name>`
- For example, say your message file is called "mymsgs.txt":
  - `txofile mymsgs.txt mymsgs.dat`
- `#include mymsgs.txt` in your application
- Download `mymsgs.dat` to the terminal with your application

API Functional Reference

- To use the Message Engine
  - `#include message.h`
- To use your message file, include the message file
  - `#include <mymsgs.txt>`
- Within your application you must first select the message file you want to use
  - Only one message file can be selected at any one time
  - You may have as many message files as you wish (memory-dependent)
    - `msg_select_file(filename.dat)`

API Functional Reference

- After the message file is selected, use the following functions to retrieve messages:
  - `msg_get(msg_num, buffer);`
    - This function will retrieve the message from the selected message file and store it into the application buffer.
  - `msg_display_at(col, line, msg_num, buffer, clr_option)`
    - This function is similar to the display_at() function but includes retrieving the message from the selected message file

String Handling - Types of Strings

Null Terminated String

A character string ending with NULL

H	O	W	D	Y	\0

- Counted String

  A string whose first byte is a count of all string elements, including the count

6	H	O	W	D	Y

String Conversion

- Counted string to Null-Terminated string
  - `int SVC_CS2AZ( zs_buff, cs_buff)`
  - `char *ctons1(buffer)`

- Null-Terminated string to Counted string
  - `int SVC_AZ2CS(cs_buffer, zs_buff)`
  - `char *ntocs(dest_buff, sourc_buff)`

String Utilities

- Integer (or Long) to Null-Terminated string
  - `void int2str(dest_buff, value)`
  - `void long2str(dest_buff, long_val)`
  - `char ltoa(long_val, buff, radix)`
  - `char ultoa(us_long, buff, radix)`
- Null-Terminated string to Integer (or Long)
  - `int str2int(buffer)`
  - `int strn2int(buffer)`
  - `int chars2int(buffer, buff_len)`
  - `long str2long(buffer)`

String Utilities

- Purge all characters not digits in a string
  - `int str2digit(buffer)`
- Remove all occurrences of a character from a string
  - `int purge_char(buffer, rem_char)`
- Delete a single character from a string
  - `int delete_char(buffer, char_pos)`
- Convert case of string
  - `void strnlwr( dest, source, size)`
  - `void strnupr( dest, source, size)`

String Utilities

- Counted string to Integer
  - `unsigned int SVC_2INT(source)`
- Integer to Counted String
  - `void SVC_INT2 (unsigned int value, buffer)`
- Convert ASCII hex to binary byte value
  - `BYTE atox (BYTE char)`
- Convert Hex to ASCII and ASCII to Hex
  - `void SVC_HEX_2_DSP(hex_buff, buff, len)`
  - `void SVC_DSP_2_HEX(buff, hex_buff, len)`

String Utilities

- Append a Character to a String
  - `int append_char(buffer, char)`
- Insert Char into a String
  - `int insert_char(buffer, position, char)`
- Insert a Decimal Point in an Amount String
  - `int insert_decimal(buffer)`
- Format a String as a Dollar Amount
  - `void f_dollar(buff, precision, dol_flag, dol_format)`

String Utilities

- Pad a Null-Terminated String
  - `int pad(dest_buf, src_buf, pad_char, pad_size, align)`
- Compress (Decompress) ASCII Data
  - `int SVC_PACK4(dest, src, size)`

    Compress
  - `int SVC_UNPK4(dest, src, size)`

    Decompress
- Compare String to a Control String
  - `char sgetf(src, control, args )`
- Format a String
  - `char sputf(store_buff, next_cs, args)`

String Utilities

- Concatenate Multiple Strings to a Buffer
  - `int mult_strcat(buff, va_arg_list)`
- Copy a Field from a Source buffer to a Dest Buff
  - `int fieldcnt(srs, start, count, dest)`
- Copy a Data String from a Source buffer to a Dest buffer
  - `int fieldfix(src, start, size, dest)`
  - `int fieldray(src, start, stop, dest)`
- Copy variable data field from Source to Dest buffer
  - `int fieldvar(source, field_num, delimiter, dest)`

LAB Exercise

LAB5： Using the Message Engine

Appreciate the usage of the message engine. Put all literal strings in an external text file, convert it into a message file and use it in application execution.

.