Verix Application Development
Lab Exercises

The lab exercises in this training program are designed to compliment the concepts that are discussed in the class. Completion of each exercise will give the participant, an insight into the implementation details.

This document provides the implementation hints for each exercise. These are hints for the experienced C Programmer.

There will be thirteen exercises

1 New Application using batch files

The hello world application built using batch files.

2 New Application using the packaged wizard and VC++

Creation of a new application, using the Verix wizard. Build the application and execute on the terminal.

3 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.

4 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.

5 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.

6 Application Idle Engine

Make use of the idle engine framework to implement simple menus and obtain user entered values.

7 Magnetic Card Reader

Make use of the magnetic stripe reader device to obtain information about the user from the user's magnetic stripe card.

8 Real time Clock and Printer

Read the clock to determine the current date and use it in transaction processing. Print the transaction data in a raw form.

9 Formatter

Beautify or format the transaction data to be printed using the report formatter. Make use of the Formcvt utilty to generate a binary formatter template that will be used runtime by the application while printing.

10 Communication

Format data and communicate with a simulated host. Print the received response in an appropriate field..

11 Running applications in the VMAC Environment

Downloading and executing six sample applications along with VMAC components and ACT shared library.

12 Debugging applications in VMAC using the SDS debugger and LogSysutility

Setup the environment for debugging using the SDS SingleStep debugger. Insert LOGSYS macros in application code and receive them via a terminal emulator running on the PC connected to the terminal. Enable / Disable logging by changing the config.sys parameters.

13 Sending / Receiving custom events between applications

Modify two applications from within the set of downloaded applications such that one of them requests the other for a service to be performed via exchange of events.

Lab 1: New Application using batch files

Goal

To create a basic application

Objectives

At the end of this session the participant will be able to

- Appreciate the steps involved in building an application
- Download and run an application on the terminal

Implementation Hints

1	Write a C program to display "HELLO WORLD".
2	Modify KC.bat, if required, and Compile the source using kc.bat
3	Modify Kl.bat, if required, and Link the object files using kl.bat
4	Sign the executable **.out** file using the VeriShield File Signing utility to get a signature file with extension **.out.p7s**
5	Download the application (**.out**) and its signature (**.out.p7s**)to the terminal using .ddl.exe

Verification

Observe the application running on the terminal with screen display as exepcted.

Lab 2 : New Application using the packaged wizard and VC++

Goal

To use the ACT 2000 Application Wizard to create a basic application

Objectives

At the end of this session the participant will be able to

- Build an application the terminal.
- Download and run an application on the terminal

Implementation Hints

1	Copy the files **Verixwizard.awx, verixwizard.cnt,wzcleanup.exe** from c:\verixaps\tools to the Microsoft Visual Studio template directory

The destination directory would be \progra~1\devstu~1\shared~1\template (assuming that the Microsoft Developer Studio is installed in the directory \progra~1\devstu~1). Please verify the installation in your system to find out the exact location.

2	Open Microsoft VC++ and click on **File | New**. Choose the **Projects** tab. Click on **VerixAppWizard**. Type in the project name <application name> and the location.

The <application name> should not be greater than 6 characters. The location path should conform to **DOS** standards.

3	Click **OK**. In the dialog that shows up, Select the framework as SDK and click on **Next.**
4	In the Next dialog select **Single App** and click on **Finish**.
5	Click **OK** on the dialog that shows up.
6	Click on **Tools | Options**. Choose the **directories** tab. Select Include files for the Show directories for box. Add the following entries (if absent) • \< ACT2000>\include • \<VFSDK>\include (VFSDK is the complete location path for the Verix SDK)

To build the application for the terminal:

1	Click on **Build | Rebuild All**
2	Ensure that the terminal is connected and is in a state to receive downloads(through system mode)

3	Execute the batch file **d.bat** available in the **output\download** directory in your application location..
4	On the completion of the download the application executes on the terminal.

On execution a sample screen is displayed

Note: Your project workspace will be <application name>sds.dsw.

Close the workspace in VC++, back up your current project folder as folder lab2 and continue working on the original folder.

Verification

Show the application running on the terminal.

Lab 3 : Count down application using the display

Goal

To access the display, display strings and re-display strings

Objectives

At the end of this session the participant will be able to

- Use the ACT 2000 API to display prompts runtime.
- Appreciate the usage of **.smk** file in the build process

Implementation Hints

1	In your .smk file make the following changes a. Add path entries for **ACTIncludes** and **ACTlibraries** b. Add **-IACTIncludes** in the include line c. Add **$(ACTLibraries)\act2000.a** and **$(ACT2000)\sdsfiles\libc.a** to the Libs section
2	In a down counting loop, Display "**countdown xx**", (where xx is the count, from 10 to 1) in one second intervals. Use SVC_WAIT for time delays in milliseconds
3	When the loop is exhausted, display "**blast off**", wait three seconds, and exit.
4	Click on **Build | Rebuild All**
5	Ensure that the terminal is connected and is ready to receive downloads(use system mode)
6	Execute the batch file **d.bat** available in the **output\download** directory in your application location..

Expected Output

```
COUNTDOWN 10  (wait 1 second)
COUNTDOWN 9   (wait 1 second)
      |
      |
COUNTDOWN 1   (wait 1 second)
BLAST OFF     (wait 3 seconds)
```

Verification

Show the application running on the terminal.

Lab 4: Beeper and Keypad

Goal

To use the Beeper to intimate the user about application's state audibly and let the user enter information via the key board.

Objectives

At the end of this session the participant will be able to

- Make use of the key pad and beeper API

Implementation Hints

1	Modify the previous exercise program to accept the starting count as keypad input.
2	Display "**countdown xx**" (where xx is the beginning count from step 1 above), wait for 1 second as it loops.
3	When the loop is exhausted (count == 0), display "BLAST OFF", generate an error beep, then restart the loop.
4	Build and download the program

Verification

Show the application running on the terminal.

Lab 5:Using the Message Engine

Goal

To alter the displayed prompts without altering the code.

Objectives

At the end of this session the participant will be able to

- make use of the message engine

Implementation Hints

1	Create a message text file (**count.txt**) that contains the static text from the program.
2	Use the **txofile.exe** utility to convert the text file in to a binary file (**count.msg**).
3	Use the message engine display functions to display the static text within your program.
4	Build the application
5	Update lab.dld in **.<application>\output\download** so that **count.msg** is downloaded with the application and signature files.

Verification

Show the application running on the terminal.

Lab 6: Application Idle Engine

Goal

To implement a simple user interface which comprise of a main Idle menu and child menus.

Objectives

At the end of this session the participant will be able to

- Appreciate the usage of the Idle table and Function tables
- Appreciate the implementation of handler functions.

Implementation Hints

1 Add global variables for transaction data. You will uses these definitions appropriately in the subsequent labs. You may also have them in an include file named **gvars.h.** This is useful in the later labs.

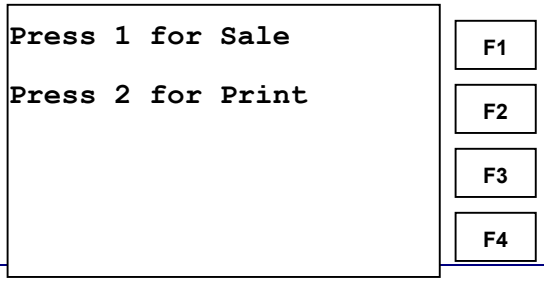
```
char r_acct_num[23];        /*  Account Number */
char r_exp_date[5];         /*  Expiration Date */
char r_amount[10];          /*  Sale Amount */
char r_merchant[20];        /*  Merchant ID */
char r_date[10];            /*  MM/DD/YY */
char r_response[22];        /*  Host Response */
```

2 Create the following tables

- for the idle table BRANCH_TBL_ENTRY idle_table
- for the manager table BRANCH_TBL_ENTRY Manager_tbl1
- for the function table **`PF_TABLE appl_table`**

3 The idle table will support the main menu: **Key 1** for **Sale**, **Key 2** for **print**, **Enter key** for **manager menu.** The manager menu option will not be shown on the screen.

- The Main Menu
-
-
-

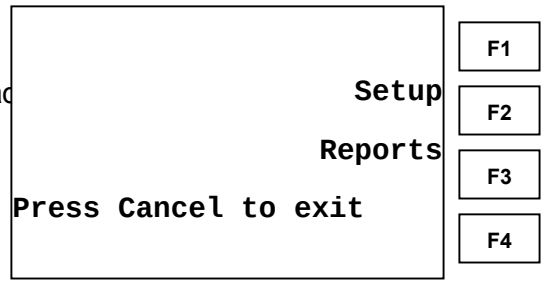


4 Implement main menu functionality

If 1 is pressed display "In Sale" Delay for 3 seconds and return to idle. (implement in the handler for key 1)

If 2 is pressed display "In Print", Delay for 3000 milliseconds and return to idle(implement in the handler for key 2)

- The manager menu on pressing enter key.
- Pressing cancel key should get the main menu bac
-
-
-



5 Implement Manager Menu functionality.

If [F2] is pressed display "In setup" If [X] is pressed return to idle.

If [F3] is pressed display "In Reports", If [X] is pressed return to idle..

5 In the Function table ensure that the numbering starts from 0 and is in consecutive contiguous order. Implement the handler functions.

Verification

Show the application running on the terminal.

Lab 7: Magnetic Card Reader

Goal

To obtain customer information from the customer's magnetic stripe card.

Objectives

At the end of this session the participant will be able to

- Open and read from the card reader
- Parse the track buffer contents into useful fields.

Implementation Hints

1	Modify the previous application to handle a card swipe at two places. In the main menu and after the user presses the 1 key.

In our application we will allow the user to swipe a card or enter the card / account information. Maintain a global flag that indicates whether the data was entered or swiped. This will be useful in the subsequent labs.

2	Open the magnetic card reader
3	Add an entry **APPL_CARD** in the idle table to handle card swipes at the Main menu. The associated card handler function is automatically called.

In the card handler function, read the card reader device and make use of the **card_parse** API to parse the raw track information into easily manageable fields of the TRACK structure.

Display the account number for 3 seconds.

4	When Key 1 is pressed make use of the API **key_card_entry** to obtain data from a swipe or key presses. Based on the API return value you can decide if you want to prompt the user to enter expiry date. Display the account number for 3 seconds.
5	Prompt the user to enter the amount and obtain the same.
6	• Copy all the data obtained to the appropriate global fields.

Verification

Show the application running on the terminal.

Lab 8: Real time Clock and Printer

Goal

To print the raw transaction data along with the current date.

Objectives

At the end of this session the participant will be able to

- Open, initialize and write to the printer port.
- Read the clock to determine the transaction Date.

Implementation Hints

1	Continuing with the previous exercise read the current date from the real-time clock Format the date in the form MM/DD/YY and save it into the global string, **r_date**

You can make use of the API **clock2int.**

2	Hard code the approval message to be **approved** and save it to the global variable **r_response.** This will be replaced by a real host response during the modem lab
3	Implement a function that will print the transaction information as available.

The printer is attached to **comport 4** So you will have to open and initialize it. Use the **set_opn_blk** API for this purpose.

```
parm.rate      = Rt_19200

parm.format    = Fmt_A8N1 | Fmt_auto

parm.protocol  = P_char_mode

parm.parameter = 0
```

After opening the port use the API **p3300_init** to initialize the printer.

Use the API **p3300_print** to print data.

Note: Lines of data need to be terminated with the newline character **'\n'**.

Verification

Show the application running on the terminal.

Lab 9: Formatter

Goal

To format the data that is to be printed

Objectives

At the end of this session the participant will be able to

- Use the report formatter utility formcvt.exe.
- Use the formatter API to print formatted data.

Implementation Hints

1	Rewrite your print routine to print formatted data. The formatter is used for this purpose
2	Create an ASCII text file that serves as the template for the formatter, say receipt.txt

You may use the example in the presentation.

3	From the DOS command line execute Formcvt.exe

e.g. >**formcvt receipt.** This requires all the global variables used in printing included in gvars.h and it will create a downloadable binary file receipt.frm and a header file applic.h. Include applic.h in your application before building it. Remember to download receipt.frm along with the applciation

4	In the print procedure do the following modifications

After the initialization of the printer invoke **formater_main(gvardata)** followed by **formater_open**

Set the conditional flag to reflect the card was swiped or entered. If swiped we want to print the card holder name else print "**Signature**".

Call **formater_line** to print the lines specified in the template.

formater_open takes a function pointer parameter (parameter 4) which is the output initializer for the formatter. In here you provide the device specific API calls which the formatter will call. You can copy this from the solution CD for purpose of this class.

Verification

Show the application running on the terminal.

This page is left blank intentionally

Lab 10a: Communication

Goal

To communicate with a simulated host to obtain an authorization.

Objectives

At the end of this session the participant will be able to

- Perform simple communication with an external host using the Modem engine API.

Implementation Hints

1	Modify the previous exercise to incorporate communication with a simulated host. Remove the hard coded response.

The communication is to be done after amount entry prior to printing.

The host is another terminal connected to a telephone exchange simulator at phone number 101. So the *ZP in your config.sys file will be T101.

2	The communication function will implement the following

Open the modem device for communicating at 2400 baud using the API **xmdm_open.** It is a good idea to flush the buffers after opening using the API **xmdm_flush**.

Use the API **xmdm_get_line_dial** to dial out. The modem initialization string is typically made available in a config.sys parameter. The string that will be used for this lab purpose is **E0V0Q0S0=1**

3	The transaction data to be sent to the host is a twenty byte long character string that is a concatenation of the account number (size should be 16, if less than 16 stuff at the end with the '@' character). The expiry date is in the last four characters in the format **YYMM**.
4	Send the Data using **xmdm_send_data**
5	Obtain the response using **xmdm_receive_data.** Copy the response received into r_response for printing purposes.
6	Wait for 400 milliseconds and hangup using **xmdm_hangup** and wait for 300 milliseconds and close using **xmdm_close**

To build the application for the terminal:

Verification

Show the application running on the terminal.

Lab 10b: Communication - TCP- IP

Goal

To communicate with a simulated host on a TCP/IP network and obtain an authorization.

Objectives

At the end of this session the participant will be able to

- Use the **UCL** and **TCP/IP** libraries to communicate with an external host

Implementation Hints

1 Modify the communication portion of your application alone.

The host is a program that will be running on the PC and listening on a specific port (5001).

2 The communication function will implement the following:
(Tip: Implement these in separate .c files)

a. Methods (functions) for getting and setting the parameters. Special method for the creation of the Application object.

b. Create the object UCLFactory

c. Create a Verix COM timer

d. Using the UCL factory object create a UCL object for communication over the ethernet media. **UclFactory->Create**

e. Pass these timer and UCL communication object as parameters to the API **AssignUclToTCP()**

f. Call **netConnect()** to establish the PPP session.

g. Call **socket()** to create a socket.

h. Call **connect()** to connect to a socket.

i. Call **send()/recv()** to establish communication.

j. Call **closesocket()** to close the session.

k. Call **netdisconnect()** to disconnect from the host.

Verification

Show the application running on the terminal. Observe that you get the same result as in the previous exercise 10a.

Lab 11: Running applications in the VMAC Environment

Goal

To download and execute all the sample applications that are packaged with VMAC 1.2.

Objectives

After completing this exercise, the participant should be able to

- Download VMAC components to a clean terminal.
- Add and remove applications to / from the VMAC environment

Implementation Hints

Note: These implementation hints provide key points for the experienced Verix developer, it does not cover the complete list of actions. E.g. we do not say how to get to the terminal system mode.

1	Clear the RAM and Flash in your terminal so that we start the labs with a clean terminal.
2	**`<vmac install directory>\Output\Core\Download> dlvmac d f`** wait for **`Downloading VMAC Complete`**. The terminal will reboot in this process.
3	**Observation:** The terminal should display the start screen which states - **`starting VMAC 1.3.1 please wait`** and then it displays **`NO APPLICATIONS PRESENT`** ! ! and then you should see the screen saver, **`directed by Veirx Multi-app conductor Press Any Key ...`** On pressing any key it displays **`NO APPLICATIONS PRESENT !!`**
4	Go to system mode in terminal and always choose partial download to group 1 to download applications. The .dld files have a set group directive e.g. setgroup.2 for group 2 to download the application set to the specified group **`<vmac install directory>\Samples\ DmLab1\Output\Download> dldmlab1 d r 2`** Here we are downloading (dldmlab1) the debug version (d) of app to the RAM (r) of group 2 (2). Wait for **`Downloading Sample Complete`** on the PC screen
5	**Observation:** Because there is just one application VMAC automatically tries to start this application and you see a screen that says - **`Application DMLab1 is not running. Make sure INI entry is correct and the shared libraries required by this app are loaded`**
6	**`<vmac install directory>\Samples\ DmLab2\Output\Download> dldmlab2 d r 3`** Wait **for** **`Downloading Sample Complete`** on the PC screen

7	**Observation:** Now the VMAC front end shows both the applications for starting, on choosing any one of them, for example **Application 2** you will see **`Application APP2 is not running. Make sure INI entry is correct and the shared libraries required by this app are loaded`**
8	Similarly partially download all the other applications **<*vmac install directory*>\Samples\sctest\Output\Downoad>dlsctest d r 4** **<*vmac install directory*>\Samples\Vmacex1\Output\Download>dlvmacex1 d r 5** **<*vmac install directory*>\Samples\Vmacex2\Output\Download>dlvmacex2 d r 6** **<*vmac install directory*>\Samples\Vmacsdk\Output\Download>dlvmacsdk d r 7**
9	Download the shared library for ACT **<*vmac install directory*>\Samples\Download>dlactlibf r**
10	**Observation:** On a final restart you will see that all the applications are listed on the FrontEnd excepting VMACex2 which is a background application. You will also observe that you will not be able to print till you choose and exit APP2. You will observe that VMACEx2 start printing form the background as soon as the printer is available. You can stop VMACEx2 printing by choosing VMACEx1 and choosing the option Acquire COM4. Again on exiting VMACEx1 or releasing COM4, VMACEx2 starts printing.

You can remove applications by individually clearing the contents of the application's group.

Lab 12a: Debugging an Application using the SDS Debugger

Goal

To set up the environment to debug a single application under the VMAC environment

Objectives

After completing this exercise, the participant should be able to

- Setup the environment to Debug one application in a set of VMAC compliant applications

Implementation Hints

Note: These implementation hints provide key points for the experienced Verix developer who has already made use of the SDS debugger. It does not cover the art / science of debugging, it just covers the environment setup that needs to be done.

1	All VMAC components should be downloaded to RAM.
2	The Application to be debugged **should always be downloaded to Group 1**. This is independent of its final destination group in normal execution. That group should be cleared. E.g. If the application to be debugged normally executes in group 6. Group 6 Should be cleared and the application should be downloaded to group 1. **NOTE: All downloads following VMAC components SHOULD be downloaded using partial download**
3	Download DBMON.out to Group 1 and DBMON.OUT.P7s to Group 1. In Group 1 set *GO=DBMON.out and *DBMON=pr where p is the port and r is the baud rate. Therefore *DBMON=19 means we are using comport 1 at a baud rate 115200. This is similar to what is done with single applications without VMAC.
4	In Group 15 set the following parameters #DEBUGPORT=COM1 (or COM2 appropriately) MASTER=<logical name of the application to be debugged> The logical name is the name that was specified at the time of registration with VMAC using the API EESL_Initialise or EESL_InitialiseEx
5	Ensure that you start debugging after the execution of the API EESL_Initialise or EESL_InitialiseEx. Afterwards perform debugging as you normally do. Be aware that SDS debugging has limitations in terms of synchronizing with other applications.

Lab 12b Debugging using LOGSYS utility

Goal

To set up an environment where in messages emitted from a piece of application code using the LOGSYS macros are received in a terminal emulator.

Objectives

After completing this exercise, the participant should be able to

- Capture the messages emitted from the application or library in debug mode, using a terminal emulator on a PC. e.g. Hyperterm.
- Disable emitting of messages using the config.sys parameters.

Implementation Hints

Note: These implementation hints provide key points for the experienced Verix developer, it does not cover the complete list of actions. eg. we do not say in this exercise that all components following VMAC are to be downloaded partially.

1	You will need a terminal emulator set up in your PC to display the messages emitted. The easiest way to do this, if you have followed the hints in exercise 1, you will have downloaded the VMAC components in debug mode so you can first capture the messages emitted by them. Setup your hyper term with the following properties for your connection Bits per second: 115200 Data Bits: 8 Parity: None Stop Bits: 1 Flow Control: None Once the setup is complete and you make the connection, you should be able to see messages emitted by the VMAC components.
2	To disable the messages from VMAC Components change in group 1 the following parameters FRONTENDLOG=N (from C) DEVMANLOG=N(from C) IMMLOG=N(from C)
3	Add LOGPRINTF messages in the application of your choice. Verify if your build setting has the LOGSYS_FLAG defined. Rebuild the application and partially download it to the terminal. Terminal. Start the Hyperterm (terminal emulator) on your PC and it should display your LOGPRINTF message.

Lab 13: Sending / Receiving Custom Events between Applications

Goal

To download and execute all the sample applications that are packaged with VMAC 1.2.

Objectives

After completing this exercise, the participant should be able to

- Define custom events in applications.
- Use EESL API to send and receive custom events in applications

Implementation Hints

Note: These implementation hints provide key points for the experienced Verix developer, it does not cover the complete list of actions.
In this particular exercise you will want to modify the application that is based on the library that you normally use in your application development. Consequently you may choose to operate only with a few applications as against all the eight applications. The point to be appreciated in these exercises is that we can use the custom events / EESL / DevMan to realize a system requirement using more than one independent application. The participant is encouraged to try other different combinations by referring to the documentation.

1	Select a pair of applications that you will want to work on.
2	Modify one application to behave as the sender of custom event and the other as the receiver of custom event.
3	Define a custom event with event ID in the custom event range say 11025 and call it **`CUSTOM_TEST_EVENT`** in both the applications.
4	In the exercise that we are implementing we want to perform a printing action in the receiver side based on the event from the sender. If we do not get the printer device we display a message that says the printer is not available.
5	In the sender application use the API - EESL_SendEvent() to send the **`CUSTOM_TEST_EVENT`** with a pointer to the data to be printed in the event data.

6

In the receiver application

- Ensure that you have an entry in the device mapping table that lets you gain control of the printer device (COM4) by posting the appropriate input event.
- In the event handler for the output event open the printer and prepare it for printing by using the appropriate library API (ACT or SDK)
- If COM4 is not available the receiver will get an event **`APP_RES_SET_UNAVAILABLE`**. In the handler for this event display the message "Printer not available".
- If the receiver is an SDK application Wait for evt_pipe and on receipt of evt_pipe, use the EESL API **`EESL_read_cust_event`** to check for your incoming events. In the case of custom events like the **`CUSTOM_TEST_EVENT`** Use it to print as you normally print in your applications.
- If the receiver is an ACT application you do not have to execute the **`wait_event`** API call it will be handled by the application Idle engine You just need to provide the procedure for PIPE_IN event and execute the **`EESL_read_cust_event`** in that procedure.

The quickest way to implement this lab is to identify the application that has printing capability e.g. VMACex2 and send the custom event to these applications and modify them to print in response to the received custom event. This in a way reflects the real world, where in applications are called upon to make use of services provided by other applications or be a service provider.

If you want to implement a different flavor, please discuss with the instructor.