

Verix eVo Porting Guide



Verix eVo Porting Guide
© 2010 VeriFone, Inc.

All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form without the written permission of VeriFone, Inc.

The information contained in this document is subject to change without notice. Although VeriFone has attempted to ensure the accuracy of the contents of this document, this document may include errors or omissions. The examples and sample programs are for illustration only and may not be suited for your purpose. You should verify the applicability of any example or sample program before placing the software into productive use. This document, including without limitation the examples and software programs, is supplied "As-Is."

VeriFone, the VeriFone logo, VeriCentre, Verix V, Verix eVo, VeriShield, VeriFind, VeriSign, VeriFont, and ZonTalk are registered trademarks of VeriFone. Other brand names or trademarks associated with VeriFone's products and services are trademarks of VeriFone, Inc.

Comments? Please e-mail all comments on this document to your local VeriFone Support Team.

Acknowledgments

RealView is a registered trademark of ARM Ltd. For information and ARM documentation, visit: www.arm.com

VISA is a registered trademark of VISA USA, Inc.

All other brand names and trademarks appearing in this manual are the property of their respective holders.

VeriFone, Inc.
2099 Gateway Place, Suite 600
San Jose, CA, 95110 USA.
1-800-VeriFone
www.verifone.com



CONTENTS

PREFACE	5
Prerequisites	5
Organization	5
Audience	5
Assumptions About the Reader	5
Conventions and Acronyms	6
Document Conventions	6
Acronyms	7
Related Documentation	7
 CHAPTER 1	
Verix V and Verix eVo Platform Comparison	
Verix V and Verix eVo Libraries	9
Verix V Libraries and Applications	9
Verix eVo Libraries and Applications	9
VCS and NCP Menu Comparison	11
Network Connections	13
 CHAPTER 2	
Verix eVo SW Programming	
CommEngine Client Application	15
TCP/IP and SSL Application	17
SSL Certificate Verify Callback Function	21
SSL Session Resumption	24
 CHAPTER 3	
Comparison of Verix V and Verix eVo APIs	
Verix V and Verix eVo TCP/IP API Comparison	25
Verix V and Verix eVo SSL Functionalities	28
 CHAPTER 4	
Verix eVo Notes	
Programming Note	29
Compiler Flag Notes	29
Building Shared Libraries	29
Linker	29
Version Compatibility	29
Compiler Flags	29
Disabling Verix eVo Components	30



This document describes VeriFone application development difference between Verix V and Verix eVo platform. It also addresses the architectural differences, new platform interfaces, and library differences between the two platforms. This guide:

- Describes how existing functionalities are supported in the Verix eVo platform.
- Explains the basic functionalities of the Verix eVo TCP/IP stack and SSL library.

Prerequisites

The following are the prerequisites in migrating Verix V applications to Verix eVo program:

- the application should be able to run on 6MB+ terminals.
- It must use RVDS 4.0 compiler.
- It must use `-b` compiler flag for Predator/Trident application compatibility. See [Compiler Flag Notes](#) for more information.

NOTE



Although this manual contains some operating instructions, please refer to the reference manual for your transaction terminal for complete operating instructions.

Organization

This guide is organized as follows:

- | | |
|---------------------------|---|
| Chapter 1 | Discusses Verix V and Verix eVo platform comparison. |
| Chapter 2 | Provides users with sample codes specific to programming in Verix eVo software. |
| Chapter 3 | Presents the Verix eVo TCP/IP stack and the Verix eVo SSL table of functionalities. |
| Chapter 4 | Details the Verix eVo programming notes and limitations. |

Audience

This document guides **application developers** who want to learn more about, or are involved in, the migration/porting of Verix V applications to the Verix eVo platform.

Assumptions About the Reader

It is assumed that the reader:

- Has basic knowledge of Verix eVo components such as the Verix eVo CommEngine (CE/CEIF) library and the Network Control Panel (NCP) application
- Knowledge of existing Verix V TCP/IP (+SSL), UCL, VMAC, and VCS libraries and applications




Conventions and Acronyms

This section provides reference to conventions and acronyms used in this manual, and discusses how to access the text files associated with code examples.

Document Conventions

Various conventions are used to help you quickly identify special formatting. Table 1 describes these conventions and provides examples of their use.

Table 1 Document Conventions

Convention		
Blue	Text in blue indicates terms that are cross referenced.	See Conventions and Acronyms .
<i>Italics</i>	Italic typeface indicates book titles or emphasis.	You <i>must</i> install a roll of thermal-sensitive paper in the printer.
Courier	The courier type face is used while specifying onscreen text, such as text that you would enter at a command prompt, or to provide an URL.	<code>http://www.verifone.com</code>
	NOTE The pencil icon is used to highlight important information.	RS-232-type devices do not work with the PINpad port.
	CAUTION The caution symbol indicates possible hardware or software failure, or loss of data.	The terminal is not waterproof or dustproof, and is intended for indoor use only.
	WARNING The lightning symbol is used as a warning when bodily injury might occur.	Due to risk of shock do not use the terminal near water.

Acronyms

Various acronyms are used in place of the full definition. The acronyms used in this manual are listed in [Table 2](#).

Table 2 **Acronyms**

Acronym	Definition
ANSI	American National Standards Institute
APDU	Application Protocol Data Units
API	Application Programming Interface
ASCII	American Standard Code For Information Interchange
CE	CommEngine
CEIF	CommEngine Interface
DDI	Device Driver Interfaces
DDL	Direct Download Utility
DLL	Dynamically Loaded Library
DTK	Development Toolkit
MIB	Management Information Block
NCP	Network Control Panel
NWIF	Network Interfaces
OSDL	Operating System Download
RAM	Random Access Memory
SRAM	Static Random-access Memory
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
VCS	Verix Communication Server
VLR	Variable-length Record
VMAC	Verix Multi-application Conductor
VPN	VeriFone Part Number
VSS	VeriShield Secure Script
Wi-Fi	Wireless Fidelity

Related Documentation

To learn more about Verix eVo and the Verix eVo Communication Server application, refer to the following set of documents:

Verix eVo Volume I: Operating System Programming Manual	VPN DOC00301
Verix eVo Volume II: Operating System and Communication Programmers Manual	VPN DOC00302
Verix eVo Volume III: Operating System Programming Tools Reference Manual	VPN DOC00303



Verix V and Verix eVo Platform Comparison

This section lists the Verix V and Verix eVo main libraries that are used for terminal applications development on both platforms. This also aims to give an overview on how they differ in terms of available libraries and applications for software development and network connections management.

Verix V and Verix eVo Libraries

Table 3 shows the different libraries for Verix V and Verix eVo applications.

Table 3 Verix V and Verix eVo Libraries

Verix V	Verix eVo
UCL	DDI (DLM/Driver)
TCP/IP (with SSL)	CE/CEIF Library
IPDownload	TCP/IP
VMAC	SSL Shared Library
VCS	OSDL
LogSys	VMAC
	VCS
	NCP
	EvOLog

Verix V Libraries and Applications

The following are the different Verix V libraries that have been discontinued or replaced because of the updates done for Verix eVo program.

UCL

Verix V UCL is not used under the Verix eVo platform.

TCP/IP (with SSL)

Verix V TCP/IP library will be replaced with the new Verix eVo TCP/IP stack and shared library.

IPDownload

Verix V IPDL library will be replaced with the new OSDL.

VCS

A new instantiation for Verix eVo application will be used for Verix eVo platform.

Verix eVo Libraries and Applications

The following are the different Verix eVo libraries and applications.

DDI (DLM/Driver)

DDI drivers are not directly used by applications but mentioning them relevant to a comparison between Verix V and Verix eVo. These are downloadable modules which are used by CommEngine (CE) to start or initiate and configure network interfaces or communication devices. Specific DDI drivers are loaded at runtime by CE depending on the media supported by the terminal being used. In Verix V platform, UCL library is used to initialize communication devices. But for Verix eVo applications, communication devices are initialized and started by CE thru the use of DDI drivers.

CE/CEIF Library

This is mainly used by applications to interface with CE applications. It is used to register CE, configure NWIFs, start or stop NWIFs, and query network status and reads/writes to the NWIF's communication device. These functionalities are somewhat similar to UCL on the Verix V platform. With the application registering to CE.out, the application may receive and manage network notifications and events which allows it to do necessary actions based on the received events.

TCP/IP

Verix eVo TCP/IP stack contains BSD standard TCP/IP programming APIs.

TCP/IP library does not include SSL-specific functionalities.

SSL Shared Library

An OpenSSL ported library used for SSL connection.

OSDL

IPDL features and APIs will be included as part of OS downloads support. The following APIs are available for OS download support:

- `int download_net (int sock, int *SSLstruc, const void *parms)`
- `int SVC_ZONTALK (unsigned char type)`
- `int SVC_ZONTALK_NET (unsigned char type)`
- `short shRegisterExtendedProcessing (fpshPreProcess, fpshPostProcess PostProcess)`
- `short shRegisterNotifyCb(fpNotify CbNotigy, short usermask)`
- `short shSetParam (unsigned short shParamID, void *ParamValue)`

See *Verix eVo Volume I: Operating System Programmer Manual, VPN-DOC00302* for more details about OSDL APIs.

VMAC

Verix V VMAC applications and library will still be used for Verix eVo platform, except that a new VMAC-CE interface application is needed to be downloaded into the terminal. This application is called VMACIF.out. VMACIF.out serves as a mediator between VMAC and CE to avoid conflicts in acquiring communication devices.

VCS

The Verix eVo Communication Server application registers to CE to receive and manage network notifications which enables VCS to do necessary actions based on the received events. This supports the existing VCS client message interfaces and functionality. However, it now utilizes the new Verix eVo TCP/IP stack and SSL library.

Only one VCS Instantiation is provided and it can be used for any network device type (Ethernet, GPRS, Wi-Fi, CDMA, and BT). VCS for Verix eVo application will not have any UI and will run as a background application.

VCS Media Switching

VCS Media Switching can be translated in Verix eVo program as the stopping and starting of specific network interfaces (NWIFs). Verix eVo program can have multiple NWIFs at the same time, with each NWIF managed independently. However, when VCS media switch is requested, it does not need to stop the currently used media/NWIF unless restricted due to hardware exclusivity issue.

NCP

This application provides a user interface for network administration, monitoring, diagnostics, and configurations, and can be used as a user interface for accessing VCS functionality.

VCS and NCP Menu Comparison

The existing VCS menus contain information and options for network configuration, TCP/IP and SSL download, media switching, and ping for network diagnostics. These VCS menus/options are also present in NCP.

Table 4 contains the VCS menus and the corresponding NCP menu which contains the same or almost the same functionality.

Table 4 Differences Between VCS and NCP Menus

VCS Menu	NCP Menu	Comparison/Description
Setup	Setup -> Communication Technology and Setup -> Device Drivers	<p>VCS Setup menu, depending on the media to be used, contains options to set network configurations such IP, Subnet, Gateway, and DNS IP addresses. It also has min/max TX timeout and max TX retry options. Once modified and saved, the terminal restarts to reflect its changes on the next connection.</p> <p>NCP also has network configuration options/setting except for the timeouts which can be set using config variables. Once modified, no terminal restart is needed, but it is necessary to manually restart the NWIF thru NW Maintenance menu to reflect the changes.</p> <p>Note: VCS and NCP NW configurations vary depending on the media/ NWIF supported by the terminal.</p>
Download Setup	Tools -> Download	<p>Both VCS and NCP has an option to set GID, Full/Partial, Server IP, and Port No. under this menu.</p> <p>In addition to these settings, NCP has the option to select TCP or SSL DL, to set Port No., App ID, and Term ID. App ID and Term ID in this menu corresponds to *ZA and *ZT in the environment variable which is used by VCS.</p> <p>Once settings are entered in NCP, download can be started under the same menu while VCS needs to select "TCP Download" or "SSL Download" menu to start the download process.</p>
TCP Download	Tools -> Download	<p>VCS menu to start TCP download.</p> <p>See NCP Tools ->Download details above.</p>
SSL Download	Tools -> Download	<p>VCS menu to start SSL download.</p> <p>See NCP Tools ->Download details above.</p>
IP Status	Terminal Info -> IP Addresses Status	<p>This menu shows the IP address information of the current connection. This displays the IP address, Subnet Mask, Gateway, and DNS addresses.</p> <p>If applicable, NCP also displays the DHCP lease start and end time.</p>

Table 4 Differences Between VCS and NCP Menus

VCS Menu	NCP Menu	Comparison/Description
Ping	Tools -> Diagnostics - > Ping IP Address	The main purpose of this menu is to check if the terminal is connected to a network. NCP has the option for single or continuous ping (press Cancel to stop continuous ping.)
Media Switch	Tools -> Network Maintenance	VCS shows the available media that it switched to whether Ethernet, GPRS, Wi-Fi, CDMA or BT. Once a new media is selected, the terminal restarts. NCP's network maintenance lists all available Network Interfaces (NWIF). Users can Start, Stop, and Restart the specific NWIF.
About	Terminal Info -> Versions	VCS Menu shows the version number of VCS and version of the library linked with it such as TCPIP, UCL, IPDL, and VMAC version. NCP Menu that shows the version numbers of NCP, CE, CEIF library, DDI Drivers etc.

Network Connections

Established network connections can be used by different applications transparently for TCP/IP communications when NWIFs are configured and started. Communication devices do not need to be acquired or to be owned by an application to do network transactions. Verix eVo application can manage multiple NWIFs running at the same time.



Verix eVo SW Programming

This section discusses some sample codes specific to programming in Verix eVo software such as codes for handling CommEngine events, managing and controlling the connection, and SSL certificate to verify callback function.

CommEngine Client Application

In the sample code below, the application starts the CommEngine incrementally and notifies it after the network interface is in OPEN state, `ceStartNWIF(CE_OPEN)`. CommEngine notifies the application by sending event `CE_EVT_START_OPEN`. At this point the application sends a command to the device to obtain its ICCID (assuming it is a GPRS device) via API `ceExCommand()`. It then starts the network by API `ceStartNWIF(CE_CONNECT)`.

Function `process_CEEvent()` illustrates how the application can be managed using the events from CommEngine. The signal strength event is used to update the display.

The sample code below also illustrates how an application enables and manages events using `ceAPI ceEnableEventNotification()`.

```
// Starting the Network and managing
// application using CommEngine Events

#include <svc.h>
#include <ceif.h>

#define MEDIA_DIAL 1
#define MEDIA_IP 2
unsigned short Flag_Process_IP_Transactions;
short commMedia; // Dial or IP
unsigned short niHandle;
process_CEEEvent(void)
{
    char cmdStr[] = {"AT^SCID"};
    char cmdResp[512];
    unsigned int cmdRespLen;
    unsigned int rc;
    stceNWEvt ceEvt;

    rc = ceGetEvent(&ceEvt, 0, NULL, NULL);

    switch (ceEvt.neEvt)
    {
        case CE_EVT_NET_UP: // Network is up
            Flag_Process_IP_Transactions = TRUE;
            commMedia = MEDIA_IP;
            break;

        case CE_EVT_NET_DN: // Network is down
            Flag_Process_IP_Transactions = FALSE;
            commMedia = MEDIA_DIAL;
            break;

        case CE_EVT_SIGNAL: // Signal Strength
            display_signal_icon(&ceEvt);
            break;

        case CE_EVT_START_OPEN: // Device ready for commands
            // Send command to device. Fetch ICCID
            ceExCommand(niHandle, cmdStr, 8, 512,
                cmdResp, &cmdRespLen, 30000);
    }
}
```



```
// Bring up the network
    ceStartNWIF(CE_CONNECT);
    break;
}
}

main(int argc, char *argv[])
{
    long osEvent;

    // Register with CommEngine
    ceRegister();

    // Init environment for CommEngine events
    ceEnableEventNotification();

    // Incremental start
    ceStartNWIF(CE_OPEN);

    // Process events
    while(TRUE)
    {
        // Wait for event
        osEvent = wait_event(EVT_PIPE);

        if (ceGetEventCount() > 0)
            process_CEEvent();
    }
}
```

TCP/IP and SSL Application

Once applications are registered to CE and functionality is added to manage CE events, it can start NWIFs and use standard BSD socket, TCP/IP, and OpenSSL programming APIs.

Below is sample code of an application using TCP/IP and SSL Verix eVo libraries. SSL-specific codes are shown in bold fonts.

```
int main(int argc, char *argv[])
{
    SSL      *ssl=NULL;
    SSL_CTX *ctx=NULL;
    int rc;
    long err;
    :
    :
    int sockHandle;
    struct sockaddr_in sockHost;
    struct timeval timeout;
    char chHostIP[50] = {"10.64.30.240"};
    int inPort = 5001;
    int isSSL = 0;//set to 1 for SSL connection

    // Register with CommEngine
    ceRegister();
    // Init environment for CommEngine events
    init_CEEvents();
    // add other CEIF API needed here...

    if(isSSL)// initialization
    {
        // call..
        //SSL_library_init()
        //SSL_load_error_strings();
        init_OpenSSL();

        ctx = setup_client_ctx();
        if (!ctx)
        {
            LOG_PRINTF("ERROR in create SSL CTX!!!");
            exit(-1);
        }
    }

    //do-socket-connect
    sockHost.sin_family = AF_INET;
    sockHost.sin_addr.s_addr = htonl (inet_addr (chHostIP));
    sockHost.sin_port = htons (inPort);
    sockHandle = socket (AF_INET, SOCK_STREAM, 0);
```

```

if (sockHandle < 0)
{
    LOG_PRINTF( "Socket creation FAILED: %d. errno: %d",
sockHandle, errno);
    return -1;
}

LOG_PRINTF( "Socket handle [%d]", sockHandle);

retVal = connect(sockHandle, (struct sockaddr*)&sockHost,
sizeof (struct sockaddr_in));

if (retVal != 0)
{
    LOG_PRINTF("Connect failed and retVal: %d errno: %d", retVal,
errno);
    return -1;
}
else
{
    LOG_PRINTF("Connect is successful returned: %d errno: %d",
retVal, errno);
}

if(isSSL)
{
    if(ssl == NULL)
    {
        if ((ssl = SSL_new(ctx)) == NULL)
            LOG_PRINTF("Error creating an SSL context");
    }
//Assign the socket into SSL
    SSL_set_fd (ssl, sockHandle);

    //Do the SSL handshake
    rc = SSL_connect(ssl);
    LOG_PRINTF("SSL_connect->%d\n", rc);

    ulErr = ERR_get_error();
    ERR_error_string_n(ulErr, chErrMsg, sizeof(chErrMsg)-10);
    LOG_PRINTF("SSL_ERROR_CODE [%X][%s]", ulErr, chErrMsg);
}

```

```
// do data transaction (send - recv)
if(isSSL)
{
    SSL_write(ssl, "test", strlen("test"));
    :
    SSL_read(ssl, chRead, sizeof(chRead));
}
else
{
    send(sockHandle, "test", strlen("test"), 0);
    :
    recv(sockHandle, chRead, sizeof(chRead), 0);
}

// disconnect
if(isSSL)
{
    err = SSL_shutdown(ssl);
    SSL_clear(ssl);
    SSL_free(ssl);
    ssl=NULL;
    SSL_CTX_free(ctx);
}
else
{
    socketclose(sockHandle);
}

return 0;
} //main

void init_OpenSSL(void)
{
    LOG_PRINTF("init SSL lib...");

    if (!SSL_library_init())
    {
        LOG_PRINTF("*** init_OpenSSL failed **\n");
        exit(-1);
    }

    SSL_load_error_strings();
}
```

```

SSL_CTX *setup_client_ctx(void)
{
    SSL_CTX *ctx;
    char *cipherStr = "AES256-SHA:AES128-SHA:DES-CBC3-SHA:EDH-DSS-DES-
CBC3-SHA:RC4-SHA:RC4-MD5:NULL-SHA:NULL-MD5";

    ctx = SSL_CTX_new(SSLv3_client_method());
    LOG_PRINTF("SSL_CTX_new %X\n", ctx);

    if(!SSL_CTX_set_cipher_list(ctx,cipherStr))
    {
        LOG_PRINTF("SSL_CTX_set_cipher_list failed!");
    }

    /* Auto retry read/write when renegotitation is requested.
     * This only makes sense when the underlying socket is blocking.
     */
    SSL_CTX_set_mode(ctx, SSL_MODE_AUTO_RETRY);

    if (SSL_CTX_load_verify_locations(ctx, CAFILE, CADIR) != 1)
        LOG_PRINTF ("Error loading CA file and/or directory");

    if (SSL_CTX_set_default_verify_paths(ctx) != 1)
        LOG_PRINTF ("Error loading CA default file and/or
directory");

    /// set certificate verification call back here..
    // SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER,app_verify_cb);

    SSL_CTX_set_verify_depth(ctx, 3);

    return ctx;
}

```

SSL Certificate Verify Callback Function

The certificate verification function returns 0 if certificate verification fails, causing SSL handshake failure. If certificate validation is successful, value of 1 is returned.

There are several important items needed to be considered in verifying a certificate:

- Check certificate validity date.
Either the start date is not valid or validity has expired.
- Check if application is accepting self-signed certificates (depth of 0 certificates).

- CRL checking
- Checking of CA trustees (for chain certificates)

Below is sample code that shows how to set and create SSL certificate verify callback function.

```
        // other validations
switch (ctx->error)
{
    // cert/ca trustees/issuer not found for chained certificates
    case X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT:
case X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY:
        return 0;

    // Not Yet Valid cert field checking
case X509_V_ERR_CERT_NOT_YET_VALID:
    case X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD:
        return 0;

    // Cert expiration checking
case X509_V_ERR_CERT_HAS_EXPIRED:
case X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD:
        return 0;

    // self-signed and/or zero cert chain depth
case X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT:
        return 0;

    // revoked certificates
case X509_V_ERR_CERT_REVOKED:
        return 0;
} //end switch

return 1;
}
```

```

:
:
// set certificate verification callback function
// this must be called after SSL_CTX_new() and before SSL_connect()
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, app_verify_cb);
:
:

:
// setting of CRL file
// this must be called after SSL_CTX_new() and before SSL_connect()
X509_STORE *pStore = NULL;
X509_LOOKUP *pLookup = NULL;

pStore = SSL_CTX_get_cert_store(ctx);

pLookup = X509_STORE_add_lookup(pStore, X509_LOOKUP_file());
X509_STORE_set_flags(pStore, X509_V_FLAG_CRL_CHECK |
X509_V_FLAG_CRL_CHECK_ALL);

retVal = X509_load_crl_file( pLookup, "I:1/MYCRLFILE.PEM",
X509_FILETYPE_PEM);
:
:

// Certification verify callback function definition
int app_verify_cb(int ok, X509_STORE_CTX *ctx)
{
    int errCode = 0xFFFFFFFF;
    int errDepth = 0xFFFFFFFF;
    X509 *errCert = (X509*) NULL;
    // checking self-signed certificate by comparing cert subject and
    issuer names
    X509_NAME_oneline(X509_get_subject_name(ctx->current_cert),
strBuf1, sizeof(strBuf1));
    X509_NAME_oneline(X509_get_issuer_name(ctx->current_cert),
strBuf2, sizeof(strBuf2));
    if(strcmp(strBuf1, strBuf2) == 0)
        return 0; // self-signed cert not allowed, handshake failure
}

```

SSL Session Resumption

When a client and a server establishes an SSL connection for the first time, they need to establish keys to be used for SSL encryption. A new session is created every time a given client and server go through a full key exchange and establish a new master key. Establishing this key takes time, depending on the public key algorithm used. Thus, having to do a key exchange for every client request seriously degrades the performance of a server. To improve performance, SSL contains a "session resumption" feature that allows a client/server pair to skip this time-consuming step if they have already established a master key during a previous connection.

The code below shows the minimal OpenSSL code required for a client to do session resumption. OpenSSL uses a session object to store the session information. `SSL_get1_session()` API is used to get the session for a given SSL object while `SSL_set_session()` is used to set the session to be reused.

```
SAVE SSL session:

SSL_SESSION *sslSession=NULL;
:
:
// after SSL_connect(), save the SSL session
sslSession=SSL_get1_session(ssl);
:
SSL_shutdown(ssl);
SSL_free(ssl);
:
:

REUSE SSL session:
:
//before SSL_connect();
SSL_set_session(ssl, sslSession); /*And resume it*/
:
SSL_connect(ssl);
:
:
```

NOTE



After `SSL_get1_session()` is called, `SSL_SESSION_free(sslSession)` must be called to clear allocated memory if the SSL session is not needed anymore.



Comparison of Verix V and Verix eVo APIs

This chapter discusses the differences in the TCP/IP APIs in the Verix TCP/IP and Verix eVo TCP/IP libraries. It also discusses the differences in the setsockopt/getsockopt APIs among the two platforms and compares the Verix and Verix eVo SSL functionalities.

Verix V and Verix eVo TCP/IP API Comparison

Table 5 shows the differences between the Verix V TCP/IP and Verix eVo TCP/IP libraries.

Table 5 Verix V and Verix eVo TCP/IP APIs

Verix V TCP/IP API	Verix eVo TCP/IP API	Notes
socket	socket	SSL bit set bit available in Verix eVo TCP/IP.
bind	bind	Verix eVo TCP/IP valid port number < 65536
closesocket	socketclose	Verix eVo TCP/IP returns: <ul style="list-style-type: none">• 0 - Success Verix V TCP/IP returns: <ul style="list-style-type: none">• 0 or 1 - Success• 3 - For socket close but host did not send final close (FIN or FIN+ACK)
recv	recv	Verix eVo TCP/IP returns: <ul style="list-style-type: none">• 0 - Number of bytes received• > 0 - EOF or the remote host has closed the connection. Verix V TCP/IP returns: <ul style="list-style-type: none">• >= 0 - Number of bytes received
send	send	Verix eVo TCP/IP has flag <code>MS_DONTWAIT</code> for <code>recv()</code> in non-blocking mode.

Table 5 Verix V and Verix eVo TCP/IP APIs

Verix V TCP/IP API	Verix eVo TCP/IP API	Notes
recvfrom	recvfrom	Verix eVo TCP/IP returns: <ul style="list-style-type: none"> > 0 - Success 0 - EOF
sendto	sendto	Verix V TCP/IP returns: <ul style="list-style-type: none"> >= 0 - Number of bytes received Verix eVo TCP/IP has flag MSG_DONTROUTE for diagnostic or routing programs. Verix eVo TCP/IP has flag MSG_DONTWAIT for recvfrom() in non-blocking mode.
ioctlsocket	socketioctl	
shutdown	shutdown	Verix eVo TCP/IP supports TCP sockets only. Verix V TCP/IP supports all socket types.
select	select	
fcntlsocket	fcntlsocket	Verix eVo TCP/IP uses setsockopt() to set socket options set via fcntlsocket().
getIPAddress	Not Available	ceSertNWParamValue() of CEIF.lib can be used to get network information.
getversion	Not Available	
readsocket	recv	
writesocket	send	
yield	Not Available	Not used for Verix eVo program non-blocking connection feature,
blockingIO	Not Available	Blocking and non-Blocking sockets are supported in both Verix V and Verix eVo TCP/IP
inet_addr	addr_t_inet_addr	
DNSClearCache	Not Available	
addhost		
deletehost		
gethostrecord	Not Available	This API retrieves the host record at the specified index. The index is one-based, i.e, the first host record in the host data file is available at index.
assignDLtoTCP	Not Available	
assignDLExttoTCP	Not Available	
assignUCLtoTCP	Not Available	
getnetconfig	get_ipconfig	
getparam	Not Available	

Table 5 Verix V and Verix eVo TCP/IP APIs

Verix V TCP/IP API	Verix eVo TCP/IP API	Notes
netconfig	set_ipconfig	
netconnect	Not Available	ceStartNWIF() from ceif.lib can be used to connect to a network.
netdisconnect	Not Available	ceStopNWIF() from ceif.lib can be used to disconnect from a network.
netstatus	Not Available	

setsockopt/getsockopt API Options

Table 6 shows the differences in the SOL_SOCKET options difference in the Verix V and Verix eVo TCP/IP libraries. Other existing options under Verix V that are not listed mean that these are available under Verix eVo application as well.

Table 6 SOL_SOCKET Options

SOL_SOCKET	Description
SO_RCVBUF	Default Verix eVo TCP/IP is 8192 bytes.
SO_SNDBUF	Default Verix eVo TCP/IP is 8192 bytes. In Verix eVo TCP/IP, value is rounded up to the next even multiple of MSS after connect(). In Verix TCP/IP, set value is retained even after connect().
SO_CLOSEDELAY	Not available in Verix eVo TCP/IP. Length of time before performing a forced closesocket() API on the socketHandle that initiated the TCP connection termination session.
SO_AUTOMATICFOURWAYCLOSE	Automatically closes the connected socket when peer closes it. Not available in Verix eVo TCP/IP.
SO_PKTRCVTIMEO	Allows inter packet time-out for recv() API to be set. Not available in Verix eVo TCP/IP.
SO_STATUS	Allows status to be determined for a particular socket handle. This is not available in Verix eVo TCP/IP.

Verix V and Verix eVo SSL Functionalities

Table 7 shows a comparison between the Verix V and Verix eVo SSL functionalities.

Table 7 Verix V and Verix eVo SSL Functionalities Comparison

Functionality	Description
TLS1.1 Protocol	Not Available in Verix eVo application.
Cipher suite support for TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	Not Available in Verix eVo application.
DER Certificate Format	Supported in Verix eVo application but not available in Verix.
SSL Initialization	SSL initialization must be implemented in the application.
<ul style="list-style-type: none"> vSSL_Init vSSL_InitEx 	
Certificate Management APIs	Certificate management must be implemented in the application.
<ul style="list-style-type: none"> vSSL_SetCRLFile vSSL_SetCAFile vSSL_SetCAPath vSSL_SetCertFile vSSL_SetKeyFile vSSL_SetServerCertChainDepth vSSL_UpdateCAFile vSSL_verify_hostname getAsn1Time 	Not available in Verix eVo application.
SSL Session Resumption/Management APIs	SSL initialization must be implemented in the application.
<ul style="list-style-type: none"> SetTimeout setSessionCacheMode getSessionID setSSLSessionID 	Not available in Verix eVo application.



Verix eVo Notes

Programming Note

Verix eVo applications are designed with certain prerequisites, including a minimum of 6MB memory (Flash and RAM combined). However, there is no available code that will prevent download of Verix eVo application to devices with less than 6MB of total memory. It is up to the end-user to ensure that the minimum device requirements for Verix eVo program are met prior to downloading a Verix eVo application to a target device. Loading a Verix eVo program to a device with less than 6 MB memory may result in insufficient memory space for applications or even potentially damaging behavior. Additionally, Verix eVo programs are designed to work with a known list of communication modules. Check the full list of devices with valid DDI drivers. Verix eVo applications should not be downloaded to devices with no available valid DDI driver.

Compiler Flag Notes

Important considerations when compiling code are presented below.

Building Shared Libraries

`vrxmlib` similarly supports `-b/-p` options with the same rules as the linker (if the library has any `-p` object files, library needs to be built `-p`).

If the library is position independent, include either the SDK `clibpi.a` (for `-b` libraries) or `clibpi11.a` (for `-p` libraries).

Linker

`vrxcc` linker uses similar `-b/-p` options. If any object file is compiled with `-p`, then it must be link with `-p`.

Also, if the application links in `ceif.o`, then, `elog.o` should also be linked. Failure to do so will result in a system crash (and not a build error).

Version Compatibility

Earlier versions of compilers may not work with Verix eVo application or may result in unpredictable compiled code. For full Verix eVo platform functionality, use RVDS 4.0 or later compilers.

Compiler Flags

Use the compiler flags to set compatibility options in the resulting compiled code. For a more full listing of compiler options, see Chapter 3 of *Verix eVo Volume III Operating System Programming Tools Reference Manual* - VPN DOC00304, but the following flags are critical for setting compatibility between the Verix V and Verix eVo platforms.

- The `-p` option enables ARM-11 compilation and Verix eVo specific features. Applications compiled with `-p` can only be run on Verix eVo terminals and will not run on Verix V terminals.

- Selecting **-b** allows compiled code to run on both ARM 9 and ARM 11 platforms.

NOTE

Compiler flags, **-p** and **-b**, are used when executing `VRXCC.exe` and `VRXLIB.exe` of the latest Verix V SDK.

The following are the conditions regarding **-p** and **-b**'s compatibility with terminals:

- If **-p** and **-b** are not used, application can run on old platforms but will not run on Verix eVo terminals.
- If only **-b** is used, application can run on both platforms but there is a minimum Verix V OS version.
- If only **-p** is used, application can run only on Verix eVo terminals.

To summarize the conditions stated above, **-b** offers a lot of flexibility but if Verix eVo program features are needed, use **-p**.

Disabling Verix eVo Components

To ease transition of solutions or applications from Verix V to the Verix eVo platform, Verix eVo components can be disabled using environment variables. This will enable existing applications to still run on Trident terminals while they are being ported to run on Verix eVo.

`Config.sys` variables in GID 1 for disabling Verix eVo components are as follows:

Table 8 **Config.sys Variables**

Config Variables	Description
*NO.VXCE = 1	Disables CommEngine
*NO.VXNCP = 1	Disables NCP
*NO.CTLS = 1	Disables CTLS driver/application



VeriFone, Inc.
2099 Gateway Place, Suite 600
San Jose, CA, 95110 USA
1-800-VeriFone
www.verifone.com

Verix eVo Porting Guide

