

Verix eVo Volume II: Operating System and Communication

Programmers Guide



Verix eVo Volume II: Operating System and Communication Programmers Guide
© 2013 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, Verix eVo, VeriCentre, Verix, 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



PREFACE	9
Organization	9
Target Audience	10
Assumptions About the Reader	10
Conventions and Acronyms	10
Conventions	11
Acronyms	12
Related Documentation	13
Wireless Applications - Required Actions, Best Practices	14
 CHAPTER 1	
TCP/IP Stack	
Interface	15
Getting Started	15
Simple Socket Application	15
Socket Functions	16
Host Information APIs	16
IP Address Conversion APIs	17
Ping APIs	17
socket()	18
bind()	20
listen()	21
accept()	22
connect()	24
getpeername()	26
getsockname()	27
setsockopt()	28
getsockopt()	37
recv()	45
send()	47
recvfrom()	50
sendto()	52
shutdown()	54
socketclose()	55
socketerrno()	56
select()	57
socketset_owner()	59
socketioctl()	60
getsocketevents()	61
Host Information APIs	63
gethostbyname()	64
gethostbyaddr()	65
DnsGetHostByAddr()	66
DnsGetHostByName()	67
freehostent()	68
DnsSetServer()	69
DnsSetUserOption()	70

	Ping APIs	71
	PingOpenStart()	72
	PingGetStats()	73
	PingCloseStop()	75
	IP Address Conversion APIs	76
	inet_addr()	77
	inet_aton()	78
	inet_ntoa()	79
	FTP Protocol Support	80
	Packet Capture	80
CHAPTER 2		
Secure Sockets Layer (SSL)	OpenSSL API	81
	Include Files	81
	Libraries	82
	Crypto Functions	82
	SSL Functions	82
	Verix eVo SSL Versus VeriFone SSL Library	83
	Best Practices	83
CHAPTER 3		
Configuration Management	Configuration Precedence	85
	Case Sensitivity of Configuration Parameter Values	86
	Configuration Options	86
	Configuration File Format	86
	Internal and External Configuration Files	87
	Verix eVo Configuration Files	87
	Customer Configuration File	87
	Naming Convention and Location	88
	Network Interface Configuration	88
	Bluetooth and WiFi Configuration Files	137
	Bluetooth Configuration File	137
	WiFi Configuration File	139
	WiFi Configuration Reset	141
CHAPTER 4		
CommEngine Interface API (CEIF.lib)	Message Exchange mxAPI & Message Formatting mfAPI	143
	ceAPI Concepts	144
	Device & Device Ownership	144
	Device Driver	144
	Network Interface (NWIF)	145
	Network Connection Process & States	146
	NWIF Events	149
	Multiple NWIFs	149
	ceAPI Summary	151
	ceRegister()	155
	ceUnregister()	156
	ceSetCommDevMgr()	157
	ceRequestDevice()	158
	ceReleaseDevice()	159
	ceGetDDParamValue()	160
	ceSetDDParamValue()	162

ceExCommand()	164
ceEnableEventNotification()	166
ceDisableEventNotification()	167
ceGetEventCount()	168
ceGetEvent()	169
ceSetSignalNotification()	171
ceIsEventNotificationEnabled()	173
ceGetNWIFCount()	174
ceGetNWIFInfo()	175
ceStartNWIF()	176
ceStopNWIF()	178
ceSetNWIFStartMode()	180
ceGetNWIFStartMode()	181
ceSetNWPParamValue()	182
ceGetNWPParamValue()	184
ceSetBattThreshold()	185
ceGetBattThreshold()	186
ceStartDialIF()	187
ceStopDialIF()	189
ceSetDialMode()	190
ceGetDialMode()	191
ceGetDialHandle()	192
ceDualSIMSwitch()	193
ceActivateNCP()	194
ceGetVersion()	195
Constants, Defines & Miscellaneous	196
List of CommEngine Events	196
Constants	198
List of Network Parameters	201
List of Error Codes	205
Extended Error Codes and Messages	209
Application Developer Notes	213
Handling CommEngine Events	213
CE_EVT_DDI_APPL for a GPRS / Mobile Phone (GSM) Driver	213
Sample code on ceSetNWPParamValue()/ceGetNWPParamValue()	214
Sample code on ceSetDDParamValue() / ceGetDDParamValue()	214
Sample code on ceEXCommand()	214
Sample code on ceStartDialIF()	214
Message Exchange mxAPI & Introduction	215
Payload Size	215
Message Exchange mxAPI – Summary	215
mxCreatePipe()	216
mxClosePipe()	217
mxGetPipeHandle()	218
mxSend()	219
mxPending()	220
mxRecv()	221
mxRecvEx()	222
mxAPI Error Codes	223
Message Formatting mfAPI	223
Tag, Length & Value (TLV)	224
TLV List	225

	Message Formatting mfAPI – Summary	225
	mfCreateHandle()	227
	mfDestroyHandle()	228
	mfAddInit()	229
	mfAddClose()	230
	mfAddTLV()	231
	mfFetchInit()	232
	mfFetchClose()	233
	mfFetchReset()	234
	mfPeekNextTLV()	235
	mfFetchNextTLV()	236
	mfFindTLV()	237
	mfEstimate()	238
	mfAPI Error Codes	239
CHAPTER 5		
3G Radio Firmware Update	Requirements for Firmware Update	241
	Firmware Files	241
	Firmware Update Operation	241
	User Prompt For Firmware Update	241
	Console Notification for Incomplete Files	242
	Firmware Update Status	242
	Firmware Update Result	242
CHAPTER 6		
GPS	GPSD Application	243
	GPS APIs	244
	gps_open()	245
	gps_close()	246
	gps_send()	247
	gps_read()	248
	gps_unpack()	249
	gps_waiting()	250
	gps_stream()	251
	gps_data()	252
	gps_errstr()	253
	gps_engine_start()	254
	gps_engine_stop()	255
	Geo-Fence APIs	256
	geof_open()	257
	geof_destroy()	258
	geof_set()	259
	geof_get_count()	260
	geof_get_by_index()	261
	geof_get_by_name()	262
	geof_del()	263
	geof_del_all()	264
	geof_set_from_file()	265
	geof_save_to_file()	266
	geof_check()	267
	Geo-Fence Downloadable File	268
	Data Structures	269

CHAPTER 7

Verix eVo Communication Engine Application

CommEngine Bootstrap Process	273
CommEngine Invocation	273
Conditionally Starting CommEngine	273
CommEngine Startup Operations	274
CommEngine Configuration and Metadata File	274
Network Configuration via Configuration Delta File	274
Network Configuration via GID 1 CONFIG.SYS parameters	276
Configuration Parameter Value Validation Checking	277
CommEngine Logging	278
Interface with Device Drivers	278
Interface with Verix eVo TCP/IP Library	279
Application Interface	279
Application Events	279
CommEngine Auto-reconnection	286
Two-tier Auto-Reconnection Scheme	286
Exempting Connection Failure	289
Configuring Auto-Reconnection Parameters	290
Other Connection Recovery Mechanisms	291

CHAPTER 8

Network Control Panel (NCP)

Startup Operation	294
Starting NCP Executable	294
Interoperability	294
IP Downloads from System Mode	297
Running under Single-Application Mode	297
Running under VMAC Environment	298
Running in VxGUI Mode	298
Configuration Files	298
User Interface	300
Customization	301
Idle Screen	303
Menu: Tools	303
Menu: Terminal Info (Terminal Information)	309
Menu: Setup	314
Menu: WiFi	321
Menu: Bluetooth	321
LogSpooler	321
Enabling Logs via LogSpooler	322
LogSpooler Settings	322

CHAPTER 9

WiFi Control Panel (WCP)

Startup Operation	325
Starting WCP Executable	325
Invoking and Exiting WCP	325
Configuration Files	325
User Interface	325
Menu: Information	325
Menu: Setup	326

CHAPTER 10	
Bluetooth Control Panel (BCP)	
Startup Operation	329
Starting BCP executable.	329
Invoking and Exiting BCP	329
Configuration Files	329
User Interface	330
Menu: Information	330
Menu: Setup	331
 CHAPTER 11	
Verix eVo Log Library	
Library Files	333
LOGSYS_FLAG	334
LOGSYS_INIT	335
LOG_PRINTF, LOG_PRINTF_MOD	336
LOG_PRINTF, LOG_PRINTF_MOD	337
LOG_INVALID_POINTER, LOG_INVALID_POINTER_MOD	338
Macros for Reporting Errors	340
Re-assigning Log Settings	342
Config.Sys Variables	343
Samples	344
Log Tracing Best Practices.	345
Backwards Compatibility Considerations Summary	346
 GLOSSARY	349
 INDEX	351



This communications manual supports the Development Toolkit (DTK) for the Verix eVo Volume II: Operating System and Communications transaction terminals. This manual:

- Describes the programming tools for the communications environment.
- Provides descriptions of the `CONFIG.SYS` variables.
- Provides API descriptions and code examples.
- Provides discussion on system and communication devices.
- Provides descriptions of the security features.
- Describes working with the IPP (internal PIN pad).
- Provides information on downloading applications into a Verix eVo terminal.

Verix eVo terminals are designed to support many types of applications, especially in the point-of-sale environment. Applications are written in the C programming language and run in conjunction with the Verix eVo operating system. This manual is designed to help programmers develop those applications.

This manual also contains explicit information regarding the Application Programming Interfaces (APIs) within the Verix eVo operating system, and with optional peripheral or internal devices.

NOTE



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

Organization

This document is organized as follows:

[Chapter 1, TCP/IP Stack](#) - discusses the settings for the TCP/IP Stack.

[Chapter 2, Secure Sockets Layer \(SSL\)](#) - discusses the usage of Secure Sockets Layer (SSL).

[Chapter 3, Configuration Management](#) - discusses the standardized configuration file format to use across all Verix eVo components requiring external configuration/customization.

[Chapter 4, CommEngine Interface API \(CEIF.lib\)](#) - discusses the usage of CommEngine Interface API (`CEIF.lib`).

[Chapter 6, GPS](#) - discusses the provided libraries for GPS support, GPS APIs, Geo-Fence APIs and downloadable files, and data structures used for GPS operations.

[Chapter 7, Verix eVo Communication Engine Application](#) - discusses the features and functionality of the Verix eVo Communication Engine Application (VXCE.out).

[Chapter 8, Network Control Panel \(NCP\)](#) - discusses the functionality of the Network Control Panel (NCP) in the Verix eVo system.

[Chapter 9, WiFi Control Panel \(WCP\)](#) - discusses the functionality of the WiFi Control Panel (WCP) in the Verix eVo system.

[Chapter 10, Bluetooth Control Panel \(BCP\)](#) - discusses the functionality of the Bluetooth Control Panel (BCP) in the Verix eVo system.

[Chapter 11, Verix eVo Log Library](#) - discusses the mechanism for logging trace information.

Target Audience

This document is of interest to application developers creating applications for use on Verix eVo-based terminals.

Assumptions About the Reader

It is assumed that the reader:

- Understands C programming concepts and terminology.
- Has access to a PC running Windows 2000 or Windows XP.
- Has installed the Verix eVo DTK on this machine.
- Has access to Verix eVo Volume II: Operating System and Communications development terminal.

Conventions and Acronyms

The following conventions assist the reader to distinguish between different kinds of information.

- The `courier` typeface is used for code entries, filenames and extensions, and anything that requires typing at the DOS prompt or from the terminal keypad.
- The *italic* typeface indicates book title or emphasis.
- Text in [blue](#) indicates terms that are cross-referenced. When the pointer is placed over these references the pointer changes to the finger pointer, indicating a link. Click on the link to view the topic.

NOTE



Notes point out interesting and useful information.

CAUTION



Cautions point out potential programming problems.

Conventions The following table lists the various conventions used throughout this manual.

Table 1 **Conventions**

Abbreviation	Definition
A	ampere
b	binary
bps	bits per second
dB	decibel
dBm	decibel meter
h	hexadecimal
hr	hours
KB	kilobytes
kbps	kilobits per second
kHz	kilohertz
mA	milliampere
MAX	maximum (value)
MB	megabytes
MHz	megahertz
min	minutes
MIN	minimum (value)
ms	milliseconds
pps	pulse per second
Rx	Receive
s	seconds
Tx	Transmit
V	volts

Acronyms The following table lists the various acronyms used in this manual.

Table 2 Acronyms

Acronym	Definition
ACK	Acknowledge
ACT	Access Technology
API	Application Program Interface
APN	Access Point Name
ARFCN	Absolute Radio Frequency Channel Number
ASCII	American Standard Code For Information Interchange
CDMA	Code Division Multiple Access
CE	Communication Engine
ceAPI	CommEngine Interface API
CEDM	CommEngine Device Management
CELL	Cell ID
CHAP	Challenge-Handshake Authentication Protocol
CRLF	Carriage Return Line Feed
DDI	Device Driver Interface
DDL	Direct Download Utility
DHCP	Dynamic Host Configuration Protocol
DLL	Dynamic Link Library
DNS	Domain Name System
DTK	Development Toolkit. See <i>VVDTK</i> .
EGPRS	Enhanced GPRS
EMV	Europay Mastercard and Visa
EOF	End Of File
EVDO	Evolution-Data Optimized
GPRS	General Packet Radio Service
GPS	Global Positioning System
GPSD	GPS Daemon
GSM	Global System for Mobile Communications
HSDPA	High Speed Downlink Packet Access
HSPA	High Speed Packet Access
ICC	Integrated Circuit Card; Smart Card
IEEE	Institute Of Electrical And Electronics Engineers
IGMP	Internet Group Management Protocol
ICMP	Internet Control Message Protocol
I/O	Input/Output
IOCTL	Input Output Control
IPP	Internal PIN Pad
LAC	Location Area Code
LAI	Location Area Identification
LAN	Local Area Network

Table 2 **Acronyms** (continued)

Acronym	Definition
LCD	Liquid Crystal Display
MAC	Message Authentication Code
MCC	Mobile Country Code
MNC	Mobile Network Code
MTU	Maximum Transmission Unit
MUX	Multiplexor
NMEA	National Marine Electronics Association
NWIF	Network Interface
OS	Operating System
PAP	Password Authentication Protocol
PCI	Payment Card Industry
PDU	Protocol Description Unit
PIN	Personal Identification Number
PLMN	Public Land Mobile Network
PPP	Point-to-Point Protocol
PUK	Personal Unblocking Key
RAM	Random Access Memory
SDK	Software Development Kit
SDLC	Synchronous Data Link Control
SIM	Subscriber Identity Module
SMS	Short Message Service
SvcState	Service State
TCP/IP	Transmission Control Protocol/Internet Protocol
TLV	Tag, Length, and Value
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
WCDMA	WideBand Code Division Multiple Access
WiFi	Wireless Fidelity

Related Documentation

To learn more about the Verix eVo Volume II: Operating System and Communications Solutions, refer to the following set of documents:

- *Verix eVo Volume I: Operating System Programmers Manual*, VPN DOC00301
- *Verix eVo Volume III: Operating System Programming Tools Reference Manual*, VPN DOC00304
- *Verix eVo Porting Guide*, VPN DOC00305
- *Verix eVo Multi-App Conductor Programmers Guide*, VPN DOC00306

Wireless Applications - Required Actions, Best Practices

- *Verix eVo Communications Server Instantiation Guide*, VPN DOC00308
- *Verix eVo Development Suite Getting Started Guide*, VPN DOC00309

Detailed operating information can be found in the reference manual for your terminal. For equipment connection information, refer to the Reference Manual or Installation Guides.

For applications that employ a wireless interface, application writers must familiarize themselves with general risks associated with attacks particular to the wireless interface. Moreover, application writers must employ the direction specified by Mastercard Worldwide to ensure the wireless interface is operating in a secure manner.

The Mastercard Worldwide Site Data Protection (SDP) Program, with the PCI DSS as its foundation, details the data security requirements and compliance validation requirements to protect stored and transmitted MasterCard payment account data.

The PCI DSS is designed to identify vulnerabilities in security processes, procedures, and Web site configurations. PCI DSS compliance and subsequent compliance with the SDP Program mandate, helps merchants, Service Providers, and customers protect themselves against security breaches, enhance consumer confidence, and protect the integrity of the overall payment system.

For more information, go to: http://www.mastercard.com/us/company/en/whatwedo/site_data_protection.html.

Table 3 lists the applicable wireless interfaces and includes pointers within the Mastercard Worldwide manual to detail the security concerns and the accepted operational modes necessary to mitigate them.

Table 3 Mastercard Worldwide Security Risks and Guidelines

Wireless Interface	Basic Information	Security Risks	Security Guidelines
WiFi	2-2	2-2 through 2-3	2-4 through 2-7
GPRS and Mobile Phone (GSM)	2-12	2-12	2-13
Bluetooth	2-14	2-14 through 2-15	2-16

NOTE



Visa, Payment Card Industry (PCI), and/or other important entities may offer documents detailing requirements as well. Application writers should inquire with any such pertinent entities for documentation before initiating application development.



TCP/IP Stack

This chapter discusses the settings for the TCP/IP Stack.

Interface

The first two items are part of the Verix eVo SDK. The parameters are listed below:

Parameter	Description
<code>svc_net.h</code>	Eventual home is <code>vrxsdk\include</code> so <code>"#include <svc_net.h>"</code> will work
<code>svc_net2.h</code>	
<code>svc_nwi.h</code>	
<code>svc_net.o</code>	Eventual home is <code>vrxsdk\lib</code> .
<code>ipstack.bin</code>	To be included with Verix eVo.

Getting Started

The network device must first be opened then handed over to the stack. Once this has been done, applications may start using sockets. The code sequences below do not include any error checking to keep the code short and easy to understand. Production code should include error checking.

Simple Socket Application

The following is a sample code for a TCP application.

[TCP_App.txt](#)

(Click to view file attachments)

The following is a sample code for a UDP application.

[UDP_App.txt](#)

(Click to view file attachments)

Socket Functions

Use the following socket functions:

- `socket()`
- `bind()`
- `listen()`
- `accept()`
- `connect()`
- `getpeername()`
- `getsockname()`
- `setsockopt()`
- `getsockopt()`
- `recv()`
- `send()`
- `recvfrom()`
- `sendto()`
- `shutdown()`
- `socketclose()`
- `socketerrno()`
- `select()`
- `socketset_owner()`
- `socketioctl()`
- `getsocketevents()`

Host Information APIs

Listed below are existing host information APIs:

- `gethostbyname()`
- `gethostbyaddr()`
- `DnsGetHostByAddr()`
- `DnsGetHostByName()`
- `freehostent()`
- `DnsSetServer()`
- `DnsSetUserOption()`

IP Address Conversion APIs

Listed below are existing IP address conversion APIs:

- `inet_addr()`
- `inet_aton()`
- `inet_ntoa()`

Ping APIs

Listed below are Ping function APIs:

- `PingOpenStart()`
- `PingGetStats()`
- `PingCloseStop()`

socket()

`socket()` creates an endpoint for communication and returns a descriptor. The family parameter specifies a communications domain in which communication will take place; this selects the protocol family that should be used. The protocol family is generally the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file `<sys/types.h>`. If protocol has been specified, but no exact match for the tuple family, type, and protocol is found, then the first entry containing the specified family and type with zero for protocol will be used. The currently understood format is `PF_INET` for ARPA Internet protocols.

Currently defined types are:

- `SOCK_STREAM`
- `SOCK_DGRAM`
- `SOCK_RAW`

A `SOCK_STREAM` type provides sequenced, reliable, two-way connection-based byte streams. An out-of-band data transmission mechanism is supported. A `SOCK_DGRAM` socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length); a `SOCK_DGRAM` user is required to read an entire packet with each `recv` call or variation of `recv` call, otherwise an error code of `EMSGSIZE` is returned. protocol specifies a particular protocol to be used with the socket. Normally, only a single protocol exists to support a particular socket type within a given protocol family. However, multiple protocols may exist, in which case, a particular protocol must be specified in this manner.

The protocol number to use is particular to the "communication domain" in which communication is to take place. If the caller specifies a protocol, then it will be packaged into a socket level option request and sent to the underlying protocol layers. Sockets of type `SOCK_STREAM` are full-duplex byte streams. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with `connect()` on the client side. On the server side, the server must call `listen()` and then `accept()`. Once connected, data may be transferred using `recv()` and `send()` calls or some variant of the `send()` and `recv()` calls. When a session has been completed, a close of the socket should be performed. The communications protocols used `socketclose()` to implement a `SOCK_STREAM` ensure that data is not lost or duplicated.

If a piece of data (for which the peer protocol has buffer space) cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with `-1` return value and with `ETIMEDOUT` as the specific socket error. The TCP protocols optionally keep sockets active by forcing transmissions roughly every two hours in the absence of other activity. An error is then indicated if no response can be elicited on an

otherwise idle connection for an extended period (for instance 5 minutes). SOCK_DGRAM or SOCK_RAW sockets allow datagrams to be sent to correspondents named in `sendto()` calls. Datagrams are generally received with `recvfrom()` which returns the next datagram with its return address. The operation of sockets is controlled by socket level options. These options are defined in the file `<svc_net.h>`. `setsockopt()` and `getsockopt()` are used to set and get options, respectively.

NOTE

*SOCKET allocates the maximum number of open sockets in the IP stack. The allocation takes place only on restart.

Min 4, default 32, Max 32

Prototype

```
int socket (int family, int type, int protocol);
```

Parameters

Parameter	Description
family	The protocol family to use for this socket.
type	The type of socket.
protocol	The layer 4 protocol to use for this socket.

Family	Type	Protocol	Actual Protocol
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
AF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
AF_INET	SOCK_RAW	IPPROTO_IGMP	IGMP

Return Values

New socket descriptor or -1 on error. If an error occurred, the socket error can be retrieved by calling `socket_errno()` and using `SOCKET_ERROR` as the socket descriptor parameter.

`socket()` can fail for any of the following reasons:

Parameter	Description
EMFILE	No more sockets are available
ENOBUFS	There was insufficient user memory available to complete the operation
EPROTONOSUPPORT	The specified protocol is not supported within this family
EPFNOSUPPORT	The Protocol family is not supported.
EAFNOSUPPORT	The Address family is not supported.

bind()

`bind()` assigns an address to an unnamed socket. When a socket is created with `socket()`, it exists in an address family space but has no address assigned. `bind()` requests that the address pointed to by `addressPtr` be assigned to the socket. Clients do not normally require that an address be assigned to a socket. However, servers usually require that the socket be bound to a standard address. The port number must be less than 32768 (`SOC_NO_INDEX`), or could be `0xFFFF` (`WILD_PORT`). Binding to the `WILD_PORT` port number allows a server to listen for incoming connection requests on all the ports. Multiple sockets cannot bind to the same port with different IP addresses (as might be allowed in UNIX).

Prototype

```
int bind (int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);
```

Parameters

Parameter	Description
<code>sockfd</code>	The socket descriptor to assign an IP address and port number to.
<code>addressPtr</code>	The pointer to the structure containing the address to assign.
<code>addressLength</code>	The length of the address structure.

Return Values

Value	Description
0	Success
-1	An error occurred

`bind()` can fail for any of the following reasons:

Value	Description
<code>EADDRINUSE</code>	The specified address is already in use.
<code>EBADF</code>	<code>sockfd</code> is not a valid descriptor.
<code>EINVAL</code>	One of the passed parameters is invalid or socket is already bound.

listen()

To accept connections, a socket is first created with `socket()`, a backlog for incoming connections is specified with `listen()` and then the connections are accepted with `accept()`. The `listen()` call applies only to sockets of type `SOCK_STREAM`. The `backLog` parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full, and the underlying protocol supports retransmission, the connection request may be ignored so that retries may succeed. For `AF_INET` sockets, the TCP retries the connection. If the backlog is not cleared by the time the TCP times out, connect fails with `ETIMEDOUT` return value.

Prototype

```
int listen (int sockfd, int backlog);
```

Parameters

Parameter	Description
<code>sockfd</code>	The socket descriptor to listen on.
<code>backlog</code>	The maximum number of outstanding connections allowed on the socket.

Return Values

Value	Description
0	Success
-1	An error occurred

If the return value is -1, `errno` will be set to one of the following values.

`listen()` can fail for any of the following reasons:

<code>errno</code>	Description
<code>EADDRINUSE</code>	The address is currently used by another socket.
<code>EBADF</code>	The socket descriptor is invalid.
<code>EOPNOTSUPP</code>	The socket is not of a type that supports the operation <code>listen</code> .

accept()

The argument `sockfd()` is a socket that has been created with `socket`, bound to an address with `bind()`, and that is listening for connections after a call to `listen()`. `accept()` extracts the first connection on the queue of pending connections, creates a new socket with the properties of `sockfd`, and allocates a new socket descriptor for the socket. If no pending connections are present on the queue and the socket is not marked as non-blocking, `accept()` blocks the caller until a connection is present. If the socket is marked as non-blocking and no pending connections are present on the queue, `accept()` returns an error as described below. The accepted socket is used to send and receive data to and from the socket that it is connected to. It is not used to accept more connections. The original socket remains open for accepting further connections. `accept()` is used with connection-based socket types, currently with `SOCK_STREAM`.

Using `select()` prior to calling `accept()`

It is possible to select a listening socket for the purpose of an `accept()` by selecting it for a `read()`. However, this will only indicate when a `connect()` indication is pending; it is still necessary to call `accept()`.

Prototype

```
int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```

Parameters

Parameter	Description
<code>sockfd</code>	The socket descriptor that was created with <code>socket</code> and bound to with <code>bind()</code> and is listening for connections with <code>listen()</code> .
<code>addressPtr</code>	The structure to write the incoming address into. If <code>addressPtr</code> and <code>addressLengthPtr</code> are equal to <code>NULL</code> , then no information about the remote address of the accepted socket is returned.
<code>addressLengthPtr</code>	Initially, it contains the amount of space pointed to by <code>addressPtr</code> . On return, it contains the length in bytes of the address returned.

Return Values

Value	Description
> 0	New socket descriptor.
-1	Error

If `accept()` fails, `errno` will be set to one of the following values:

<code>errno</code>	Description
<code>EBADF</code>	The socket descriptor is invalid.
<code>EINVAL</code>	<code>addressPtr</code> was a null pointer.

errno	Description
EINVAL	addressLengthPtr was a null pointer.
EINVAL	The value of addressLengthPtr was too small.
EPERM	Cannot accept call without listening to call first.
EOPNOTSUPP	The referenced socket is not of type SOCK_STREAM.
EWOULDBLOCK	The socket is marked as non-blocking and no connections are present to be accepted.

connect()

The parameter `sockfd()` is a socket. If it is of type `SOCK_DGRAM`, `connect()` specifies the peer with which the socket is to be associated; this address is the address to which datagrams are to be sent if a receiver is not explicitly designated; it is the only address from which datagrams are to be received. If the socket `sockfd()` is of type `SOCK_STREAM`, `connect()` attempts to make a connection to another socket (either local or remote). The other socket is specified by `addressPtr`. `addressPtr` is a pointer to the IP address and port number of the remote or local socket. If `sockfd` is not bound, then it will be bound to an address selected by the underlying transport provider. Generally, stream sockets may successfully connect only once; datagram sockets may use `connect()` multiple times to change their association. Datagram sockets may dissolve the association by connecting to a null address. Note that a non-blocking `connect()` is allowed. In this case, if the connection has not been established, and not failed, `connect()` returns `SOCKET_ERROR`, and `socket_errno` returns `EINPROGRESS` error code indicating that the connection is pending. `connect()` should never be called more than once. Additional calls to `connect()` fails with `EISCONN` error code returned by `socket_errno`.

Non-blocking connect() and select()

After issuing one non-blocking `connect()`, the user can call `select()` with the write mask set for that socket descriptor to check for connection completion. When `select()` returns with the write mask set for that socket, the user can call `getsockopt()` with the `SO_ERROR` option name. If the retrieved pending socket error is `ENOERROR`, then the connection has been established. Otherwise an error occurred in the connection, as indicated by the retrieved pending socket error.

Non-blocking connect() and polling

Alternatively, after the user issues a non-blocking `connect()` call that returns `SOCKET_ERROR`, the user can poll for completion, by calling `socket_errno()` until `socket_errno()` no longer returns `EINPROGRESS`.

If `connect()` fails, the socket is no longer usable, and must be closed. `connect()` cannot be called again on the socket.

Prototype

```
int connect (int sockfd, const struct sockaddr *servaddr, socklen_t
addrlen);
```

Parameters

Parameter	Description
<code>sockfd</code>	The socket descriptor to assign a name (port number) to.
<code>addressPtr</code>	The pointer to the structure containing the address to connect to for TCP.

Parameter	Description
	For UDP it is the default address to send to and the only address to receive from.
addressLength	The length of the address structure.

Return Values The function returns the following values.

Value	Description
0	Success
-1	An error occurred

If the return value is -1, errno will be set to one of the following values.

connect () can fail for any of the following reasons:

errno	Description
EADDRINUSE	The socket address is already in use.
EADDRNOTAVAIL	The specified address is not available on the remote / local machine.
EPFNOSUPPORT	Addresses in the specified address family cannot be used with this socket
EINPROGRESS	The socket is non-blocking and the current connection attempt has not yet been completed.
EISCONN	connect () has already been called on the socket. Only one connect () call is allowed on a socket.
EBADF	sockfd is not a valid descriptor.
ECONNREFUSED	The attempt to connect was refused by the remote machine. For example, this can happen if no program is listening on the specified destination port. The calling program should close the socket descriptor, and issue another socket call to obtain a new descriptor before attempting another connect () call.
EPERM	Cannot call connect () after listen () call.
EINVAL	One of the parameters is invalid
EHOSTUNREACH	No route to the host we want to connect to. The calling program should close the socket descriptor, and should issue another socket call to obtain a new descriptor before attempting another connect () call.
ETIMEDOUT	Connection establishment timed out, without establishing a connection. The calling program should close the socket descriptor, and issue another socket call to obtain a new descriptor before attempting another connect () call.

getpeername()

This function returns the IP address / Port number of the remote system to which the socket is connected.

Prototype `int getpeername (int sockfd, struct sockaddr *localaddr, socklen_t *addrlen);`

Parameters

Parameter	Description
sockfd	The socket descriptor that we wish to obtain information about.
fromAddressPtr	A pointer to the address structure that we wish to store this information into.
addressLengthPtr	The length of the address structure.

Return Values The function returns the following values.

Value	Description
0	Success
-1	An error occurred

If the return value is -1, `errno` will be set to one of the following values.
`getpeername()` can fail for any of the following reasons:

errno	Description
EBADF	<code>socketDescriptor</code> is not a valid descriptor.
ENOTCONN	The socket is not connected.
EINVAL	One of the passed parameters is not valid.

getsockname()

This function returns to the caller the Local IP Address/Port Number that we are using on a given socket.

Prototype

```
int getsockname (int sockfd, struct sockaddr *peeraddr, socklen_t *addrlen);
```

Return Values

The function returns the following values.

Value	Description
0	Success
-1	An error occurred

If the return value is -1, errno will be set to one of the following values.

getsockname() can fail for any of the following reasons:

errno	Description
EBADF	sockfd is not a valid descriptor.
EINVAL	One of the passed parameters is not valid.

Parameters

Parameter	Description
sockfd	The socket descriptor that we wish to inquire about.
myAddressPtr	The pointer to the address structure where the address information will be stored.
addressLengthPtr	The length of the address structure.

setsockopt()

`setsockopt()` is used to manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost "socket" level. When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the "socket" level, `protocolLevel` is specified as `SOL_SOCKET`. To manipulate options at any other level, `protocolLevel` is the protocol number of the protocol that controls the option. For example, to indicate that an option is to be interpreted by the TCP protocol, `protocolLevel` is set to the TCP protocol number. The parameters `optionValuePtr` and `optionlength` are used to access option values for `setsockopt()`. `optionName` and any specified options are passed un-interpreted to the appropriate protocol module for interpretation. The include file `<svc_net.h>` contains definitions for the options described below. Most socket-level options take an `int` pointer for `optionValuePtr`. For `setsockopt()`, the integer value pointed to by the `optionValuePtr` parameter should be non-zero to enable a boolean option, or zero if the option is to be disabled. `SO_LINGER` uses a struct `linger` parameter that specifies the desired state of the option and the linger interval (see below). struct `linger` is defined in `<svc_net.h>`. struct `linger` contains the following members:

Parameter	Description
<code>l_onoff</code>	On = 1 / Off = 0
<code>l_linger</code>	Linger time, in seconds

SOL_SOCKET level

The following options are recognized at the socket level:

protocolLevel options	Description
<code>SO_BROADCAST</code>	Requests permission to send broadcast datagrams on the socket. With protocols that support out-of-band data, the <code>SO_OOBINLINE</code> option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with <code>recv</code> call without the <code>MSG_OOB</code> flag. <code>SO_SNDBUF</code> and <code>SO_RCVBUF</code> are options that adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections or may be decreased to limit the possible backlog of incoming data. The Internet protocols place an absolute limit of 64 KB on these values for UDP and TCP sockets (in the default mode of operation).

protocolLevel options	Description
SO_DONTROUTE	Enable/disable routing bypass for outgoing messages. Default 0.
SO_KEEPALIVE	Enable/disable keep connections alive. Default 0.
SO_LINGER	Controls the action taken when unsent messages are queued on a socket and a close on the socket is performed. If the socket promises reliable delivery of data and <code>SO_LINGER</code> is set, the system will block the process on the close of the socket attempt until it is able to transmit the data or decides it is unable to deliver the information. A timeout period, termed the linger interval, is specified in the <code>setsockopt</code> call when <code>SO_LINGER</code> is requested. If <code>SO_LINGER</code> is disabled and a close on the socket is issued, the system will process the close of the socket in a manner that allows the process to continue as quickly as possible. Default is on with linger time of 60 seconds.
SO_OOBINLINE	Enable/disable reception of out-of-band data in band. Default 0.
SO_RCVBUF	Set buffer size for input. Default 8192 bytes.
SO_RCVCOPYTCP	TCP socket: fraction use of a receive buffer below which we try and append to a previous receive buffer in the socket receive queue. UDP socket: fraction use of a receive buffer below which we try and copy to a new receive buffer, if there is already at least a buffer in the receive queue. This is to avoid keeping large pre-allocated receive buffers, which the user has not received yet, in the socket receive queue. Default value is 4 (25%).
SO_RCVLOWAT	The low water mark for receiving data.
SO_REUSEADDR	Indicates that the rules used in validating addresses supplied in a <code>bind()</code> call should allow reuse of local addresses. <code>SO_KEEPALIVE</code> enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken. <code>SO_DONTROUTE</code> indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.
SO_RCV_DGRAMS	The number of non-TCP datagrams that can be queued for receive on a socket. Default 8 datagrams.
SO_RCVTIMEO	Receive timeout. Retrieved using a <code>timeval</code> struct. Default is 0. Resolution of timeout is 5 ms.

protocolLevel options	Description
SO_SNDAPPEND	TCP socket only. Threshold in bytes of send buffer below, which we try and append, to previous send buffer in the TCP send queue. Only used with send. This is to try and regroup lots of partially empty small buffers in the TCP send queue waiting to be ACKED by the peer; otherwise we could run out of memory, since the remote TCP will delay sending ACKs.
SO_SNDBUF	Set buffer size for output. Default 8192 bytes.
SO_SND_DGRAMS	The number of non-TCP datagrams that can be queued for send on a socket. Default 8 datagrams.
SO_SNDLOWAT	The low water mark for sending data.
SO_SNDTIMEO	Send timeout. Retrieved using a timeval struct. Default is 0. Resolution of timeout is 1 ms.

IP_PROTOIP level

The following options are recognized at the IP level:

protocolLevel options	Description
IPO_HDRINCL	This is a toggle option used on Raw Sockets only. If the value is non-zero, it instructs the stack that the user is including the IP header when sending data. Default 0.
IPO_TOS	IP type of service. Default 0.
IPO_TTL	IP Time To Live in seconds. Default 64.
IPO_SRCADDR	Our IP source address. Default: first multi-home IP address on the outgoing interface.
IPO_MULTICAST_TTL	Change the default IP TTL for outgoing multicast datagrams.
IPO_MULTICAST_IF	Specify a configured IP address that will uniquely identify the outgoing interface for multicast datagrams sent on this socket. A zero IP address parameter indicates that we want to reset a previously set outgoing interface for multicast packets sent on that socket.
IPO_ADD_MEMBERSHIP	Add group multicast IP address to given interface (see struct ip_mreq data type below).
IPO_DROP_MEMBERSHIP	Delete group multicast IP address from given interface (see struct ip_mreq data type below).

`ip_mreq` structure definition:

```
struct ip_mreq
{
    struct in_addr    imr_multiaddr;
    struct in_addr    imr_interface
};
```

`ip_mreq` structure Members

Member	Description
<code>imr_multiaddr</code>	IP host group address that the user wants to join/leave
<code>imr_interface</code>	IP address of the local interface that the host group address is to be joined on, or is to leave from.

IP_PROTOTCP level

The following options are recognized at the TCP level. Options marked with an asterisk can only be changed prior to establishing a TCP connection.

protocolLevel options	Description
<code>TCP_KEEPAIVE</code>	Sets the idle time in seconds for a TCP connection, before it starts sending keep alive probes. It cannot be set below the default value. Note that keep alive probes will be sent only if the <code>SO_KEEPAIVE</code> socket option is enabled. Default 7,200 seconds.
<code>TCP_MAXRT</code>	Sets the amount of time in seconds before the connection is broken, once TCP starts retransmitting, or probing a zero window, when the peer does not respond. A <code>TCP_MAXRT</code> value of 0 means to use the system default, and -1 means to retransmit forever. If a positive value is specified, it may be rounded-up to the connection next retransmission time. Note that unless the <code>TCP_MAXRT</code> value is -1 (wait forever), the connection can also be broken if the number of maximum retransmissions has been reached (<code>TCP_MAX_REXMIT</code>). See <code>TCP_MAX_REXMIT</code> below. Default 0 (Which means use system default of <code>TCP_MAX_REXMIT</code> times network computed round trip time for an established connection; for a non established connection, since there is no computed round trip time yet, the connection can be broken when either 75 seconds, or when <code>TCP_MAX_REXMIT</code> times default network round trip time have elapsed, whichever occurs first).

protocolLevel options	Description
TCP_MAXSEG	Sets the maximum TCP segment size sent on the network. Note that the <code>TCP_MAXSEG</code> value is the maximum amount of data (including TCP options, but not the TCP header) that can be sent per segment to the peer., i.e. the amount of user data sent per segment is the value given by the <code>TCP_MAXSEG</code> option minus any enabled TCP option (for example 12 bytes for a TCP time stamp option). The <code>TCP_MAXSEG</code> value can be decreased or increased prior to a connection establishment, but it is not recommended to set it to a value higher than the IP MTU minus 40 bytes (for example 1460 bytes on Ethernet), since this would cause fragmentation of TCP segments. Note: setting the <code>TCP_MAXSEG</code> option will inhibit the automatic computation of that value by the system based on the IP MTU (which avoids fragmentation), and will also inhibit Path Mtu Discovery. After the connection has started, this value cannot be changed. Note also that the <code>TCP_MAXSEG</code> value cannot be set below 64 bytes. Default value is IP MTU minus 40 bytes.
TCP_NODELAY	Set this option value to a non-zero value, to disable the Nagle algorithm that buffers the sent data inside the TCP. Useful to allow client's TCP to send small packets as soon as possible (like mouse clicks). Default 0.
TCP_NOPUSH	Set this option value to a non-zero value, to force TCP to delay sending any TCP data until a full sized segment is buffered in the TCP buffers. Useful for applications that send continuous big chunks of data like FTP, and know that more data is coming. (Normally the TCP code sends a non full-sized segment, only if it empties the TCP buffer). Default 0.
TCP_STDURG	Set this option value to a zero value, if the peer is a Berkeley system since Berkeley systems set the urgent data pointer to point to last byte of urgent data+1. Default 1 (urgent pointer points to last byte of urgent data as specified in RFC1122).
TCP_PEND_ACCEPT_RECV_WND	Specify the size (in bytes) of the listening socket's receive window. This size will override the default size or the size specified by <code>setsockopt()</code> with the <code>SO_RCVBUF</code> flag. Once <code>accept()</code> is called on the listening socket, the window size will return to the size specified by <code>SO_RCVBUF</code> (or the default). Note: This size may not be larger than the default window size to avoid shrinking of the receive window.
TCP_SEL_ACK	Set this option value to a non-zero value, to enable sending the TCP selective Acknowledgment option. Default 1.
TCP_WND_SCALE	Set this option value to a non-zero value, to enable sending the TCP window scale option. Default 1.
TCP_TS	Set this option value to a non-zero value, to enable sending the Time stamp option. Default 1.

protocolLevel options	Description
TCP_SLOW_START	Set this option value to zero, to disable the TCP slow start algorithm. Default is 1.
TCP_DELAY_ACK	Sets the TCP delay ACK time in milliseconds. Default is 200 milliseconds.
TCP_MAX_REXMIT	Sets the maximum number of retransmissions without any response from the remote, before TCP gives up and aborts the connection. See also TCP_MAXRT above. Default is 12.
TCP_KEEPA_LIVE_CNT	Sets the maximum numbers of keep alive probes without any response from the remote, before TCP gives up and aborts the connection. See also TCP_KEEPA_LIVE above. Default 8.
TCP_FINWT2TIME	Sets the maximum amount of time TCP will wait for the remote side to close, after it initiated a close. Default 600 seconds.
TCP_2MSLTIME	Sets the maximum amount of time TCP will wait in the TIME WAIT state, once it has initiated a close of the connection. Default 60 seconds.
TCP_RTO_DEF	Sets the TCP default retransmission timeout value in milliseconds, used when no network round trip time has been computed yet. Default 3,000 milliseconds.
TCP_RTO_MIN	Sets the minimum retransmission timeout in milliseconds. The network computed retransmission timeout is bound by TCP_RTO_MIN and TCP_RTO_MAX. Default 1000 milliseconds.
TCP_RTO_MAX	Sets the maximum retransmission timeout in milliseconds. The network computed retransmission timeout is bound by TCP_RTO_MIN and RTO_MAX. Default 60,000 milliseconds.
TCP_PROBE_MIN	Sets the minimum window probe timeout interval in milliseconds. The network computed window probe timeout is bound by TCP_PROBE_MIN and TCP_PROBE_MAX. Default 500 milliseconds.
TCP_PROBE_MAX	Sets the maximum window probe timeout interval in milliseconds. The network computed window probe timeout is bound by TCP_PROBE_MIN and TCP_PROBE_MAX. Default 60,000 milliseconds.
TCP_KEEPA_LIVE_INTV	Sets the interval between Keep Alive probes in seconds. See TCP_KEEPA_LIVE_CNT. This value cannot be changed after a connection is established, and cannot be bigger than 120 seconds. Default 75 seconds.
TCP_PUSH_ALL	Set PUSH on every packet even when not emptying the send queue! Use sparingly. TCP_NODELAY option is preferable to this option. Also preserve packet boundaries when sending data.

protocolLevel options	Description
-----------------------	-------------

TCP_PACKET	Preserve user packet boundary when sending data: Set PUSH on every packet even when not emptying the send queue. Preserve packet boundaries even on retransmissions.
------------	--

Prototype

```
int setsockopt (int sockfd, int level, int optname, const void *optval,
socklen_t optlen);
```

Parameters

Parameter	Description
sockfd	The socket descriptor to set the options on.
protocolLevel	The protocol to set the option on. See below.
optionName	The name of the option to set. See below and above.
optionValuePtr	The pointer to a user variable from which the option value is set. User variable is of data type described below.
optionLength	The size of the user variable. It is the size of the option data type described below.

Values for protocolLevel:

protocolLevel	Description
SOL_SOCKET	Socket level protocol.
IPPROTOIP	IP level protocol.
IPPROTOTCP	TCP level protocol

Values for optionName:

Protocol Level	Option Name	Option Data Type	Option Value
SOL_SOCKET	SO_DONTROUTE	int	0 or 1
	SO_KEEPAIVE	int	0 or 1
	SO_LINGER	struct linger	0 or 1
	SO_OOINLINE	int	
	SO_RCVBUF	unsigned long	
	SO_RCVLOWAT	unsigned long	
	SO_REUSEADDR	int	0 or 1
	SO_RCVCOPY	unsigned int	
	SO_RCV_DGRAMS	unsigned long	
	SO_RCVTIMEO	struct timeval	
	SO_SNDAPPEND	unsigned int	
	SO_SNDBUF	unsigned long	

Protocol Level	Option Name	Option Data Type	Option Value
	SO_SND_DGRAMS	unsigned long	
	SO_SNDLOWAT	unsigned long	
	SO_SNDTIMEO	struct timeval	
IP_PROTOIP	IPO_TOS	unsigned char	
	IPO_TTL	unsigned char	
	IPO_SRCADDR	ttUserIpAddress	
	IPO_MULTICAST_TTL	unsigned char	
	IPO_MULTICAST_IF	struct in_addr	
	IPO_ADD_MEMBERSHIP	struct ip_mreq	
	IPO_DROP_MEMBERSHIP	struct ip_mreq	
IP_PROTOTCP	TCP_KEEPA_LIVE	int	
	TCP_MAXRT	int	
	TCP_MAXSEG	int	
	TCP_NODELAY	int	0 or 1
	TCP_NOPUSH	int	0 or 1
	TCP_STDURG	int	0 or 1
	TCP_PACKET	int	0 or 1
	TCP_2MSLTIME	int	
	TCP_DELAY_ACK	int	
	TCP_FINWT2TIME	int	
	TCP_KEEPA_LIVE_CNT	int	
	TCP_KEEPA_LIVE_INTV	int	
	TCP_MAX_REXMIT	int	
	TCP_PROBE_MAX	unsigned long	
	TCP_PROBE_MIN	unsigned long	
	TCP_RTO_DEF	unsigned long	
	TCP_RTO_MAX	unsigned long	
	TCP_RTO_MIN	unsigned long	
	TCP_SEL_ACK	int	0 or 1
	TCP_SLOW_START	int	0 or 1
	TCP_TS	int	0 or 1

Protocol Level	Option Name	Option Data Type	Option Value
	TCP_WND_SCALE	int	0 or 1

Return Values

The function returns the following values:

Value	Description
0	Successful set of option
-1	An error occurred

If the return value is -1, errno will be set to one of the following values.

setsockopt () can fail for any of the following reasons:

errno	Description
EBADF	The socket descriptor is invalid
EINVAL	One of the parameters is invalid
ENOPROTOOPT	The option is unknown at the level indicated.
EPERM	Option cannot be set after the connection has been established.
EPERM	IPO_HDRINCL option cannot be set on non-raw sockets.
ENETDOWN	Specified interface not yet configured.
EADDRINUSE	Multicast host group already added to the interface.
ENOBUF	Not enough memory to add new multicast entry.
ENOENT	Attempted to delete a non-existent multicast entry on the specified interface.

getsockopt()

`getsockopt()` is used to retrieve options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost "socket" level. When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the "socket" level, `protocolLevel` is specified as `SOL_SOCKET`. To manipulate options at any other level, `protocolLevel` is the protocol number of the protocol that controls the option. For example, to indicate that an option is to be interpreted by the TCP protocol, `protocolLevel` is set to the TCP protocol number. For `getsockopt()`, the parameters `optionValuePtr` and `optionLengthPtr` identify a buffer in which the value(s) for the requested option(s) are to be returned. For `getsockopt()`, `optionLengthPtr` is a value-result parameter, initially containing the size of the buffer pointed to by `optionValuePtr`, and modified on return to indicate the actual size of the value returned. `optionName` and any specified options are passed un-interpreted to the appropriate protocol module for interpretation. The include file `<svc_net.h>` contains definitions for the options described below. Options vary in format and name. Most socket-level options take an int for `optionValuePtr`. `SO_LINGER` uses a struct `linger` parameter that specifies the desired state of the option and the linger interval (see below). struct `linger` is defined in `<svc_net.h>`. struct `linger` contains the following members:

Parameter	Description
<code>l_onoff</code>	On = 1 / Off = 0
<code>l_linger</code>	Linger time, in seconds

SOL_SOCKET level

The following options are recognized at the socket level:

protocolLevel options	Description
<code>SO_ACCEPTCONN</code>	Enable/disable listening for connections. <code>listen()</code> turns on this option.
<code>SO_DONTROUTE</code>	Enable/disable routing bypass for outgoing messages. Default 0.
<code>SO_ERROR</code>	When an error occurs on a socket, the stack internally sets the error code on the socket. It is called the pending error for the socket. If the user had called <code>select</code> for either readability or writability, <code>select</code> returns with either or both conditions set. The user can then retrieve the pending socket error, by calling <code>getsockopt()</code> with this option name at the <code>SOL_SOCKET</code> level, and the stack will reset the internal socket error. Alternatively if the user is waiting for incoming data, <code>read</code> or other <code>recv</code> APIs can be called. If

protocolLevel options	Description
	there is no data queued to the socket, the read/recv call returns <code>SOCKET_ERROR</code> , the stack resets the internal socket error, and the pending socket error can be returned if the user calls <code>socket_errno</code> (equivalent of <code>errno</code>). Note that the <code>SO_ERROR</code> option is useful when the user uses connect in non-blocking mode, and select.
<code>SO_KEEPAVIVE</code>	Enable/disable keep connections alive. Default 0 (disable).
<code>SO_OOBINLINE</code>	Enable/disable reception of out-of-band data in band. Default is 0.
<code>SO_REUSEADDR</code>	Enable this socket option to bind the same port number to multiple sockets using different local IP addresses. Default is 0 (disable).
<code>SO_RCVLOWAT</code>	The low water mark for receiving.
<code>SO_SNDLOWAT</code>	The low water mark for sending.
<code>SO_RCVBUF</code>	The buffer size for input. Default is 8192 bytes.
<code>SO_SNDBUF</code>	The buffer size for the output. Default is 8192 bytes. If this option was set using <code>setsockopt()</code> and a connect command is issued, the buffer size will be rounded to the nearest multiple of the MSS. MSS is the minimum size.
<code>SO_RCVCOPY</code>	TCP socket: fraction use of a receive buffer below which we try and append to a previous receive buffer in the socket receive queue. UDP socket: fraction use of a receive buffer below which we try and copy to a new receive buffer, if there is already at least a buffer in the receive queue. This is to avoid keeping large pre-allocated receive buffers, which the user has not received yet, in the socket receive queue. Default value is 4 (25%).
<code>SO_SNDAPPEND</code>	TCP socket only. Threshold in bytes of 'send' buffer below, which we try and append, to previous 'send' buffer in the TCP send queue. Only used with send. This is to try to regroup lots of partially empty small buffers in the TCP send queue waiting to be ACKED by the peer; otherwise we could run out of memory, since the remote TCP will delay sending ACKs. Default value is 128 bytes.
<code>SO_SND_DGRAMS</code>	The number of non-TCP datagrams that can be queued for send on a socket. Default 8 datagrams.
<code>SO_RCV_DGRAMS</code>	The number of non-TCP datagrams that can be queued for receive on a socket. Default 8 datagrams.

protocolLevel options	Description
SO_REUSEADDR	Indicates that the rules used in validating addresses supplied in a bind call should allow reuse of local addresses. <code>SO_KEEPALIVE</code> enables the periodic transmission of messages (every 2 hours) on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken. <code>SO_DONTROUTE</code> indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.
SO_LINGER	controls the action taken when unsent messages are queued on a socket and a close on the socket is performed. If the socket promises reliable delivery of data and <code>SO_LINGER</code> is set, the system will block the process on the close of the socket attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the <code>setsockopt</code> call when <code>SO_LINGER</code> is requested). If <code>SO_LINGER</code> is disabled and a close on the socket is issued, the system will process the close of the socket in a manner that allows the process to continue as quickly as possible. The option <code>SO_BROADCAST</code> requests permission to send broadcast datagrams on the socket. With protocols that support out-of-band data, the <code>SO_OOBINLINE</code> option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with <code>recv</code> call without the <code>MSG_OOB</code> flag. <code>SO_SNDBUF</code> and <code>SO_RCVBUF</code> are options that adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections or may be decreased to limit the possible backlog of incoming data. The Internet protocols place an absolute limit of 64 KB on these values for UDP and TCP sockets (in the default mode of operation).
SO_RCVTIMEO	Receive timeout. Retrieved using a <code>timeval</code> struct. Default is 0. Resolution of timeout is 5 ms.
SO_SNDTIMEO	Send timeout. Retrieved using a <code>timeval</code> struct. Default is 0. Resolution of timeout is 1 ms.

IP_PROTOIP level

The following options are recognized at the IP level:

protocolLevel options	Description
IPO_MULTICAST_IF	Get the configured IP address that uniquely identifies the outgoing interface for multicast datagrams sent on this socket. A zero IP address parameter indicates that we want to reset a previously set outgoing interface for multicast packets sent on that socket.
IPO_MULTICAST_TTL	Get the default IP TTL for outgoing multicast datagrams.
IPO_SRCADDR	Get the IP source address for the connection.
IPO_TOS	IP type of service. Default 0.
IPO_TTL	IP Time To Live in seconds. Default 64

IP_PROTOTCP level

The following options are recognized at the TCP level. Options marked with an asterisk can only be changed prior to establishing a TCP connection.

protocolLevel options	Description
TCP_KEEPA* LIVE	Get the idle time in seconds for a TCP connection before it starts sending keep alive probes. Note that keep alive probes will be sent only if the <code>SO_KEEPA* LIVE</code> socket option is enabled. Default 7,200 seconds.
TCP_MAXRT	Get the amount of time in seconds before the connection is broken once TCP starts retransmitting, or probing a zero window when the peer does not respond. A <code>TCP_MAXRT</code> value of 0 means the system default, and -1 means retransmit forever. Note that unless the <code>TCP_MAXRT</code> value is -1 (wait forever), the connection can also be broken if the number of maximum retransmission <code>TCP_MAX_REXMIT</code> has been reached. See <code>TCP_MAX_REXMIT</code> below. Default 0. (which means use system default of <code>TCP_MAX_REXMIT</code> times network computed round trip time for an established connection. For a non established connection, since there is no computed round trip time yet, the connection can be broken when either 75 seconds or when <code>TCP_MAX_REXMIT</code> times default network round trip time have elapsed, whichever occurs first).
TCP_MAXSEG	Get the maximum TCP segment size sent on the network. Note that the <code>TCP_MAXSEG</code> value is the maximum amount of data (including TCP options, but not the TCP header) that can be sent per segment to the peer. i.e. the amount of user data sent per segment is the value given by the <code>TCP_MAXSEG</code> option minus any enabled TCP option (for example 12 bytes for a TCP time stamp option). Default is IP MTU minus 40 bytes.

protocolLevel options	Description
TCP_NODELAY	If this option value is non-zero, the Nagle algorithm that buffers the sent data inside the TCP is disabled. Useful to allow client's TCP to send small packets as soon as possible (like mouse clicks). Default 0.
TCP_NOPUSH	If this option value is non-zero, then TCP delays sending any TCP data until a full sized segment is buffered in the TCP buffers. Useful for applications that send continuous big chunks of data and know that more data will be sent such as FTP. (Normally, the TCP code sends a non full-sized segment, only if it empties the TCP buffer). Default 0.
TCP_STDURG	If this option value is zero, then the urgent data pointer points to the last byte of urgent data + 1, like in Berkeley systems. Default 1 (urgent pointer points to last byte of urgent data as specified in RFC1122).
TCP_PACKET	If this option value is non-zero, then TCP behaves like a message-oriented protocol (i.e. respects packet boundaries) at the application level in both send and receive directions of data transfer. Note that for the receive direction to respect packet boundaries, the TCP peer which is sending must also implement similar functionality in its send direction. This is useful as a reliable alternative to UDP. Note that preserving packet boundaries with TCP will not work correctly if you use out-of-band data. Default 0.
TCP_SEL_ACK	If this option value is zero, then TCP selective Acknowledgment options are disabled. Default 1.
TCPWND_SCALEI	If this option value is non-zero, then the TCP window scale option is enabled. Default 1.
TCP_TS	If this option value is non-zero, then the TCP time stamp option is enabled. Default 1.
TCP_SLOW_START	If this option value is non-zero, then the TCP slow start algorithm is enabled. Default 1.
TCPDELAY_ACK	Get the TCP delay ACK time in milliseconds. Default 200 milliseconds.
TCPMAX_REXMIT	Get the maximum number of retransmissions without any response from the remote before TCP gives up and aborts the connection. See also <code>TCP_MAXRT</code> above. Default 12.
TCP_KEEPA_LIVE_CNT	Get the maximum number of keep alive probes without any response from the remote before TCP gives up and aborts the connection. See also <code>TCP_KEEPA_LIVE</code> above. Default 8.
TCPFINWT2TIME	Get the maximum amount of time TCP will wait for the remote side to close after it initiated a close. Default 600 seconds.

protocolLevel options	Description
TCP2MSLTIME	Get the maximum amount of time TCP will wait in the TIME WAIT state once it has initiated a close of the connection. Default 60 seconds.
TCP_PEND_ACCEPT_RECV_WND	Specify the size (in bytes) of the listening socket's receive window. This size will override the default size or the size specified by setsockopt() with the SO_RCVBUF flag. Once accept() is called on the listening socket, the window size will return to the size specified by SO_RCVBUF (or the default). Note: This size may not be larger than the default window size to avoid shrinking of the receive window.
TCP_RTO_DEF	Get the TCP default retransmission timeout value in milliseconds. Used when no network round trip time has been computed yet. Default 3,000 milliseconds.
TCP_RTO_MIN	Get the minimum retransmission timeout in milliseconds. The network computed retransmission timeout is bound by TCP_RTO_MIN and TCP_RTO_MAX. Default 100 milliseconds.
TCPRTO_MAX	Get the maximum retransmission timeout in milliseconds. The network computed retransmission timeout is bound by TCPRTO_MIN and RTO_MAX. Default 64,000 milliseconds.
TCPPROBE_MIN	Get the minimum window probe timeout interval in milliseconds. The network computed window probe timeout is bound by TCP_PROBE_MIN and TCP_PROBE_MAX. Default 500 milliseconds.
TCP_PROBE_MAX	Get the maximum window probe timeout interval in milliseconds. The network computed window probe timeout is bound by TCP_PROBE_MIN and TCP_PROBE_MAX. Default 60,000 milliseconds.
TCP_KEEPA_LIVE_INTV	Get the interval between Keep Alive probes in seconds. See TCP_KEEPA_LIVE_CNT. Default 75 seconds.

Prototype

```
int getsockopt (int sockfd, int level, int optname, void *optval,
socklen_t *optlen);
```

Parameters

Parameter	Description
sockfd	The socket descriptor to get the option from.
protocolLevel	The protocol to get the option from. See below.
optionName	The option to get. See above and below.
optionValuePtr	The pointer to a user variable into which the option value is returned. User variable is of data type described below.
optionLengthPtr	Pointer to the size of the user variable, which is the size of the option data type, described below. It is a value-result parameter, and the user should set the size prior to the call.

Values for `protocolLevel`:

Protocol Level	Description
<code>SOL_SOCKET</code>	Socket level protocol.
<code>IPPROTOIP</code>	IP level protocol.
<code>IPPROTOTCP</code>	TCP level protocol

Values for `optionName`:

Protocol Level	Option Name	Option Data Type	Option Value
<code>SOL_SOCKET</code>	<code>SO_ACCEPTCON</code>	int	0 or 1
	<code>SO_DONTROUTE</code>	int	0 or 1
	<code>SO_ERROR</code>	int	
	<code>SO_KEEPALIVE</code>	int	0 or 1
	<code>SO_LINGER</code>	struct linger	
	<code>SO_OOBINLINE</code>	int	0 or 1
	<code>SO_RCVBUF</code>	unsigned long	
	<code>SO_RCVLOWAT</code>	unsigned long	
	<code>SO_REUSEADDR</code>	int	0 or 1
	<code>SO_SNDBUF</code>	unsigned long	
	<code>SO_SNDLOWAT</code>	unsigned long	
	<code>SO_RCVCOPY</code>	unsigned int	
	<code>SO_SND_DGRAMS</code>	unsigned long	
	<code>SO_SNDAPPEND</code>	unsigned int	
<code>IPPROTOIP</code>	<code>IPO_MULTICAST_TTL</code>	unsigned char	
	<code>IPO_MULTICAST_IF</code>	struct in_addr	
	<code>IPO_TOS</code>	unsigned char	
	<code>IPO_TTL</code>	unsigned char	
	<code>IPO_SRCADDR</code>	ttUserIpAddress	
<code>IPPROTOTCP</code>	<code>TCP_KEEPALIVE</code>	int	
	<code>TCP_MAXRT</code>	int	
	<code>TCP_MAXSEG</code>	int	
	<code>TCP_NODELAY</code>	int	0 or 1
	<code>TCP_NOPUSH</code>	int	0 or 1
	<code>TCP_STDURG</code>	int	0 or 1

Protocol Level	Option Name	Option Data Type	Option Value
	TCP_2MSLTIME	int	
	TCP_DELAY_ACK	int	
	TCP_FINWT2TIME	int	
	TCP_KEEPALIVE_CN	int	
	TCP_KEEPALIVE_IN	int	
	TCP_MAX_REXMIT	int	
	TCP_PACKET	int	0 or 1
	TCP_PROBE_MAX	unsigned long	
	TCP_PROBE_MIN	unsigned long	
	TCP_RTO_DEF	unsigned long	
	TCP_RTO_MAX	unsigned long	
	TCP_RTO_MIN	unsigned long	
	TCP_SEL_ACK	int	0 or 1
	TCP_SLOW_START	int	0 or 1
	TCP_TS	int	0 or 1
	TCP_WND_SCALE	int	0 or 1

Return Values The function returns the following values:

Value	Description
0	Successful set of option
-1	An error occurred

If the return value is -1, errno will be set to one of the following values:

getsockopt () can fail for any of the following reasons:

errno	Description
EBADF	The socket descriptor is invalid
EINVAL	One of the parameters is invalid
ENOPROTOPT	The option is unknown at the level indicated

recv()

`recv()` is used to receive messages from another socket. `recv()` may be used only on a connected socket (see [connect\(\)](#), [accept\(\)](#)). `sockfd` is a socket created with `socket()` or `accept()`. The length of the message is returned. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see [socket\(\)](#)). The length of the message returned could also be smaller than `bufferLength` (this is not an error). If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is non-blocking, or the `MSG_DONTWAIT` flag is set in the `flags` parameter, in which case -1 is returned with socket error being set to `EWOULDBLOCK`.

Out-of-band data not in the stream (urgent data when the `SO_OOBINLINE` option is not set (default)) (TCP protocol only).

A single out-of-band data byte is provided with the TCP protocol when the `SO_OOBINLINE` option is not set. If an out-of-band data byte is present, `recv()` with the `MSG_OOB` flag not set will not read past the position of the out-of-band data byte in a single `recv()` request. That is, if there are 10 bytes from the current read position until the out-of-band byte, and if we execute a `recv()` specifying a `bufferLength` of 20 bytes, and a flag value of 0, `recv()` will only return 10 bytes. This forced stopping is to allow us to execute the `SIOCATMARK` socketioctl to determine when we are at the out-of-band byte mark. Alternatively, `GetOobDataOffset` can be used instead of `socketioctl` to determine the offset of the out-of-band data byte.

Out-of-band data (when the `SO_OOBINLINE` option is set (see [setsockopt\(\)](#)).

(TCP protocol only) If the `SO_OOBINLINE` option is enabled, the out-of-band data is left in the normal data stream and is read without specifying the `MSG_OOB`. More than one out-of-band data bytes can be in the stream at any given time. The out-of-band byte mark corresponds to the final byte of out-of-band data that was received. In this case, the `MSG_OOB` flag cannot be used with `recv()`. The out-of-band data will be read in line with the other data. Again, `recv()` will not read past the position of the out-of-band mark in a single `recv()` request. Again, `socketioctl` with the `SIOCATMARK` can be used to determine where the last received out-of-band byte is in the stream.

`select()` may be used to determine when more data arrives, or/and when out-of-band data arrives.

Prototype

```
int recv (int sockfd, void *buff, size_t nbytes, int flags);
```

Parameters

Parameter	Description
sockfd	The socket descriptor from which to receive data.
bufferPtr	The buffer into which the received data is placed.
bufferLength	The length of the buffer area that bufferPtr points to.
flags	See below

The flags parameter is formed by ORing one or more of the following:

Parameter	Description
MSG_DONTWAIT	Do not wait for data, but rather return immediately.
MSG_OOB	Read any "out-of-band" data present on the socket rather than the regular "in-band" data.
MSG_PEEK	"Peek" at the data present on the socket; the data is returned, but not consumed, so that a subsequent receive operation will see the same data.
0	Read data from the socket's read buffer and discard data on reading.

Return Values

The function returns the following values.

Value	Description
> 0	Number of bytes actually received from the socket
0	EOF or remote host has closed the connection.
-1	An error occurred

If the return value is -1, errno will be set to one of the following values.

recv() can fail for any of the following reasons:

errno	Description
EBADF	The socket descriptor is invalid.
EMSGSIZE	The socket requires that message be received atomically, and bufferLength was too small.
EWOULDBLOCK	The socket is marked as non-blocking or the MSG_DONTWAIT flag is used and no data is available to be read, or the MSG_OOB flag is set and the out of band data has not arrived yet from the peer.
EINVAL	One of the parameters is invalid, or the MSG_OOB flag is set and, either the SO_OOBINLINE option is set, or there is no out of band data to read or coming from the peer.
ENOTCONN	Socket is not connected.
EHOSTUNREACH	No route to the connected host.

send()

`send()` is used to transmit a message to another transport end-point. `send()` may be used only when the socket is in a connected state. `sockfd()` is a socket created with `socket()`.

If the message is too long to pass automatically through the underlying protocol (non-TCP protocol), then the error `EMSGSIZE` is returned and the message is not transmitted.

A return value of -1 indicates locally detected errors only. A positive return value does not implicitly mean the message was delivered, but rather that it was sent.

Blocking socket `send()`: if the socket does not have enough buffer space available to hold the message being sent, `send` blocks.

Non blocking stream (TCP) socket `send()`: if the socket does not have enough buffer space available to hold the message being sent, the `send()` call does not block. It can send as much data from the message as can fit in the TCP buffer and returns the length of the data sent. If none of the message data fits, then -1 is returned with socket error being set to `EWOULDBLOCK`.

Non blocking datagram socket `send`: if the socket does not have enough buffer space available to hold the message being sent, no data is being sent and -1 is returned with socket error being set to `EWOULDBLOCK`.

The `select()` call may be used to determine when it is possible to send more data.

Sending Out-of-Band Data

For example, if you have remote login application, and you want to interrupt with a ^C keystroke, at the socket level you want to be able to send the ^C flagged as special data (also called out-of-band data). You also want the TCP protocol to let the peer (or remote) TCP know as soon as possible that a special character is coming, and you want the peer (or remote) TCP to notify the peer (or remote) application as soon as possible.

At the TCP level, this mechanism is called TCP urgent data. At the socket level, the mechanism is called out-of-band data. Out-of-band data generated by the socket layer, is implemented at the TCP layer with the urgent data mechanism. The user application can send one or several out-of-band data bytes. With TCP you cannot send the out-of-band data ahead of the data that has already been buffered in the TCP send buffer, but you can let the other side know (with the urgent flag, i.e. the term urgent data) that out-of-band data is coming, and you can let the peer TCP know the offset of the current data to the last byte of out-of-band data.

So with TCP, the out-of-band data byte(s) are not sent ahead of the data stream, but the TCP protocol can notify the remote TCP ahead of time that some out-of-band data byte(s) exist. What TCP does, is mark the byte stream where urgent data ends, and set the Urgent flag bit in the TCP header flag field, as long as it is sending data before, or up to, the last byte of out-of-band data.

In your application, you can send out-of-band data, by calling the `send()` function with the `MSG_OOB` flag. All the bytes of data sent that way (using `send()` with the `MSG_OOB` flag) are out-of-band data bytes. Note that if you call `send()` several times with out-of-band data, TCP will always keep track of where the last out-of-band byte of data is in the byte data stream, and flag this byte as the last byte of urgent data. To receive out-of-band data, please see the `recv()` section of this manual.

Prototype

```
int send (int sockfd, const void *buff, size_t nbytes, int flags);
```

Parameters

Parameter	Description
<code>sockfd</code>	The socket descriptor to use to send data
<code>bufferPtr</code>	The buffer to send
<code>bufferLength</code>	The length of the buffer to send
<code>flags</code>	See below

The flags parameter is formed by ORing one or more of the following:

Parameter	Description
<code>MSG_DONTWAIT</code>	Do not wait for data send to complete, but rather return immediately.
<code>MSG_OOB</code>	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only <code>SOCK_STREAM</code> sockets created in the <code>AF_INET</code> address family support out-of-band data.
<code>MSG_DONTROUTE</code>	The <code>SO_DONTROUTE</code> option is turned on for the duration of the operation. Only diagnostic or routing programs use it.
0	Writes data as normal data

Return Values

The function returns the following values:

Value	Description
<code>>= 0</code>	Number of bytes actually sent on the socket
<code>-1</code>	An error occurred

If the return value is -1, `errno` will be set to one of the following values.

`send()` can fail for any of the following reasons:

errno	Description
EBADF	The socket descriptor is invalid.
EINVAL	One of the parameters is invalid. the <code>bufferPtr</code> is NULL, the <code>bufferLength</code> is less than or equal to 0 or an unsupported flag is set.
ENOBUFS	There was insufficient user memory available to complete the operation.
EHOSTUNREACH	Non-TCP socket only. No route to destination host.
EMSGSIZE	The socket requires that message to be sent automatically, and the message was too long.
EWouldBlock	The socket is marked as non-blocking and the send operation would block.
ENOTCONN	Socket is not connected.
ESHUTDOWN	User has issued a write shutdown or a <code>socketclose</code> call (TCP socket only).

recvfrom()

Use `recvfrom()` to receive messages from another socket. `recvfrom()` may be used to receive data on a socket whether it is in a connected state or not but not on a TCP socket. `sockfd()` is a socket created with `socket()`. If `fromPtr` is not a NULL pointer, the source address of the message is filled in.

`fromLengthPtr` is a value-result parameter, initialized to the size of the buffer associated with `fromPtr`, and modified on return to indicate the actual size of the address stored there. The length of the message is returned. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see [socket\(\)](#)). If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is non-blocking, or the `MSG_DONTWAIT` flag is set in the flags parameter, in which case -1 is returned with socket error being set to `EWOULDBLOCK`.

`select()` may be used to determine when more data arrives, or/and when out-of-band data arrives.

Prototype

```
int recvfrom (int sockfd, void * buff, size_t nbytes, int flags, struct
sockaddr *from, socklen_t *addrlen);
```

Parameters

Parameter	Description
<code>sockfd</code>	The socket descriptor to receive data from.
<code>bufferPtr</code>	The buffer to put the received data
<code>bufferLength</code>	The length of the buffer area that <code>bufferPtr</code> points to
<code>flags</code>	See Below.
<code>fromPtr</code>	The socket from which the data is (or to be) received.
<code>fromLengthPtr</code>	The length of the data area the <code>fromPtr</code> points to then upon return the actual length of the from data

The flags parameter is formed by ORing one or more of the following:

Value	Description
<code>MSG_DONTWAIT</code>	Do not wait for data, but rather return immediately
<code>MSG_PEEK</code>	"Peek" at the data present on the socket; the data is returned, but not consumed, so that a subsequent receive operation will see the same data.
0	Reads data from the socket's read buffer and discards data on reading.

Return Values The function returns the following values.

Value	Description
> 0	Number of bytes actually received from the socket.
0	EOF
-1	An error occurred

If the return value is -1, `errno` will be set to one of the following values.

`recvfrom()` can fail for any of the following reasons:

errno	Description
EBADF	The socket descriptor is invalid.
EINVAL	One of the parameters is invalid.
EMSGSIZE	The socket requires that message be received automatically, and <code>bufferLength</code> was too small.
EPROTOTYPE	TCP protocol requires usage of <code>recv()</code> , not <code>recvfrom()</code> .
EWOULDBLOCK	The socket is marked as non-blocking and no data is available

sendto()

`sendto()` is used to transmit a message to another transport end-point. `sendto()` may be used at any time (either in a connected or unconnected state), but not for a TCP socket. `sockfd()` is a socket created with `socket()`. The address of the target is given by `toPtr` with `toLength` specifying its size.

If the message is too long to pass automatically through the underlying protocol, then -1 is returned with the socket error being set to `EMSGSIZE`, and the message is not transmitted.

A return value of -1 indicates locally detected errors only. A positive return value does not implicitly mean the message was delivered, but rather that it was sent.

If the socket does not have enough buffer space available to hold the message being sent, and is in blocking mode, `sendto()` blocks. If it is in non-blocking mode or the `MSG_DONTWAIT` flag has been set in the flags parameter, -1 is returned with the socket error being set to `EWOULDBLOCK`.

The `select()` call may be used to determine when it is possible to send more data.

Prototype

```
int sendto (int sockfd, void * buff, size_t nbytes, int flags, const
struct sockaddr *to, socklen_t addrlen);
```

Parameters

Parameter	Description
<code>sockfd</code>	The socket descriptor to use to send data.
<code>bufferPtr</code>	The buffer to send.
<code>bufferLength</code>	The length of the buffer to send.
<code>toPtr</code>	The address to send the data to.
<code>toLength</code>	The length of the to area pointed to by <code>toPtr</code> .
<code>flags</code>	See below

The flags parameter is formed by ORing one or more of the following:

Value	Description
<code>MSG_DONTWAIT</code>	Do not wait for data send to complete, but rather return immediately.
<code>MSG_DONTROUTE</code>	The <code>SO_DONTROUTE</code> option is turned on for the duration of the operation. Only diagnostic or routing programs use it.
0	Writes data as normal data

Return Values

The function returns the following values:

Value	Description
<code>>= 0</code>	Number of bytes actually sent on the socket
<code>-1</code>	An error occurred
<code>EHOSTDOWN</code>	Destination host is down.

If the return value is `-1`, `errno` will be set to one of the following values.

`sendto()` can fail for any of the following reasons:

errno	Description
<code>EBADF</code>	The socket descriptor is invalid.
<code>ENOBUFS</code>	There was insufficient user memory available to complete the operation.
<code>EINVAL</code>	One of the parameters is invalid: the <code>bufferPtr</code> is <code>NULL</code> , the <code>bufferLength</code> is less than or equal to 0, an unsupported flag is set, <code>toPtr</code> is <code>NULL</code> or <code>toLength</code> contains an invalid length.
<code>EHOSTUNREACH</code>	No route to destination host.
<code>EMSGSIZE</code>	The socket requires that message be sent automatically, and the message was too long.
<code>EPROTOTYPE</code>	TCP protocol requires usage of <code>send()</code> not <code>sendto()</code> .
<code>EWOULDBLOCK</code>	The socket is marked as non-blocking and the <code>send()</code> operation would block.

shutdown()

shutdown() shuts down a socket in read, write, or both directions determined by the parameter `howToShutdown`.

Prototype

```
int shutdown (int sockfd, int howToShutdown);
```

Parameters

Parameter	Description
<code>sockfd</code>	The socket to shutdown
<code>howToShutdown</code>	Direction: 0 = Read 1 = Write 2 = Both

Return Values

The function returns the following values.:

Value	Description
0	Success
-1	An error occurred

If the return value is -1, `errno` will be set to one of the following values.

shutdown() can fail for any of the following reasons:

errno	Description
EBADF	The socket descriptor is invalid
EINVAL	One of the parameters is invalid
EOPNOTSUPP	Invalid socket type - can only shutdown TCP sockets.
ESHUTDOWN	Socket is already closed or is in the process of closing.

socketclose()

This function is used to close a socket. It is not called close to avoid confusion with an embedded kernel file system call.

Prototype `int socketclose (int sockfd);`

Parameters

Parameter	Description
sockfd	The socket descriptor to close

Return Values The function returns the following values:

Value	Description
0	Operation completed successfully
-1	An error occurred

If the return value is -1, errno will be set to one of the following values.

socketclose() can fail for the following reasons:

errno	Description
EBADF	The socket descriptor is invalid.
EALREAY	A previous socketclose() call is already in progress.
ETIMEDOUT	The linger option was on with a non-zero timeout value, and the linger timeout expired before the TCP close handshake with the remote host could complete (blocking TCP socket only).

socketerrno()

This function is used when any socket call fails (`SOCKET_ERROR`), to get the error value back. This call has been added to allow for the lack of a per-process `errno` value that is lacking in most embedded realtime kernels.

Prototype `int socketerrno(int sockfd);`

Parameters

Parameter	Description
<code>sockfd</code>	The socket descriptor to get the error on.

Return Values The last `errno` value for a socket.

select()

`select()` examines the socket descriptor sets whose addresses are passed in `readSocketsPtr`, `writeSocketsPtr`, and `exceptionSocketsPtr` to see if any of their socket descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. Out-of-band data is the only exceptional condition. The `numberSockets` argument specifies the number of socket descriptors to be tested. Its value is the maximum socket descriptor to be tested, plus one. The socket descriptors from 0 to `numberSockets - 1` in the socket descriptor sets are examined. On return, `select()` replaces the given socket descriptor sets with subsets consisting of those socket descriptors that are ready for the requested operation. The return value from the call to `select()` is the number of ready socket descriptors. The socket descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such file descriptor sets:

Value	Description
<code>FD_ZERO (&fdset);</code>	Initializes a socket descriptor set (<code>&fdset</code>) to the null set.
<code>FD_SET (fd, &fdset);</code>	Includes a particular socket descriptor <code>fd</code> in <code>fdset</code> .
<code>FD_CLR (fd, &fdset);</code>	Removes <code>fd</code> from <code>fdset</code> .
<code>FD_ISSET (fd, &fdset);</code>	Is non-zero if <code>fd</code> is a member of <code>fdset</code> , zero otherwise.
<code>FD_COPY(const fd_set *f, fd_set *t)</code>	Copies all bits from one such structure to another such structure

NOTE



The term `fd` is used for BSD compatibility since `select()` is used on both file systems and sockets under BSD UNIX.

The timeout parameter specifies a length of time to wait for an event to occur before exiting this routine. `struct timeval` contains the following members:

Value	Description
<code>tv_sec</code>	Number of seconds to wait
<code>tv_usec</code>	Number of microseconds to wait

If the total time is less than one millisecond, `select()` will return immediately to the user.

Prototype

```
int select(int maxfd, fd_set *in, fd_set *out, fd_set *ex, struct timeval *timeout);
```

Parameters

Parameter	Description
numberSockets	Biggest socket descriptor to be tested, plus one.
readSocketsPtr	The pointer to a mask of sockets to check for a read condition.
writeSocketsPtr	The pointer to a mask of sockets to check for a write condition.
exceptionSocketsPtr	The pointer to a mask of sockets to check for an exception condition: Out of Band data.
timeOutPtr	The pointer to a structure containing the length of time to wait for an event before exiting.

Return Values

The function returns the following values:

Value	Description
> 0	Number of sockets that are ready
0	Time limit exceeded
-1	An error occurred

If the return value is -1, errno will be set to one of the following values:

errno	Description
EBADF	One of the socket descriptors is bad.

socketset_owner()

socketset_owner() transfers ownership of an open socket to another task. Following this call the caller will not be able to access the socket. No changes to the socket state are made and buffers are not cleared. (The caller may wish to do this before transferring control.) Pending events for the socket are not transferred to the new task.

Prototype

```
int socketset_owner(int sockfd, int task_id);
```

Parameters

Parameter	Description
sockfd	Socket descriptor
task_id	Task ID

Return Values

The function returns the following values:

Value	Description
0	Success.
-1	Error

If the return value is -1, errno will be set to one of the following values.

errno	Description
EBADF	Invalid handle, or caller does not own device
EINVAL	Invalid task number

socketioctl()

This function is used to set/clear non-blocking I/O, to get the number of bytes to read, or to check whether the specified socket's read pointer is currently at the out of band mark. It is not called ioctl to avoid confusion with an embedded kernel file system call.

Parameter	Description
FIONBIO	Set/clear nonblocking I/O: if the int cell pointed to by <i>argumentPtr</i> contains a non-zero value, then the specified socket non-blocking flag is turned on. If it contains a zero value, then the specified socket non-blocking flag is turned off.
FIONREAD	Stores in the int cell pointed to by <i>argumentPtr</i> the number of bytes available to read from the socket descriptor.
SIOCATMARK	Stores in the int cell pointed to by <i>argumentPtr</i> a non-zero value if the specified socket's read pointer is currently at the out-of-band mark, zero otherwise. See <i>revc</i> call for a description of out-of-band data.

Prototype

```
int socketioctl(int sockfd, int cmd, int*arg);
```

Parameters

Parameter	Description
<i>sockfd</i>	The socket descriptor we want to perform the ioctl request on.
<i>request</i>	FIONBIO, FIONREAD, or SIOCATMARK
<i>argumentPtr</i>	A pointer to an int cell in which to store the request parameter or request result.

Return Values

The function returns the following values.:

Value	Description
0	Success.
-1	An error has occurred.

If the return value is -1, *errno* will be set to one of the following values:

errno	Description
EBADF	The socket descriptor is invalid.
EINVAL	Request is not one of FIONBIO, FIONREAD, or SIOCATMARK.

getsocketevents()

`getsocketevents()` takes a socket handle as a parameter. It returns the socket event mask for the specified socket. This is normally called after receiving a Verix event on socket.

Prototype

```
int getsocketevents(int sockfd, int *events);
```

Return Values

Value	Description
0	Success
-1	Error

Parameters

Parameter	Description
<code>sockfd</code>	Socket descriptor

`getsocketevents()` can fail for any of the following reasons:

errno	Description
EBADF	Invalid handle
EACCES	Events refer to an invalid address

The socket event mask may contain 0 or more of the following events:

Event Flags	Description
<code>SE_CONNECT_COMPLT</code>	Non-blocking connect issued earlier by the user has now completed, and the connection is established with the peer on the socketDescriptor. The user is now able to send and receive data on the connection given by the socketDescriptor.
<code>SE_ACCEPT</code>	A remote host has established a connection to the listening server socketDescriptor. The user can now issue an accept call to retrieve the socket descriptor of the newly established connection.
<code>SE_RECV</code>	Incoming data has arrived on the connection given by socketDescriptor. The user can now issue any of the allowed recv calls for the protocol associated with the socket to retrieve the incoming data.
<code>SE_RECV_OOB</code>	Incoming out-of-band data has arrived on the connection given by socketDescriptor. The user can use the appropriate method to retrieve the out of band data as described in the recv section above.
<code>SE_SEND_COMPLT</code>	Some sent data has been received, and asked by the peer.

Event Flags	Description
SE_REMOTE_CLOSE	Peer has shutdown the connection (sent a FIN) with no more incoming new data coming. The user should empty the socket receive queue using any of the receive calls. The user must then close the connection.
SE_WRITE_READY	Indicates that there is more room on the send queue. The user can now send more data on the connection given by the socket descriptor.
SE_SOCKET_ERROR	An error has occurred on the connection. The user can issue a send or recv call to retrieve the error as described in the send and recv sections. Note that recv will return all outstanding incoming data before returning the error. The user should then issue close to close the connection.
SE_RESET	The peer has sent a RESET. The user needs to issue close to close the socket.
SE_CLOSE_COMPLT	The user issued close has now completed.

Host Information APIs

Use the following APIs for host information:

- `gethostbyname()`
- `gethostbyaddr()`
- `DnsGetHostByAddr()`
- `DnsGetHostByName()`
- `freehostent()`
- `DnsSetServer()`
- `DnsSetUserOption()`

gethostbyname()

Provides a pointer to a host entity (specified by `struct hostent`) identified by `hostname`. The host entity contains the IP address and other information. This function allocates the memory for the `hostent` structure, the `hostname` string, and a variable size array of IP addresses that belong to this host. The user must call `freehostent` to release this memory.

Prototype `int gethostbyname(const char *hostname, struct hostent **he)`

Parameters

Parameter	Description
<code>hostname</code>	Hostname to resolve
<code>he</code>	Pointer to a pointer to <code>hostent</code> structure. The memory for the <code>hostent</code> structure is allocated by this function and a pointer to this structure is given to the host using this parameter. The user must call <code>freehostent</code> to release this memory.

Return Values

Value	Description
<code>ETIMEDOUT</code>	Attempts to query if the DNS has timed-out. The user should continue to call <code>gethostbyname</code> to restart DNS query attempts.
<code>ENOMEM</code>	Could not allocate memory for a cache entry or <code>hostname</code> array.
<code>ENOBUFS</code>	Temporarily ran out of buffers for sending a DNS request. Try again later.
<code>EINVAL</code>	One of the input parameters is invalid.
<code>EWouldBlock</code>	DNS lookup in progress. The user should continue to call <code>gethostbyname</code> with the same parameters until it returns a value other than <code>EWouldBlock</code> . Only returned in non-blocking mode.
<code>0</code>	Success

gethostbyaddr()

Provides a pointer to a host entity (specified by `struct hostent`) which contains the host name and other information about the host. It is identified by a given IP address. This function allocates the memory for the `hostent` structure, the `hostname` string and the IP address. The user must call `freehostent` to release this memory.

Prototype

```
int gethostbyaddr(const char *addr, socklen_t len, int family, struct hostent **he)
```

Parameters

Parameter	Description
<code>addr</code>	Pointer to byte array that specifies the IP address.
<code>len</code>	The length of the IP address, pointed to by <code>addr</code> . This parameter is currently unused as only IPv4 is supported. This should be set to 4 in the event that this function adds support for IPv6.
<code>family</code>	Specifies the address family to which the IP address belongs. This parameter is currently unused as only IPv4 is supported. This should be set to <code>AF_INET</code> in the event that this function adds support for IPv6.
<code>he</code>	This function will pass back a pointer to a <code>hostent</code> structure. The memory for the <code>hostent</code> structure is allocated by this function and a pointer to this structure is given to the host using this parameter.

Return Values

Value	Description
<code>ETIMEDOUT</code>	Attempts to query if the DNS has timed-out. The user should continue to call <code>gethostbyaddr</code> to restart DNS query attempts.
<code>ENOMEM</code>	Could not allocate memory for a cache entry or <code>hostname</code> array.
<code>ENOBUFS</code>	Temporarily ran out of buffers for sending a DNS request. Try again later.
<code>EINVAL</code>	One of the input parameters is invalid.
<code>EWouldBlock</code>	DNS lookup in progress. The user should continue to call <code>gethostbyaddr</code> with the same parameters until it returns a value other than <code>EWouldBlock</code> . Only returned in non-blocking mode.
<code>0</code>	Success

DnsGetHostByAddr()

This function retrieves the hostname associated with the given IP address.

Prototype

```
int DnsGetHostByAddr(in_addr_t ip, char *hostname, int hostnameLen)
```

Parameters

Parameter	Description
ip	IP address of which this function retrieves the hostname.
hostname	Pointer to an array of characters in which the hostname will be returned to the user. The user must allocate the necessary memory pointed to by <code>hostname</code> and specify the length of that memory using <code>hostnameLen</code> .
hostnameLen	Specifies the length of the character array pointed to by <code>hostname</code> .

Return Values

Value	Description
TM_DNS_ECACHE_FULL	Could not find a cache entry to remove. problem with finding a complete entry and with <code>ref</code> count as 0.
TM_ENOMEM	Could not allocate memory for a cache entry or <code>hostname</code> array.
TM_EINPROGRESS	Cache entry incomplete, or <code>ref</code> count is positive.
TM_ENOBUFS	Temporarily ran out of buffers for sending a DNS request. Try again later.
EACCES	Could not access the memory space. This should never happen.
TM_EMFILE	Could not open a socket.
TM_DNS_EANSWER	Cache entry is stale (invalid) but <code>ref</code> count is positive. Need to try again.
EINVAL	One of the input parameters is invalid.
EWOULDBLOCK	DNS lookup in progress. The user should continue to call <code>DnsGetHostName</code> with the same parameters until it returns a value other than <code>TM_EWOULDBLOCK</code> . This response is only returned in non-blocking mode.
0	Success

DnsGetHostByName()

This function retrieves the IP address associated with the given hostname.

Prototype `int DnsGetHostByName(const char *hostname, in_addr_t *ip);`

Parameters

Parameter	Description
hostname	Hostname to resolve
ip	Pointer to the IP address which is returned to the user by this parameter.

Return Values

Value	Description
ETIMEDOUT	Attempts to query if the DNS has timed-out. The user should continue to call <code>DNSgethostbyname</code> to restart DNS query attempts.
ENOMEM	Could not allocate memory for a cache entry or <code>hostname</code> array.
ENOBUFS	Temporarily ran out of buffers for sending a DNS request. Try again later.
EINVAL	One of the input parameters is invalid.
EWouldBlock	DNS lookup in progress. The user should continue to call <code>DnsGetHostbyname()</code> with the same parameters until it returns a value other than <code>EWouldBlock</code> . Only returned in non-blocking mode.
0	Success

freehostent()

Frees the memory associated with host entity. The host entity was previously allocated by a call to `gethostbyname` or `gethostbyaddr`.

Prototype `int freehostent(struct hostent *he)`

Parameters

Parameter	Description
he	Pointer to host entry structure returned

Return Values

Value	Description
1	Error. This value is returned if <code>he = NULL</code> . This function will not check if the pointer is valid and will assume a correctly allocated host entity.
0	Success

DnsSetServer()

This function sets the address of the primary and secondary DNS server. To set the primary DNS server, serverNumber should be set to DNS_PRI_SERVER; for the secondary server it should be set to DNS_SEC_SERVER. To remove a previously set entry, set serverIpAddress to zero.

Prototype `int DnsSetServer(ip_addr serverIpAddress, int serverNumber);`

Parameters

Parameter	Description
serverIpAddress	IP address of the DNS server
serverNumber	Primary or secondary server

Return Values The function returns the following values:

Value	Description
0	Success
-1	Error

If the return value is -1, errno will be set to one of the following values.

errno	Description
EINVAL	serverNumber is not DNS_PRI_SERVER or DNS_SEC_SERVER.
ENOERROR	DNS server set successfully.

DnsSetUserOption()

This function sets various DNS options which are outlined below:

Option Type	Type	Description
DNS_OPTION_RETRIES	(int)	Maximum number of times of retransmit a DNS request.
DNS_OPTION_CACHE_SIZE	(int)	Maximum number of entries in the DNS cache. Must be greater than zero.
DNS_OPTION_TIMEOUT	(int)	Amount of time (in seconds) to wait before retransmitting a DNS request.
DNS_OPTION_CACHE_TTL	(int)	The maximum amount of time to keep a DNS response in the cache. Note: This value only affects new entries as they are added to the cache. Existing entries will not be changed.

Prototype `int DnsSetUserOption(int optType, void *optValue, int optLen);`

Parameters

Parameter	Description
optionType	See above
optionValuePtr	Pointer to the value for above option
optionLen	Length, in bytes, of the value pointed to by optionValuePtr

Return Values

The function returns the following values:

Value	Description
0	Success
-1	Error

If the return value is -1, errno will be set to one of the following values.

errno	Description
EINVAL	Invalid value for above option.
ENOPROTOPT	Option not supported (not in above list).
ENOERROR	Option set successfully.

Ping APIs

Use the following APIs for Ping functions:

- PingOpenStart()
- PingGetStats()
- PingCloseStop()

PingOpenStart()

This function opens an ICMP socket and starts sending PING echo requests to a remote host as specified by the `remoteHostName` parameter. PING echo requests are sent every `pingInterval` seconds. The PING length (not including IP and ICMP headers) is given by the `pingDataLength` parameter. To get the PING connection results and statistics, the user must call `PingGetStats()`. To stop the system from sending PING echo requests and to close the ICMP socket, the user must call `PingCloseStop()`.

Prototype `int PingOpenStart(const char *remoteHost, int seconds, int datalen);`

Parameters

Parameter	Description
<code>remoteHostNamePtr</code>	Pointer to character array containing a dotted decimal IP address.
<code>pingInterval</code>	Interval in seconds between PING echo requests. If set to zero, defaults to 1 second.
<code>pingDataLength</code>	User Data length of the PING echo request. If set to zero, defaults to 56 bytes. If set to a value between 1, and 3, defaults to 4 bytes.

Return Values The function returns the following values:

Value	Description
> 0	Socket descriptor
-1	Error

If the return value is -1, `errno` will be set to one of the following values:

errno	Description
EINVAL	<code>remoteHostNamePtr</code> was a null pointer
EINVAL	<code>pingInterval</code> was negative
EINVAL	<code>pingDataLength</code> was negative or bigger than 65595, maximum value allowed by the IP protocol.
ENOBUFS	There was insufficient user memory available to complete the operation.
EMSGSIZE	<code>pingDataLength</code> exceeds socket send queue limit, or <code>pingDataLength</code> exceeds the IP MTU, and fragmentation is not allowed.
EHOSTUNREACH	No route to remote host

PingGetStats()

This function gets Ping statistics in the `ttPingInfo` structure that `pingInfoPtr` points to. `sockfd` should match the socket descriptor returned by a call to `PingOpenStart`. `pingInfoPtr` must point to a `ttPingInfo` structure allocated by the user.

Prototype

```
int PingGetStats(int sockfd, PingInfo *pingInfoPtr);
```

Parameters

Parameter	Description
<code>sockfd</code>	The socket descriptor as returned by a previous call to <code>PingOpenStart</code> .
<code>pingInfoPtr</code>	The pointer to an empty structure where the results of the PING connection will be copied upon success of the call. (See below for details.)

`PingInfo` Data structure:

Field	Data Type	Description
<code>pgiTransmitted</code>	unsigned long	Number of transmitted PING echo request packets so far.
<code>pgiReceived</code>	unsigned long	Number of received PING echo reply packets so far (not including duplicates)
<code>pgiDuplicated</code>	unsigned long	Number of duplicated received PING echo reply packets so far.
<code>pgiLastRtt</code>	unsigned long	Round trip time in milliseconds of the last PING request/reply.
<code>pgiMaxRtt</code>	unsigned long	Maximum round trip time in milliseconds of the PING request/ reply packets.
<code>pgiMinRtt</code>	unsigned long	Minimum round trip time in milliseconds of the PING request/reply packets.
<code>pgiAvgRtt</code>	unsigned long	Average round trip time in milliseconds of the PING request/reply packets.
<code>pgiSumRtt</code>	unsigned long	Sum of all round trip times in milliseconds of the PING request/reply packets.
<code>pgiSendErrorCode</code>	int	PING send request error code if any.
<code>pgiRecvErrorCode</code>	int	PING rcv error code if any (including ICMP error from the network).

Return Values This function call returns the following values:

0	Success
-1	Error

If the return value is -1, `errno` will be set to one of the following values.

errno	Description
EBADF	sockfd is not a valid descriptor.
EINVAL	pingInfoPtr is a NULL pointer. sockfd was not opened with PingOpenStart.

Example The sample code for structure `PingInfo` is shown below.

```
typedef struct PingInfo
{
    struct sockaddr_storage pgiPeerSockAddr;    /* Peer socket address */
    unsigned long pgiLastRtt;                  /* Last round trip time computed */
    unsigned long pgiMaxRtt;                   /* maximum round trip time */
    unsigned long pgiMinRtt;                   /* minimum round trip time */
    unsigned long pgiAvgRtt;                   /* average round trip time */
    unsigned long pgiSumRtt;                   /* sum of all round trip time received so far
                                              (to compute average) */

    unsigned long pgiTransmitted;              /* number of ping transmitted packets */

    unsigned long pgiReceived;                 /* number of received echo replies packets
                                              (not including duplicated
                                              packets)*/
    unsigned long pgiDuplicated;               /* Number of duplicated ping echo replies
                                              packets */
    int pgiSendErrorCode;                     /* Send Error code if any that occurred */
    int pgiRecvErrorCode;                     /* Recv Error code if any that occurred */
} PingInfo;
```

PingCloseStop()

This function stops the sending of any PING echo requests and closes the ICMP socket that had been opened via PingOpenStart.

Prototype `int PingCloseStop(int sockfd);`

Parameters

Parameter	Description
socketDescriptor	An ICMP PING socket descriptor as returned by PingOpenStart

Return Values

Value	Description
0	Success
-1	Error

If the return value is -1, errno will be set to one of the following values.

errno	Description
EBADF	socketDescriptor is not a valid descriptor

IP Address Conversion APIs

The IP address conversion API enables conversion of IP address strings in dotted decimal format to 32-bit IPv4 address, and from 32-bit IPv4 address to IP address strings in dotted decimal format.

The IP address conversion functions are:

- `inet_addr()`
- `inet_aton()`
- `inet_ntoa()`

inet_addr()

This function converts a NULL-terminated C string pointed to by `strptr` to a 32-bit IPv4 address and returns this value. If the input string is invalid, this function returns 0xFFFFFFFF.

Prototype `unsigned long inet_addr(char* strptr);`

Parameters

<code>strptr</code>	IPv4 address string in dotted decimal format (for example, "15.101.103.104")
---------------------	--

Return Values

Value other than 0xFFFFFFFF	32-bit network byte-ordered IP address
0xFFFFFFFF	<code>strptr</code> is NULL or contains invalid characters.

inet_aton()

This function converts a C string pointed to by `strptr` to a 32-bit IPv4 address, stored through the pointer `addrptr`.

Prototype `int inet_aton(char* strptr, struct in_addr* addrptr);`

Parameters

<code>strptr</code>	IPv4 address string in dotted decimal format, (for example, "15.101.103.104")
<code>addrptr</code>	Pointer to the <code>in_addr</code> structure, where the 32-bit IPv4 address is returned

Return Values

Success:	1
Failure:	0: <ul style="list-style-type: none">• String invalid. Possibly because the string contains invalid characters or each decimal number exceeds 255.• <code>strptr</code> NULL.• <code>addrptr</code> NULL.

inet_ntoa()

This function converts a 32-bit IPv4 address into its corresponding dotted-decimal string.

Prototype `char* inet_ntoa(void* addrPtr, char* strPtr, int size);`

Parameters

<code>addrPtr</code>	Pointer to the <code>struct in_addr</code> structure containing the 32-bit IPv4 address.
<code>strPtr</code>	Character buffer for storing the IPv4 dotted decimal address string.
<code>size</code>	Size of <code>strPtr</code> . <code>size</code> must be at least 16 bytes.

Return Values

Pointer to dotted decimal string.	Pointer to the string containing the dotted decimal IPv4 address string.
NULL	<code>addrPtr</code> NULL, <code>strPtr</code> is NULL, or <code>size</code> is less than 16.

FTP Protocol Support

This section describes FTP protocol support in the TCP/IP stack.

The TCP/IP stack supports two FTP functions, namely:

- `int ftpGet(ftp_parm_t *ftp);`
- `int ftpPut(ftp_parm_t *ftp);`

where `ftp_parm_t` is defined in `SVC_NET.h`.

The following sample code illustrates how to use `ftpGet`:

[ftpGet.txt](#)

(Click to view file attachments)

Packet Capture

The stack outputs to the system log network packets in packet capture (PCAP) format. Since the system log is designed for printable ASCII data and PCAP is binary data, the PCAP data is converted to ASCII hex characters. A program must post-process the syslog data to extract the PCAP data into a binary file for WireShark to read.

The lines between `<PCAP_B>` and `<PCAP_E>` must contain hex characters (0-9, A-F) which represent one PCAP record. This may contain more than one line, if needed.

```
<PCAP_B>
001122334455...
DDEEFF...
<PCAP_E>
```

To eliminate the risk of accidentally revealing card data, this feature limits the length of each packet to the first 100 bytes. This should include enough of the network packet headers for troubleshooting and debugging.

The `GID 1 CONFIG.SYS` variable `*PCAP` will control this feature.

`*PCAP=1` The stack writes PCAP records to the system log.

If `*PCAP` is another other value or not present, the stack does not write PCAP records to the system log.



Secure Sockets Layer (SSL)

Verix eVo SSL will be a port of OpenSSL 0.9.8n with changes required to interface with Verix V security features such as hardware random number generator. For Trident, hardware crypto accelerators for AES and RSA will be interfaced to OpenSSL.

While the primary use for OpenSSL is for SSL over TCP for transactions, it will also be used for IP downloads. WPA/WPA2 Enterprise supplicants require SSL over Ethernet (no IP, no TCP) support so the library must also support this mode of operation.

OpenSSL API

Rarely used, proprietary, or flawed crypto algorithms such as blowfish, Camellia, CAST, Diffie-Hellman, DSA, elliptic curve, IDEA, Kerberos, mdc2, ripemd, rc5, ssl1, and ssl2 are not included.

Include Files

Crypto Functions

aes.h	md2.h	rc2.h
asn1.h	md5.h	rc4.h
bio.h	obj_mac.h	rsa.h
bn.h	objects.h	safestack.h
buffer.h	ocsp.h	sha.h
comp.h	opensslconf.h	ssl2.h
conf.h	opensslv.h	ssl3.h
crypto.h	ossl_typ.h	ssl23.h
des.h	pem2.h	ssl.h
dtls1.h	pem.h	stack.h
e_os2.h	pkcs7.h	symhacks.h
err.h	pkcs12.h	tls1.h
evp.h	pq_compat.h	x509.h
hmac.h	pqueue.h	x509_vfy.h
lhash.h	rand.h	x509v3.h

SSL Functions

The following is from the ssl main page.

Currently the OpenSSL ssl library provides the following C header files containing the prototypes for the data structures and functions:

`ssl.h`

The common header file for the SSL/TLS API. Include it into your program to make the SLL library APIs available. It internally includes both private SSL headers and headers from the crypto library. For hard-core details on the internals of the SSL API, look inside this header file.

`ssl3.h`

The sub header file that deals with the SSLv3 protocol only. This sub header file is already included in `ssl.h`.

`ssl23.h`

The sub header file that deals with the combined use of the SSLv2 and SSLv3 protocols. This sub header file is already included in `ssl.h`.

`tls1.h`

The sub header file that deals with the TLSv1 protocol only. This sub header file is already included in `ssl.h`.

Libraries

The crypto and SSL libraries are built as Verix V shared libraries.

`ssl.lib` — SLL shared library

`ssl.o` — Stub functions. Applications link with this file.

Crypto Functions

See the OpenSSL crypto library main pages for details.

<http://www.openssl.org/docs/crypto/crypto.html>

SSL Functions

See the OpenSSL SSL main page for details.

<http://www.openssl.org/docs/ssl/ssl.html>

Verix eVo SSL Versus VeriFone SSL Library

The Verix eVo SSL library is a relatively straight forward port of OpenSSL to Verix V. The IP stack socket functions do not interface to the SSL library so the SSL library must be called to use SSL. OpenSSL is designed to operate in this fashion where the SSL library calls the socket functions as needed rather than the socket functions calling OpenSSL.

Best Practices

To maximize the level of security and remain MasterCard PTS compliant the following ciphers are used:

- AES256-SHA
- DES-CBC3-SHA
- AES128-SHA.

Additionally, RSA or DSA should be used with key lengths at least 1024 bits or higher. PCI PTS sections addressing Open Protocol have deprecated SHA1 in favor of the stronger SHA2. This guidance must be followed as part of the compliance requirements with PCI.



Configuration Management

New SW components are being introduced as part of Verix eVo SW Architecture. Having separate modules with specific responsibilities, moving from static dependencies to loading modules at run-time, all facilitates future enhancements with minimal impact to SW certifications.

While implementing new components enhances current SW architecture, it adds overhead when trying to handle configuration files and parameters for each specific module.

This section describes the standardized configuration file format to use across all Verix eVo components requiring external configuration/customization.

Files editable by customers and internal Verix eVo files follow a similar format, but they are designed to handle different details of the same information. This document presents a common template to use for both types of files. Content and usage within Verix eVo is outside the scope of this document.

Verix eVo components will be released with default settings. Customers will have the ability to overwrite some / all parameters during deployment via a delta file.

Configuration Precedence

All configuration parameters have a default value that CommEngine uses if none is specified. To override the default values, the user has the option to specify the value in the configuration delta file (.INI, see [Network Configuration via Configuration Delta File](#)). By specifying a `CONFIG.SYS` parameter, it overrides the value in the delta file. [Table 4](#) details the configuration precedence.

Table 4 Configuration Precedence

Location	Precedence
Default value	Lowest
Configuration value in delta file (Network Configuration via Configuration Delta File)	
Configuration value in <code>CONFIG.SYS</code> (Network Configuration via Configuration Delta File)	Highest

CommEngine at start up reads the `CONFIG.SYS` values and updates the configuration delta file. After the value is updated, the `CONFIG.SYS` value is deleted.

When a value is accessed, CommEngine looks for it in the delta file and if not present uses the default value.

Case Sensitivity of Configuration Parameter Values

Verix eVo configuration parameters are case insensitive as Verix eVo automatically converts lower case letter to upper case. For example the configuration parameter `PPPDIAL.USERNAME` whose value is “verifone” results in “VERIFONE” (without the quotes).

If the parameter value is escaped in special characters (`/ * */`) then CommEngine will treat it as lower case letters. For example:

```
PPPDIAL.USERNAME = /*VERIFONE*/
```

For VeriFone the sequence is `V/*ERI*/F/*ONE*/`. In this case all the letters are in lower case except for V and F.

Configuration Options

To override the default value users have two options – the delta file and `CONFIG.SYS`. The two options are not mutually exclusive and using one does not preclude the other. However, when both options are used, the value specified via `CONFIG.SYS` takes precedence.

This gives users the option to specify the configuration via the delta file and is usually applicable to its entire terminal base. However should a few terminals require a different configuration, it can be altered by specifying `CONFIG.SYS` parameters rather than having a different delta file.

Configuration File Format

The configuration file can be described as a collection of one or more tables and each table consisting of one or more records. Each record may optionally consist of one or more attribute.

The following is a sample CE Delta File:

[CE_Delta_File.txt](#)

(Click to view file attachments)

A configuration file may contain comment and blank lines to make the file self documenting and readable. A comment line starts with the character `;` (0x3B) may be followed by any character and terminated by end of line (CRLF 0x0D0A). Similarly a blank line is any number (zero or more) tabs (0x09) and `/` or space (0x20) characters terminated with an end of line.

The following is a sample Ethernet Driver Delta File:

[ETH_Delta_File.txt](#)

(Click to view file attachments)

Internal and External Configuration Files

While delta files can be updated by just downloading a new delta file to GID 1. It is recommended that eVo users update Device Driver information via NCP or using ceAPIs `ceSetDDParamValue()` or `ceGetDDParamValue()`.

Verix eVo Configuration Files

These files are designed and generated by Verix eVo team. They will be included as part of the default Verix eVo installation package and will reside on GID 46, visible to Verix eVo components only.

Because total space precedes file being readable, for all Verix eVo setup files concise notation should be used. Short and meaningful comments are still encouraged.

Factory Default Settings

Every Verix eVo component relying on Configuration Files includes its 'factory default' settings. These settings allow every component to operate even if no custom settings are provided. These values also become the recommended Verix eVo settings for specific components (i.e. device timeouts).

Using the sample code described in CE Delta File, the following configuration file allows an Ethernet device to connect to Ethernet using static configuration with IP address 10.64.80.175.

The following is a sample of the CE Default Settings:

[CE_Default_File.txt](#)

(Click to view file attachment)

Metadata File

This file will be used by Verix eVo components to manipulate dynamic settings. Metadata files reside in GID 46 and are designed and generated by the Verix eVo Volume II team. This file states the limits on the parameters users want to set.

The following is a sample of a metadata file:

[Metadata.txt](#)

(Click to view file attachment)

Customer Configuration File

This file allows customers to provide specific configuration values. Following filename, extension and format defined by Verix eVo, this file will be provided during installation on GID 1. Verix eVo will read its default settings and will overwrite them with the customer's settings.

The following is a sample customer configuration file:

[Customers_Configuration_File.txt](#)

(Click to view file attachment)

Naming Convention and Location

The table below details the naming convention and location.

Location	Filename	Description
N:46 or I:46	<basename>.INI	Default configuration file. This file contains the name and value of configurable parameters. For CE, the master file or the default configuration file is created at run-time and stored in I:46.
I:1	<basename>.INI	User's configuration file. This file contains custom settings defined during deployment. Values on this file will overwrite those configured on N:46/<basename>.INI. Changes via API "update" will be reflected on this file.
N:46	<basename>.MTD	This file contains the name and editable information to configure parameters. This file is primarily for Verix eVo processing. No run-time changes should happen to this file.

Network Interface Configuration

When starting NWIF users may want to set or get IP and PPP configurations, eVo users may use NCP to update these settings. eVo users may also use ceAPI [ceSetNWPParamValue\(\)](#) and [ceGetNWPParamValue\(\)](#).

eVo application developers may set or get the device driver parameters depending on the driver that is available per terminal. eVo users may also use NCP to update these settings. eVo users may also use ceAPI [ceSetDDParamValue\(\)](#) and [ceGetDDParamValue\(\)](#).

eVo users are recommended to use the macros defined in `vos_ddi_ini.h` and `vos_ioctl.h` when updating device driver parameters at run-time via ceAPI.

The available IOCTL calls for all device drivers are listed in [Table 5](#).

Table 5 IOCTL Calls Available for all Device Drivers

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_DRIVER_BUILD_DATE	None	NA	String Ex: Jan 4 201014:02:01	Response is the build date of the driver
IOCTL_GET_EXTENDED_ERROR	None	NA	String Ex: Ethernet Port Open Failed	Response is the extended error message encountered
IOCTL_GET_DRIVER_STATES	None	NA	String Ex: 10 31 21 41	Response is the network, power, communication, and connection states of the device

Table 5 IOCTL Calls Available for all Device Drivers (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_EXTENDED_ERROR_CODE	None	NA	String Ex: E_DDI_PORT_OPEN_FAIL	Response is the extended error code encountered (see Extended Error Codes and Messages)

Ethernet Device Driver

The available IOCTL calls for Ethernet device driver are listed in [Table 6](#).

Table 6 IOCTL Calls Available for Ethernet

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_MAC	None	NA	String Ex: 00-0B-4F-30-31-C4	Response is the MAC address associated to the device
IOCTL_GET_LINK_SPEED	None	NA	String Ex: 100 MB/full	Response is the link speed

The configuration parameters for Ethernet DDI DDL are listed in [Table 7](#).

Table 7 Ethernet DDI DDL Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Attribute R= read only RW= read/	Data Type	Config.sys	Description
INI_ALLOW_MULCAST	0	0-1	RW	number	ETI_MULCAST	If set to 1 allows reception of broadcast packets. Note: Since Trident Ethernet is not a USB device setting ETI_MULCAST to 1 will have no affect on the driver.

Dial Device Driver

The available IOCTL calls for the Dial device driver is listed in [Table 8](#).

Table 8 IOCTL Calls Available for Dial

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_CONNECTION_BAUD	None	NA	String	Get the connection baud rate

The configuration parameters for Dial DDI DLL are listed in [Table 9](#).

Table 9 Dial DDI DLL Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Attribute R= read only RW= read/ write	Data Type	Value (Number) / Length (String)		Config.sys	Description
				Min	Max		
INI_MAX_CONNECTION_RETRIES	1	RW	number	1	10	CONN_RET	<p>Maximum connection retries.</p> <p>The driver first dials using the DL_PHO_PRI (primary phone number). If the dial response is BUSY, NO CARRIER or NO ANSWER, it uses the DL_PHO_SEC (secondary phone number) if provided.</p> <p>If no DL_PHO_SEC (secondary phone number) is provided it would attempt a connection retry using again the primary number until CONN_RET value is reached.</p> <p>If DL_PHO_SEC (secondary phone number) is provided and it still encounters BUSY, NO CARRIER or NO ANSWER, it would attempt a</p>

Table 9 **Dial DDI DLL Configuration Parameters** (continued)

INI Parameter Name (Type:macro)	Default Values	Attribute R= read only RW= read/ write	Data Type	Value (Number) / Length (String)		Config.sys	Description
				Min	Max		
							connection retry (call the primary number then secondary if necessary) depending on the CONN_RET value. Once a critical error is encountered such as NO LINE, NO DIAL TONE, ERROR or connection timeout, dial call is aborted be it using a primary or a secondary phone number.
INI_CONNECTION_RETRY_TIMEOUT	1000	RW	number	500	60000	CONNRET_TM	Connection retry timeout in milliseconds.
INI_MODEM_START	Conexant : AT&FE0Q0V0 Silabs: ATFE0Q0V0	RW	string	2	50	DL_START	Start up string
INI_MODEM_DIAL	ATD	RW	string	2	50	DL_DIAL	AT command string to dial. Be cautious in editing/ changing the DL_DIAL parameter value as it may affect the dialing behavior. The Dial DDI Driver sends the dial string sequence as: <DL_DIAL> <DL_DIALTYPE> <DL_PHO_PRI> or <DL_DIAL> <DL_DIALTYPE> <DL_PHO_SEC>.

Table 9 Dial DDI DLL Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Attribute R= read only RW= read/ write	Data Type	Value (Number) / Length (String)		Config.sys	Description
				Min	Max		
INI_MODEM_INIT_RETRIES	10	RW	number	1	10	DL_INITRET	Please refer to the corresponding AT Command Reference for more information. Number of times in sending the start up string until an OK is returned
INI_MODEM_AT_DELAY	250	RW	number	250	1000	DL_ATDELAY	Number of milliseconds to wait between sending AT commands
INI_MODEM_INIT_1	ATS0=2	RW	string	2	50	DL_INIT_1	Init String
INI_MODEM_INIT_2	AT\V2&D2%C0\N0	RW	string	2	50	DL_INIT_2	Init String Applicable only for Silabs DDI Driver
INI_MODEM_INIT_3	NONE	RW	string	2	50	DL_INIT_3	Init String 3
INI_MODEM_FORMAT	4	RW	number	0	5	DL_FORMAT	Format numeric representation: <ul style="list-style-type: none"> • 0 = Fmt_A7E1 • 1 = Fmt_A7N1 (for Silabs only) • 2 = Fmt_A7O1 • 3 = Fmt_A8E1 (for Silabs only) • 4 = Fmt_A8N1 • 5 = Fmt_A8O1 (for Silabs only)
INI_MODEM_AT_SEND_TIMEOUT	20000	RW	number	500	60000	DL_ATSNDTO	Timeout to send the AT command in milliseconds
INI_MODEM_HW_FLOW_CONTROL	true	RW	boolean	NA	NA	DL_HW_FCTL	Hardware flow control
INI_MODEM_INIT_TIMEOUT	20000	RW	number	500	60000	DL_INIT_TO	Response timeout used for all init AT commands in milliseconds

Table 9 **Dial DDI DLL Configuration Parameters** (continued)

INI Parameter Name (Type:macro)	Default Values	Attribute R= read only RW= read/ write	Data Type	Value (Number) / Length (String)		Config.sys	Description
				Min	Max		
INI_MODEM_CONNECT_TIMEOUT	60000	RW	number	10000	120000	DL_CONN_TO	Data connection timeout in milliseconds
INI_MODEM_DISCONNECT_TIMEOUT	1400	RW	number	500	60000	DL_DISC_TO	Data disconnect timeout in milliseconds
INI_MODEM_INTERCHAR_TIMEOUT	500	RW	number	500	60000	DL_ICHAR_TO	Inter-character timeout for all AT commands in milliseconds
INI_DIAL_PRIMARY	none	RW	string	1	128	DL_PHO_PRI	Primary phone number to dial
INI_DIAL_SECONDARY	none	RW	string	1	128	DL_PHO_SEC	Secondary phone number to dial
INI_SDLC_PROTOCOL	false	RW	boolean	NA	NA	DL_SDLC_ON	Enable/Disable SDLC protocol
INI_SDLC_INIT	Conexant : ATW2X4 S25=1&D 2%C0\N0 +A8E=,,1 Silabs: ATV2X4 &D2%C0\N0	RW	string	2	50	DL_SDLCINT	SDLC Init String
INI_SDLC_MODULATION	Conexant : AT+MS=V22,0,30 0,1200,30 0,1200 Silabs: AT+MS=V22	RW	string	2	50	DL_SDLC_MOD	AT Command string to set the modulation
INI_SDLC_FAST_CONNECT	true	RW	boolean	NA	NA	DL_FAST_CO	Enable/Disable Fast connect
INI_DIAL_TYPE	0	RW	number	0	3	DL_DIALTYPE	Dial Type Dial Type numeric representation:

Table 9 **Dial DDI DLL Configuration Parameters** (continued)

INI Parameter Name (Type:macro)	Default Values	Attribute R= read only RW= read/write	Data Type	Value (Number) / Length (String)		Config.sys	Description
				Min	Max		
INI_DIAL_TYPE	0	RW	number	0	3	DL_DIALTYPE	<ul style="list-style-type: none"> • 0 = TONE • 1 = PULSE • 2 = BLIND TONE • 3 = BLIND PULSE

The available IOCTL calls for GPRS are listed in [Table 10](#).

Table 10 **IOCTL Calls Available for GPRS**

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_DSIM_GET_USED	None	NA	String	Response is the currently used SIM while connected. <ul style="list-style-type: none"> • 1: SIM in slot 1 • 2: SIM in slot 2
IOCTL_GET_CON_NETWORK	None	NA	String	Response is the network the terminal is connected to. Possible values are: <ul style="list-style-type: none"> • 1 – GPRS • 2 – EGPRS • 3 – UMTS • 4 – HSPA • 5 – HSPA+
IOCTL_GET_FW_VERSION	None	NA	String Ex: 01.03	Response is the firmware version of the GSM/GPRS radio modem used.
IOCTL_GET_FW_SUB_VERSION	None	NA	String Ex: "01"	Response is the sub firmware version of the GSM/GPRS radio modem used.
IOCTL_GET_MODEL_ID	None	NA	String	Response is the model ID of the modem.

Table 10 **IOCTL Calls Available for GPRS** (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_MFG_ID	None	NA	String	Response is the manufacturing ID of the modem.
IOCTL_GET_OPERATOR_LAI	None	NA	String	Response is the operator's GSM Location Area Identification (LAI) which consists of the 3-digit Mobile Country Code (MCC) plus the 2- or 3-digit Mobile Network Code (MNC).
IOCTL_GET_PKT_SWITCHED_STATUS	None	NA	String	<p>Response is the last packet switch URC received from the radio.</p> <ul style="list-style-type: none"> • 0: GPRS/EGPRS not available • 1: GPRS available • 2: GPRS attached • 3: EGPRS available • 4: EGPRS attached • 5: WCDMA available • 6: WCDMA attached • 7: HSDPA available • 8: HSDPA attached • 9: HSPA available • 10: HSPA attached • 11: HSPA+ attached

Table 10 **IOCTL Calls Available for GPRS** (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_RSSI	None	NA	String Ex: 100 (means a signal strength of approximately -51 dBm)	Response is a percentage equivalent of the RSSI value retrieved from the modem.
IOCTL_GET_RSSI_ABS	None	NA	String Ex: 30 (means a signal strength of approximately -53 dBm)	Response is a range indicator of the RSSI signal, measured in dBm. Please see AT Commands document for the GSM/GPRS radio modem used.
IOCTL_GET_RSSI_DBM	None	NA	String Ex: -59 (means a signal strength percentage of 85%)	Response is a dBm approximate of the RSSI value retrieved from the modem.
IOCTL_GET_BER	None	NA	String Ex: 99 (means an unknown bit error rate)	Response is the bit error rate associated to the RSSI.
IOCTL_GET_ICCID	None	NA	String	Response is the SIM card identification number
IOCTL_GET_IMEI	None	NA	String	Response is the International Mobile Equipment Identity of the modem
IOCTL_GET_IMSI	None	NA	String	Response is the International Mobile Subscriber Identity
IOCTL_GET_LINK STATUS	None	NA	String Ex: 1 (means the data link is up)	Response can be either: <ul style="list-style-type: none"> • 1 (data link is up) • 0 (data link is not up)

Table 10 IOCTL Calls Available for GPRS (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_FORBDN_LOCINFO	None	NA	String	Response is the Public Land Mobile Network (PLMN) location information.
IOCTL_GET_GPRS_LOCINFO	None	NA	String	Response is the GPRS location information.
IOCTL_GET_GSM_LOCINFO	None	NA	String	Response is the GSM location information.
IOCTL_GET_NET_AVAIL	None	NA	String	Response is the network availability status. Note: This is only supported in 2G (MC55i/BGS2).
IOCTL_GET_OPERATOR_NAME	None	String	NA	Response is the GPRS network operator name.
IOCTL_GET_PDNET_REG	None	NA	String	Response is the packet domain network registration status.
IOCTL_GET_SERVCELL_INFO	None	NA	ddi_StCellInfo	Response is the information for the serving cell site.
IOCTL_GPS_START	None	None	NA	Starts GPS navigation.
IOCTL_GPS_STOP	None	None	NA	Stops GPS navigation.
IOCTL_GPS_GET_CONFIG	None	NA	ddi_gps_cfg*	Response is the current GPS configuration.
IOCTL_GPS_SET_CONFIG	GPS configuration information	ddi_gps_cfg*	NA	Sets the GPS configuration.
IOCTL_GPS_RESET_CONFIG	None	None	NA	Resets the GPS configuration to its default values.
IOCTL_GPS_LAS_LOC	None	NA	ddi_gps_loc*	Response is the last location after IOCTL_GPS_STOP was performed.

Table 10 **IOCTL Calls Available for GPRS** (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_SEND_TEXT_SMS	<p>Data Format: "<DestinationAddress> , <Text>"</p> <p>Where: <DestinationAddress> - Recipient's Number <Text> - Message to be sent</p> <p>Note: Any characters after the "," is part of the text message, even white spaces.</p>	String	None	Sends the message <Text> to <DestinationAddress>. The modem response from this IOCTL call is sent through an application event.
IOCTL_SEND_PDU_SMS	<p>Data Format: <Length> , <PDU></p> <p>Where: <Length> - the length of the actual TP data unit in octets <PDU> - message in octets</p> <p>Data example: "42,07915892000000F00 1000B915892214365F700 0021493A283D0795C3F33 C88FE06CDCB6E32885EC6 D341EDF27C1E3E97E72E"</p> <p>This example sends the text: "It is easy to send text messages" to +85291234567.</p> <p>Note: Any characters after the "," is part of the text message, even white spaces.</p>	String	None	Sends SMS in PDU format. The modem response from this IOCTL call is sent through an application event.
IOCTL_SET_BER_NOTIFICATION	<p>Pointer to a string that contains either 1 or 0.</p>	String	None	<p>Sets signal quality notification (in terms of bit error rate) if the application wants to have notifications or not. Valid input data values are:</p> <ul style="list-style-type: none"> • 1 - Turn on notifications • 0 - Turn off notifications

Table 10 **IOCTL Calls Available for GPRS** (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_SET_CALL_NOTIFICATION	Pointer to a string that contains either 1 or 0.	String	None	Sets call status notification if the application wants to have notifications or not. Valid input data values are: <ul style="list-style-type: none"> • 1 - Turn on notifications • 0 - Turn off notifications
IOCTL_SET_ROAM_NOTIFICATION	Pointer to a string that contains either 1 or 0.	String	None	Sets roaming status notification if the application wants to have notifications or not. Valid input data values are: <ul style="list-style-type: none"> • 1 - Turn on notifications • 0 - Turn off notifications
IOCTL_SET_SERVICE_NOTIFICATION	Pointer to a string that contains either 1 or 0.	String	None	Sets network registration notification if the application wants to have notifications or not. Valid input data values are: <ul style="list-style-type: none"> • 1 - Turn on notifications • 0 - Turn off notifications
IOCTL_SET_SMS_FORMAT	Pointer to a string that contains either "1" or "0".	String	None	Sets the SMS format of the modem. Valid input data values are: <ul style="list-style-type: none"> • 1 - Text format • 0 - PDU format

Table 10 **IOCTL Calls Available for GPRS** (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_SET_SMS_FULL_NOTIFICATION	Pointer to a string that contains either 1 or 0.	String	None	<p>Sets full SMS memory notification if the application wants to have notifications or not. Valid input data values are:</p> <ul style="list-style-type: none"> • 1 - turn on notifications • 0 - turn off notifications
IOCTL_SET_SMS_NOTIFICATION	Pointer to a string that contains either 1 or 0.	String	None	<p>Sets unread short message notification if the application wants to have notifications or not. Valid input data values are:</p> <ul style="list-style-type: none"> • 1 - Turn on notifications • 0 - Turn off notifications

Table 11 presents the member variables of `ddi_StCellInfo`.

Table 11 ddi_StCellInfo Member Variables

Parameter	Data Type	Description
ACT	int	Access Technology. Possible values are: <ul style="list-style-type: none"> NWTYPE_2G (2) NWTYPE_3G (3)
ARFCN	int	Absolute Radio Frequency Channel Number
MCC	int	Mobile Country Code (first part of the PLMN code)
MNC	int	Mobile Network Code (second part of the PLMN code)
LAC	int	Location Area Code
CELL	int	Cell ID
SvcState	int	Service State. Possible values are: <ul style="list-style-type: none"> SVCSTATE_SEARCH (1) SVCSTATE_NOCONN (2) SVCSTATE_LIMSRV (3) SVCSTATE_CELLRESEL (4) Not set (0)

NOTE



`ddi_StCellInfo` member variables are applicable to GPRS and GSM.

Table 12 presents the member variables of `ddi_gps_cfg`.

Table 12 ddi_gps_cfg Member Variables

Parameter	Data Type	Range	Description
pollTime	int	1 - 65535	Polling interval in seconds of receiving NMEA sentences from the GPS port. Default value is 1.
lstnPort	int	1 - 65535	Port number that GPSD listens to for incoming clients. Default value is 2947.
lstnIP	char[20]	9 - 15	IP address that GPSD listens to for incoming clients. Default value is localhost. Note: This is a read-only configuration since GPSD currently only allows listening to localhost.
agpsSet	int	0 - 2	Enables or disables A-GPS. The description for each value are listed below: <ul style="list-style-type: none"> 0: Disable A-GPS. 1: Enable A-GPS. No automated update of <code>GpsOneXTRA</code> binary file. 2: Enable A-GPS. Automated update of <code>GpsOneXTRA</code> binary file. (Default value)

Table 12 **ddi_gps_cfg Member Variables** (continued)

Parameter	Data Type	Range	Description
agpsFile	char[20]	1 - 15	Specifies the complete path and filename of the GpsOneXTRA binary file (named <code>xtra.bin</code>) to use when A-GPS is enabled. The binary file supplies the almanac of the GPS satellites for faster locking and requires update every 7 days. When <code>GPS_AGPS_SET</code> is 2, EOS performs automatic HTTP download of the binary file when it is found expired or the file does not exist during GPS start. Download is performed from the <code>INI</code> specified links. Terminal must have internet access to perform the download. When <code>GPS_AGPS_SET</code> is 1, it is up to the user/application to update the binary file. EOS returns an error during GPS start when loading of binary file fails. Default value is "I:1/xtra.bin".
agpsLink	char[3][64]	1 - 63	Link to Web servers where EOS connects to download the binary file. Default values are: <ul style="list-style-type: none"> • Index 0: xtra1.gpsonextra.net • Index 1: xtra2.gpsonextra.net • Index 2: xtra3.gpsonextra.net

Table 13 presents the member variables of `ddi_gps_loc`.

Table 13 **ddi_gps_loc Member Variables**

Parameter	Data Type	Description
time	double	Time of update.
ept	double	Expected time uncertainty.
latitude	double	Latitude in degrees.
epy	double	Latitude position uncertainty in meters.
longitude	double	Longitude in degrees.
epx	double	Longitude position uncertainty in meters.
altitude	double	Altitude in meters.
epv	double	Vertical position uncertainty in meters.

Table 14 lists the configuration parameters for GPRS DDI DDL driver.

Table 14 GPRS DDI DLL Driver Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_MAX_CONNECTION_ RETRIES	5	1-9999	RW	Integer	CONN_RET	Maximum number of retries to dial to GPRS service. If the dial response is BUSY, NO CARRIER, or NO ANSWER, redialling to the GPRS service will be repeated depending on the CONN_RET value. However, once a critical error is encountered such as NO LINE, NO DIAL TONE, ERROR or connection timeout (no modem response), connecting to the GPRS service will be aborted. Setting to 9999 disables the retry limit and causes the retry operation to be infinite.
INI_CONNECTION_RETRY_ TIMEOUT	12000	500- 120000	RW	Integer	CONNRET_TM	The time interval between connection retries (in milliseconds)
INI_GPRS_ALLOW_NO_SIM	False	NA	RW	Boolean	GP_ALLOW_ NO_SIM	When set to FALSE, DDI startup will fail when SIM is not present. When set to TRUE, DDI startup will succeed even when SIM is not present. DDI will go into no network state during this operation.

Table 14 GPRS DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GPRS_APN	None	1-64	RW	String	GP_APN	GPRS access point for SIM 1.
INI_GPRS_APN2	None	1-64	RW	String	GP_APN2	GPRS access point for SIM 2.
INI_GPRS_DSIM_PRI	1	1-2	RW	Integer	GP_DSIM_PRI	Primary SIM to use for initial connect. <ul style="list-style-type: none"> • 1: SIM in slot 1 • 2: SIM in slot 2
INI_GPRS_PRIMARY	None	4-128	RW	String	GP_PRIMARY	GPRS dial number
INI_GPRS_SIMID_CHK	False	NA	RW	Boolean	GP_SIMID_CHK	Enables/disables call of AT^SCID to check for SIM presence during DDI initialization. This needs to be set to TRUE for terminals that require a 10-second interval after radio power up before the SIM can be used.
INI_GPRS_MONITOR	0	NA	RW	Boolean	GP_MONITOR	If the driver needs to check GPRS availability before connection. <ul style="list-style-type: none"> • 0 (do not monitor GPRS) • 1 (monitor GPRS)
INI_GPRS_REGISTRATION	false	NA	RW	Boolean	GP_REG	This is set to true if the application wants to track network registration status and automatically restore connection upon re-registration when roaming.
INI_GPRS_REGISTRATION_TIMEOUT	0	0-180000	RW	Integer	GP_REG_TO	Duration that allows GPRS network registration status to remain unregistered, in milliseconds.

Table 14 GPRS DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GPRS_SIM_PIN	None	0-8	RW	String	GP_SIM_PIN	GPRS SIM PIN
INI_GPRS_SIM_PIN2	None	0-8	RW	String	GP_SIM_PIN2	GPRS SIM PIN2
INI_GPRS_SIM2_PIN	None	0-8	RW	String	GP_SIM2_PIN	GPRS SIM2 PIN
INI_GPRS_SIM2_PIN2	None	0-8	RW	String	GP_SIM2_ PIN2	GPRS SIM2 PIN2
INI_GPRS_SIM_PUK	None	0-8	RW	String	GP_SIM_PUK	GPRS SIM PUK
INI_GPRS_SIM_PUK2	None	0-8	RW	String	GP_SIM_PUK2	GPRS SIM PUK2
INI_GPRS_SIM2_PUK	None	0-8	RW	String	GP_SIM2_PUK	GPRS SIM2 PUK
INI_GPRS_SIM2_PUK2	None	0-8	RW	String	GP_SIM2_ PUK2	GPRS SIM2 PUK2
INI_HSPA_NETWORK	0	0-3	RW	Integer	HS_NETWORK	Network to select. Allowed values are: <ul style="list-style-type: none"> • 0: Use default radio settings • 1: Set radio to Automatic mode • 2: Set radio to 2G mode only • 3: Set radio to 3G mode only Note: In Automatic mode, the faster network which is 3G takes priority over 2G.
INI_GPRS_MIN_RSSI	10	0-100	RW	Integer	GP_MINRSSI	Minimum allowable RSSI percentage for the driver to connect
INI_GPRS_AT_SEND_TIMEOUT	20000	500-120000	RW	Integer	GP_SEND_TO	Timeout in milliseconds when sending an AT command to the modem
INI_GPRS_AT_RESPONSE_TIMEOUT	20000	500-120000	RW	Integer	GP_RESP_TO	Timeout in milliseconds when receiving a response from an AT command is sent to the modem

Table 14 GPRS DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GPRS_CONNECT_TIMEOUT	40000	500- 120000	RW	Integer	GP_CONN_TO	Data link connection timeout in milliseconds
INI_GPRS_NETWORK_TIMEOUT	20000	500- 120000	RW	Integer	GP_NET_TO	General network negotiation timeout applied when: <ul style="list-style-type: none"> • Querying for RSSI • Querying for GPRS availability
INI_GPRS_DISCONNECT_TIMEOUT	20000	500- 120000	RW	Integer	GP_DISC_TO	Data link disconnection timeout in milliseconds
INI_GPRS_INTERCHAR_TIMEOUT	2000	500- 120000	RW	Integer	GP_CHAR_TO	Intercharacter timeout in milliseconds
INI_GPRS_AT_DELAY	2000	250- 2000	RW	Integer	GP_ATDELAY	Delay in between retries in sending initialization strings to the modem (milliseconds)
INI_GPRS_ATTACH_TIMEOUT	300000	500- 300000	RW	Integer	GP_ATCH_TO	GPRS network attach timeout in milliseconds
INI_GPRS_INIT_RETRIES	3	1-9999	RW	Integer	GP_INIT_RT	Retries in sending the initialization strings in the modem
INI_GPRS_POWER_UP_RETRIES	3	1-9999	RW	Integer	GP_POWR_RT	Retries in powering up the GPRS modem
INI_GPRS_DETACH	false	NA	RW	Boolean	GP_DETACH	Set to true if the driver needs to detach before attaching to the network
INI_GPRS_SERVICE_NOTIFICATION	false	NA	RW	Boolean	GP_SERV_STS	This should be set to true before the driver is loaded if the application wants to get notified with network registration

Table 14 GPRS DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GPRS_BER_ NOTIFICATION	false	NA	RW	Boolean	GP_BER_STS	status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter. This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.
INI_GPRS_ROAM_ NOTIFICATION	false	NA	RW	Boolean	GP_ROAMSTS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.
INI_GPRS_SMS_ NOTIFICATION	false	NA	RW	Boolean	GP_SMS_STS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.

Table 14 GPRS DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GPRS_CALL_ NOTIFICATION	false	NA	RW	Boolean	GP_CALLSTS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.
INI_GPRS_SMS_FULL_ NOTIFICATION	false	NA	RW	Boolean	GP_SMSFSTS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.
INI_GPRS_START	"ATE0Q 0V0"	0 - 50	RW	String	GP_START	This is the start-up command sent to the modem. Warning: Changing this value may render the modem unusable.
INI_GPRS_INIT_1	"AT&D2 &S1&C 1\Q3X4"	0 - 50	RW	String	GP_INIT_1	This is the first initialization command sent to the modem. Warning: Changing this value may render the modem unusable.
INI_GPRS_INIT_2	None	0 - 50	RW	String	GP_INIT_2	This is the second initialization command sent to the modem.

Table 14 GPRS DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GPRS_INIT_3	None	0 - 50	RW	String	GP_INIT_3	This is the third initialization command sent to the modem.
INI_GPRS_INIT_4	None	0 - 50	RW	String	GP_INIT_4	This is the fourth initialization command sent to the modem.
INI_GPRS_RADIO_SHUTDOWN	True	NA	RW	Boolean	GP_SHUTDWN	This enables/disables sending of shutdown AT command to the radio modem.
INI_GPRS_WAIT_DCD_LOW_TIMEOUT	7500	500 - 120000	RW	Integer	GP_DCDLWTO	Time in milliseconds for the driver to wait for low DCD before starting force disconnection of the modem. This is done to give time for the implicit PDP context deactivation to take place properly and ultimately affect DCD to go low.

Table 15 lists the available IOCTL calls for mobile phone (GSM).

Table 15 IOCTL Calls Available for Mobile Phone (GSM)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_RSSI	None	NA	String Ex: 100 (means a signal strength of approximately -51 dBm)	Response is a percentage equivalent of the RSSI value retrieved from the modem.

Table 15 **IOCTL Calls Available for Mobile Phone (GSM)** (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_RSSI_ABS	None	NA	String Ex: 30 (means a signal strength of approximately -53 dBm)	Response is a range indicator of the RSSI signal, measured in dBm. Please see AT Commands document for the GSM/GPRS radio modem used.
IOCTL_GET_RSSI_DBM	None	NA	String Ex: -59 (means a signal strength percentage of 85%)	Response is a dBm approximate of the RSSI value retrieved from the modem.
IOCTL_GET_FW_VERSION	None	NA	String Ex: 1.03	Response is the firmware version of the GSM/GPRS radio modem used.
IOCTL_GET_FW_SUB_VERSION	None	NA	String Ex: "01"	Response is the sub firmware version of the GSM/GPRS radio modem used.
IOCTL_GET_CONNECTION_BAUD	None	NA	String Ex: 9600 (means the connection is running on 9600 bits per second)	Response is the connection baud rate established on the mobile phone (GSM) network
IOCTL_GET_BER	None	NA	String Ex: 99 (means an unknown bit error rate)	Response is the bit error rate associated to the RSSI
IOCTL_GET_FORBDN_LOCINFO	None	NA	String	Response is the Public Land Mobile Network (PLMN) location information.
IOCTL_GET_GPRS_LOCINFO	None	NA	String	Response is the GPRS location information.
IOCTL_GET_GSM_LOCINFO	None	NA	String	Response is the GSM location information.
IOCTL_GET_LINK_STATUS	None	NA	String Ex: 1 (means the data link is up)	Response can be either: <ul style="list-style-type: none"> • 1 (data link is up) • 0 (data link is not up)

Table 15 IOCTL Calls Available for Mobile Phone (GSM) (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_NET_AVAIL	None	NA	String	Response is the network availability status. Note: This is only supported in 2G (MC55i/BGS2).
IOCTL_GET_PDNET_REG	None	NA	String	Response is the packet domain network registration status.
IOCTL_GET_SERVCELL_INFO	None	NA	ddi_StCellInfo	Response is the information for the serving cell site.

Table 16 shows the parameters for mobile phone (GSM) DDI DLL configuration.

Table 16 Mobile Phone (GSM) DDI DLL Driver Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Attribute R= read only RW= read/write	Data Type	Config.sys	Description
INI_DRIVER_VERSION	Driver Version Number	NA	R	String	NA	Provides the DDI DLL version number
INI_SUPPORTS_AT	1	NA	R	Boolean	NA	If AT commands are supported by DDI DLL: <ul style="list-style-type: none"> • 1 (supported) • 0 (not supported)
INI_SUPPORTS_RSSI	1	NA	R	Boolean	NA	If RSSI values will be provided by DDI DLL <ul style="list-style-type: none"> • 1 (supported) • 0 (not supported)
INI_MAX_CONNECTION_RETRIES	5	1-9999	RW	Integer	CONN_RET	Maximum number of retries to dial to PPP server. If the dial response is BUSY, NO CARRIER or NO ANSWER, redialling to the PPP server will be repeated depending on the CONN_RET value.

Table 16 Mobile Phone (GSM) DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
						However, once a critical error is encountered such as NO LINE, NO DIAL TONE, ERROR, or connection timeout (no modem response), connecting to the PPP server will be aborted. Setting to 9999 disables the retry limit and causes the retry operation to be infinite.
INI_CONNECTION_RETRY_TIMEOUT	12000	500-120000	RW	Integer	CONNRET_TM	The time interval between connection retries (in milliseconds)
INI_GSM_PRIMARY	None	4-128	RW	String	GS_PRIMARY	Mobile phone (GSM) dial number
INI_GSM_SIMID_CHK	False	NA	RW	Boolean	GS_SIMID_CHK	Enables/disables call of AT^SCID to check for SIM presence during DDI initialization. This needs to be set to TRUE for terminals that require a 10-second interval after radio power up before the SIM can be used.
INI_GSM_SIM_PIN	None	0-5	RW	String	GS_SIM_PIN	Mobile phone (GSM) SIM PIN
INI_GSM_SIM_PUK	None	0-5	RW	String	GS_SIM_PUK	Mobile phone (GSM) SIM PUK
INI_GSM_SIM_PIN2	None	0-8	RW	String	GS_SIM_PIN2	GSM SIM PIN2

Table 16 Mobile Phone (GSM) DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GSM_SIM2_PIN	None	0-8	RW	String	GS_SIM2_ PIN	GSM SIM2 PIN
INI_GSM_SIM2_PIN2	None	0-8	RW	String	GS_SIM2_ PIN2	GSM SIM2 PIN2
INI_GSM_SIM_PUK2	None	0-8	RW	String	GS_SIM_ PUK2	GSM SIM PUK2
INI_GSM_SIM2_PUK	None	0-8	RW	String	GS_SIM2_ PUK	GSM SIM2 PUK
INI_GSM_SIM2_PUK2	None	0-8	RW	String	GS_SIM2_ PUK2	GSM SIM2 PUK2
INI_GSM_MINRSSI	10	0-100	RW	Integer	GS_MINRSSI	Minimum allowable RSSI percentage for the driver to connect. RSSI percentage value lower than the specified MINRSSI indicates poor network state.
INI_GSM_AT_SEND_ TIMEOUT	20000	500- 120000	RW	Integer	GS_SEND_TO	Timeout in milliseconds when sending an AT command to the modem
INI_GSM_AT_RESPONSE_ TIMEOUT	20000	500- 120000	RW	Integer	GS_RESP_TO	Timeout in milliseconds when receiving a response from an AT command is sent to the modem
INI_GSM_CONNECT_ TIMEOUT	60000	500- 120000	RW	Integer	GS_CONN_TO	Data link connection timeout in milliseconds
INI_GSM_NETWORK_ TIMEOUT	20000	500- 120000	RW	Integer	GS_NET_TO	General network negotiation timeout applied when querying for RSSI
INI_GSM_DISCONNECT_ TIMEOUT	20000	500- 120000	RW	Integer	GS_DISC_TO	Data link disconnection timeout in milliseconds

Table 16 Mobile Phone (GSM) DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GSM_INTERCHAR_TIMEOUT	2000	500-120000	RW	Integer	GS_CHAR_TO	Interchar timeout in milliseconds
INI_GSM_AT_DELAY	2000	250-2000	RW	Integer	GS_ATDELAY	Delay in between retries in sending initialization strings to the modem (milliseconds)
INI_GSM_INIT_RETRIES	3	1-9999	RW	Integer	GS_INIT_RT	Retries in sending the initialization strings in the modem
INI_GSM_POWER_UP_RETRIES	3	1-9999	RW	Integer	GS_POWR_RT	Retries in powering up the mobile phone (GSM) modem
INI_GSM_SERVICE_NOTIFICATION	false	NA	RW	Boolean	GS_SERV_STS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.
INI_GSM_BER_NOTIFICATION	false	NA	RW	Boolean	GS_BER_STS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.

Table 16 **Mobile Phone (GSM) DDI DLL Driver Configuration Parameters** (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GSM_ROAM_ NOTIFICATION	false	NA	RW	Boolean	GS_ROAMSTS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.
INI_GSM_SMS_ NOTIFICATION	false	NA	RW	Boolean	GS_SMS_STS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.
INI_GSM_CALL_ NOTIFICATION	false	NA	RW	Boolean	GS_CALLSTS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.

Table 16 Mobile Phone (GSM) DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GSM_SMS_FULL_NOTIFICATION	false	NA	RW	Boolean	GS_SMSFSTS	This should be set to true before the driver is loaded if the application wants to get notified with network registration status. This configuration can be modified at run-time using its corresponding "set" IOCTL parameter.
INI_GSM_START	"ATE0Q0V0"	0 - 50	RW	String	GS_START	This is the start-up command sent to the modem. Warning: Changing this value may render the modem unusable.
INI_GSM_INIT_1	"AT+CBST=7,0,1"	0 - 50	RW	String	GS_INIT_1	This is the first initialization command sent to the modem. Warning: Changing this value may render the modem unusable.
INI_GSM_INIT_2	"AT&D2&S1&C1\Q3X4"	0 - 50	RW	String	GS_INIT_2	This is the second initialization command sent to the modem. Warning: Changing this value may render the modem unusable.
INI_GSM_INIT_3	None	0 - 50	RW	String	GS_INIT_3	This is the third initialization command sent to the modem.

Table 16 Mobile Phone (GSM) DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_GSM_INIT_4	None	0 - 50	RW	String	GS_INIT_4	This is the fourth initialization command sent to the modem.
INI_GSM_RADIO_SHUTDOWN	True	NA	RW	Boolean	GS_SHUTDWN	This enables/disables sending of shutdown AT command to the radio modem.
INI_GSM_BEARER_SVC_TYPE	None	0 - 64	RW	String	GS_BEARER_SVC_TYPE	Parameter for the AT+CBST command to set the bearer service type. Value should follow the format "<speed> , <name> , <ce>".

NOTE



The INI and IOCTL parameters are defined in `vos_ddi_ini.h` and `vos_ioctl.h` respectively. Applications using these parameters should include and use macros defined in the said header files.

Table 17 lists the available CDMA IOCTL calls.

Table 17 IOCTL Calls Available for CDMA

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_RSSI	None	NA	Integer value	RSSI format as currently defined for Wireless DDI drivers.
IOCTL_GET_CALL_STATUS	None	NA	Integer value CONNECTED 103 DISCONNECTED 104 DORMANT 132	Provides the Call status.
IOCTL_GET_ROAM_STATUS	None	NA	Integer	Provides the current roam status
IOCTL_GET_ACTIVATION_STATUS	None	NA	Integer value (HEX format)	Provides the activation status of modem.
IOCTL_GET_RSSI	None	NA	None	Provides the RSSI of the current mode of CDMA

Table 17 **IOCTL Calls Available for CDMA**

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_PRL_VERSION	None	NA	None	Provides the PRL version
IOCTL_GET_COVERAGE_STATUS	None	NA	Integer value (HEX format)	Provides the coverage status
IOCTL_GET_HDR_SERVICE_STATE	None	NA	None	Provides the HDR service state
IOCTL_GET_ROAM_STATUS	None	NA	Integer value (HEX format)	Provides the CDMA 1XRTT ROAM status
IOCTL_GET_FW_VERSION	None	NA	None	Provides the Firmware version
IOCTL_GET_ELECTRONIC_SRL_NO	None	NA	Integer value (HEX format)	Provides the Electronic serial number
IOCTL_GET_MIN_NUMBER	None	NA	Integer value	Provides the MIN number
IOCTL_DISABLE_ALL_NOTIFY	None	NA	Result of operation: <ul style="list-style-type: none"> • 0 - Success • > 0 - Operation Failure 	Disables all the unsolicited notifications.
IOCTL_ENABLE_ALL_NOTIFY	None	NA	Result of operation: <ul style="list-style-type: none"> • 0 - Success • > 0 - Operation Failure 	Enables all the unsolicited notifications.
IOCTL_SET_ROAM_NOTIFICATION	Pointer to a string that contains either 1 or 0	String	None	Sets roaming status notification on or off. Valid values are: <ul style="list-style-type: none"> • 1 - Turn on notifications • 0 - Turn off notifications

Table 18 lists the parameters for CDMA DDI DLL driver configuration.

Table 18 CDMA DDI DLL Driver Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Attribute R= read only RW= read/write	Data Type	Config.sys	Description
CONN_RET	5	1 - 9999	RW	Integer	CONN_RET	Maximum number of retries to dial to CDMA service. If the dial response is BUSY, NO CARRIER, or NO ANSWER, redialling to the CDMA service will be repeated depending on the CONN_RET value. However, once a critical error is encountered such as NO LINE, NO DIAL TONE, ERROR or connection timeout (no modem response), connecting to the CDMA service will be aborted. Setting to 9999 disables the retry limit and causes the retry operation to be infinite.
CONNRET_TM	12000	500 - 120000	RW	Integer	CONNRET_TM	Time interval between connection retries (in milliseconds).
EV_SEND_TO	20000	500 - 120000	RW	Integer	EV_SEND_TO	Timeout in milliseconds when sending an AT command to the modem
EV_RESP_TO	20000	500 - 120000	RW	Integer	EV_RESP_TO	Response timeout after sending an AT command (ms)
EV_CONN_TO	40000	500 - 120000	RW	Integer	EV_CONN_TO	Data connection timeout (ms).
EV_DISC_TO	20000	500 - 120000	RW	Integer	EV_DISC_TO	Data disconnection timeout (ms)

Table 18 CDMA DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
EV_CHAR_TO	200	500 - 120000	RW	Integer	EV_CHAR_TO	Using a single intercharacter timeout for all AT commands (ms)
EV_ATDELAY	150	100 - 2000	RW	Integer	EV_ATDELAY	Number of milliseconds to wait between sending AT commands
EV_NET_TO	20000	500 - 120000	RW	Integer	EV_NET_TO	EVDO network transaction timeout
EV_INIT_RT	3	1 - 9999	RW	Integer	EV_INIT_RT	The number of retries spent in trying "EVDO_start"
EV_POWR_RT	3	1 - 9999	RW	Integer	EV_POWR_RT	The number of attempts the driver would check if the modem responds during power up
EV_PRIMARY	"#777"	1 - 64	RW	String	EV_PRIMARY	Primary number to dial
EV_MONITOR	FALSE	0 - 1	RW	Boolean	EV_MONITOR	EVDO monitor switch
EV_SIM_PIN	None	0 - 5	RW	Integer	EV_SIM_PIN	SIM Pin
EV_NETMODE	8	2 - CDMA Mode 4 - HDR Mode 8 - CDMA/ HDR Hybrid mode	RW	Integer	EV_NETMODE	Network Mode
EV_MINRSSI	20	2 - CDMA Mode	RW	Integer	EV_MINRSSI	Minimum percentage of signal strength
EV_SERVSTS	FALSE	0 or 1	RW	Boolean	EV_SERVSTS	Toggle to enable/disable application to get network registration notification
EV_ROAMSTS	FALSE	0 or 1	RW	Boolean	EV_ROAMSTS	Toggle to enable/disable application to get roaming status

Table 18 CDMA DDI DLL Driver Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Attribute R= read only RW= read/write	Data Type	Config.sys	Description
EV_DORMSTS	FALSE	0 or 1	RW	Boolean	EV_DORMSTS	Toggle to enable/disable application to get dormancy status
EV_CALLSTS	FALSE	0 or 1	RW	Boolean	EV_CALLSTS	Toggle to enable/disable application notification to get call status
EV_COVRSTS	FALSE	0 or 1	RW	Boolean	EV_COVRSTS	Toggle to enable/disable application notification to get coverage status
EV_START	"ATE1V1"	0 - 50	RW	String	EV_START	AT Command Start up string
EV_SHUTDOWN	FALSE	0 or 1	RW	Boolean	EV_SHUTDOWN	Do shutdown mechanism before closing port
EV_LOWBATT	10	10 to 100	RW	Integer	EV_LOWBATT	Minimum battery charge percentage
EV_SYSINFO	1000	500 to 120000	RW	Integer	EV_SYSINFO	Delay time between two subsequent AT^SYSINFO commands. Value is in milliseconds (ms).

Table 19 lists the available IOCTL calls for Bluetooth.

Table 19 IOCTL Calls Available for Bluetooth

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_SEARCH	None	NA	String	Initiate a search for Bluetooth devices. Response is "OK" when search is successfully initiated. Operation status and results are provided via EOS application events.
IOCTL_GET_SEARCH_CNT	None	NA	String	Response is the number of Bluetooth devices found after a search operation.

Table 19 **IOCTL Calls Available for Bluetooth** (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_SEARCH_RESULT	None	NA	ddi_bt_device *	Response is a list of Bluetooth devices found after a search operation.
IOCTL_BT_PAIR	Pairing information	ddi_bt_pair *	NA	Pair with a Bluetooth device that was found after a search operation.
IOCTL_GET_BT_PAIR_CNT	None	NA	String	Response is the number of paired Bluetooth devices.
IOCTL_GET_BT_PAIRIED_DEVICES	None	NA	ddi_bt_device *	Response is a list of paired Bluetooth devices.
IOCTL_GET_PAN_DEVICE	None	NA	ddi_bt_device*	Response is the connection information for the PAN port.
IOCTL_GET_DUN_DEVICE	None	NA	ddi_bt_device*	Response is the connection information for the DUN port.
IOCTL_GET_SPP_DEVICE	None	NA	ddi_bt_device*	Response is the connection information for the SPP port.
IOCTL_SET_CONNECT_INFO	Connection information	ddi_bt_device *	NA	Set connection information for a Bluetooth device port. Information is set by assigning the Bluetooth device port to a paired Bluetooth device.
IOCTL_EX_BT_REMOVE_PAIR	Paired Bluetooth device information	ddi_bt_device *	NA	Remove pairing with a paired Bluetooth device.
IOCTL_EX_BTBASE_FW_UP	Firmware upgrade information	ddi_bt_base_file_info	NA	Initiates Bluetooth base firmware upgrade. Response is "OK" when firmware upgrade is successfully initiated. Operation status and results are provided via EOS application events. Note: Applicable to VX 680 BT AP only.

Table 19 **IOCTL Calls Available for Bluetooth** (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_EX_SELECT_REMOTE_BT_AP	Bluetooth address	String	NA	Sets the Bluetooth base to be used for the IOCTL_GET_BTBASE_INFO operation. Note: Applicable to VX 680 BT AP only.
IOCTL_GET_BTBASE_INFO	None	NA	ddi_bt_base_info	Retrieves the following information from the Bluetooth base: <ul style="list-style-type: none"> • Firmware version • PAN connection type • Ethernet cable state • Router IP address • Serial number Note: Applicable to VX 680 BT AP only.
IOCTL_BTBASE_RESTORE_FACTORY	Bluetooth address	String	NA	Restores the Bluetooth base to factory settings. Response is “OK” when the factory restore is successfully initiated. Operation status and results are provided via EOS application events. Note: Applicable to VX 680 BT AP only.
IOCTL_GET_BT_LOCAL_ADDRESS	None	NA	String	Response is the MAC address of the local Bluetooth device.
IOCTL_GET_FW_VERSION	None	NA	String	Response is the version of the OS Bluetooth Driver.

NOTE



There are no signal strength ranges for IOCTL Calls for Bluetooth.

Table 20 lists the ddi_bt_device parameters.

Table 20 ddi_bt_device Member Variables

Parameter	Data Type	Description
bd_addr	Char[16]	String value of the Bluetooth address.
bt_class	Char[8]	String value of the Bluetooth class of device (CoD). The supported Bluetooth device ports of the remote device can be determined by doing bit checks of the CoD. The value can be equated to an integer variable and compared with the bits below to check the supported ports: <ul style="list-style-type: none"> • 0x020000 – PAN capable. /DEV/BT_PAN is supported. • 0x400000 – DUN capable. /DEV/BT_MODEM is supported. • 0x001F00 – Unclassified. /DEV/BT_SERIAL is supported. This is the CoD of the Dione Base.
name	Char[64]	String value of the friendly name.
port_bt	Char[16]	String value of the Bluetooth device port for IOCTL_SET_CONNECT_INFO call. The following values are allowed: <ul style="list-style-type: none"> • /DEV/BT_PAN • /DEV/BT_MODEM • /DEV/BT_SERIAL • ALL_PORTS The IOCTL_SET_CONNECT_INFO will fail when the port set is not supported by the device. When connecting to a Dione base, the required port setting is /DEV/BT_SERIAL for both Ethernet and Dial.
port_bt_bit	Integer	Bit setting of Bluetooth device ports for IOCTL_GET_BT_PAIRIED_DEVICES call. The following bits are possible: <ul style="list-style-type: none"> • BT_PAN_PORT_BIT • BT_MDM_PORT_BIT • BT_SPPI_PORT_BIT • BT_OBEX_PORT_BIT

Table 21 lists the ddi_bt_pair parameters.

Table 21 ddi_bt_pair Member Variables

Parameter	Data Type	Description
bd_addr	Char[16]	String value of the Bluetooth address.
pin	Char[16]	(Optional) String value of the pin. Set this to empty when pairing with a VX 680 BT base.
name	Char[64]	Friendly name.

Table 22 lists the ddi_bt_base_file_info parameters.

Table 22 ddi_bt_base_file_info Member Variables

Parameter	Data Type	Description
bd_addr	Char[16]	String value of the Bluetooth address.
filename	Char[40]	Filename of the firmware file.
gid	Int	GID location of the firmware file.

Table 23 lists the ddi_bt_base_info parameters.

Table 23 ddi_bt_base_info Member Variables

Parameter	Data Type	Description
fw_ver	Char[16]	Firmware version following the format "xx.xx.xx.xx".
pan_type	Int	PAN connection type. The following values are returned: <ul style="list-style-type: none"> • 1: Bridge • 2: Router • -1: Unknown (Error encountered in Bluetooth base during query of this information)
eth_cable_stat	Int	Ethernet cable state. The following values are returned: <ul style="list-style-type: none"> • 1: Connected • 0: Not connected
ip_addr	Char[16]	Router's IPv4 address.
serial_no	Char[16]	Serial number following the format "xxx-xxx-xxx".

Table 24 lists the Bluetooth DDI DLM driver configuration parameters.

Table 24 Bluetooth DDI DLM Driver Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Attribute R= read only RW= read/write	Data Type	Config.sys	Description
INI_DRIVER_VERSION	Driver version number	NA	R	String	NA	DDI DLM version number.
INI_SUPPORTS_AT	0	NA	R	Boolean	NA	AT commands are supported by DDI DLM.
INI_SUPPORTS_RSSI	0	NA	R	Boolean	NA	RSSI will be provided by DDI DLM.
INI_VFI_BASE_CLASS	520300	NA	RW	String	BT_VFI_BASE_CLASS	Device class of VeriFone base stations.
INI_SRCH_VFI_BASE	0	NA	RW	Boolean	BT_SRCH_VFI_BASE	Filter search for Bluetooth devices to VeriFone base stations only.
INI_SRCH_MAX	8	1-16	RW	Integer	BT_SRCH_MAX	Maximum number of Bluetooth devices to search.

Table 24 Bluetooth DDI DLM Driver Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_SRCH_TIMEOUT	10000	3000- 120000	RW	Integer	BT_SRCH_ TIMEOUT	Timeout in milliseconds when performing a search for Bluetooth devices.
INI_PAIR_MAX	8	1-8	RW	Integer	BT_PAIR_ MAX	Maximum number of paired Bluetooth devices.
INI_BT_ENABLE_SSP	1	NA	RW	Boolean	BT_ ENABLE_ SSP	Enable Secure Simple Pairing.

Table 25 lists the Dial over Bluetooth configuration parameters.

Table 25 Dial over Bluetooth Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_NO_NET_RET_TM	300000	30000- 7200000	RW	Number	BTDL_NO_NET_ RET_TM	Duration in milliseconds that reconnection is attempted when the Bluetooth link is disconnected.
INI_LD_MDM_PROF	1	NA	RW	Boolean	BTDL_LD_MDM_ PROF	Enables or disables loading of modem profile from terminal. The following values are allowed: <ul style="list-style-type: none"> • 1 - Enable • 0 - Disable

Please refer to [Table 9](#) for the rest of the configuration parameters. The regular Dial and Dial over Bluetooth uses the same dial parameters with difference only in the `config.sys`. The Dial over Bluetooth will have "BT" prefixed in the regular Dial `config.sys` parameters, e.g. `DL_PHO_PRI` for regular Dial is `BTDL_PHO_PRI` for Dial over Bluetooth.

Table 26 lists the Ethernet over Bluetooth configuration parameters.

Table 26 Ethernet over Bluetooth Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_NO_NET_RET_TM	300000	30000- 7200000	RW	Number	BTET_NO_NET_ RET_TM	Duration in milliseconds that reconnection is attempted when the Bluetooth link is disconnected.

Table 27 lists the mobile phone (GSM) over Bluetooth configuration parameters.

Table 27 Mobile Phone (GSM) over Bluetooth Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_MAX_CONNECTION_ RETRIES	1	1-10	RW	Number	BTMP_CONN_ RET	Maximum connection retries. The driver first dials using the DL_PHO_PRI (primary phone number). If the dial response is BUSY, NO CARRIER or NO ANSWER, it uses the DL_PHO_ SEC (secondary phone number) if provided. If no DL_PHO_SEC is provided, it would attempt a connection retry using again the primary number until CONN_RET value is reached.

Table 27 **Mobile Phone (GSM) over Bluetooth Configuration Parameters** (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
						<p>If DL_PHO_SEC is provided and it still encounters BUSY, NO CARRIER or NO ANSWER, it would attempt a connection retry (call the primary number then secondary if necessary) depending on the CONN_RET value.</p> <p>But, once a critical error is encountered such as NO LINE, NO DIAL TONE, ERROR or connection timeout, dial call is aborted be it using a primary or a secondary phone number.</p>
INI_CONNECTION_RETRY_TIMEOUT	1000	500-60000	RW	Number	BTMP_CONNRET_TM	Connection retry timeout in milliseconds
INI_NO_NETWORK_RETRIES	10	1-1000	RW	Number	BTMP_NO_NET_RET	The number of retries when Bluetooth link is disconnected.
INI_MODEM_START	ATE0	2-50	RW	String	BTMP_START	Start up string
INI_MODEM_DIAL	ATD	2-50	RW	String	BTMP_DIAL	<p>AT command string to dial.</p> <p>Be cautious in editing changing the DL_DIAL parameter value as it may affect the dialing behavior.</p>

Table 27 **Mobile Phone (GSM) over Bluetooth Configuration Parameters** (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
						The Dial DDI Driver sends the dial string sequence as: <DL_DIAL> <DL_DIALTYPE> <DL_PHO_PRI> or <DL_DIAL> <DL_DIALTYPE> <DL_PHO_SEC>. Refer to the corresponding AT Command Reference for more information.
INI_MODEM_INIT_RETRIES	10	1-10	RW	Number	BTMP_INITRET	Number of times in sending the start up string until an "OK" is returned.
INI_MODEM_AT_DELAY	250	250-1000	RW	Number	BTMP_ATDELAY	Number of milliseconds to wait between sending AT commands.
INI_MODEM_INIT_1	NONE	2-50	RW	String	BTMP_INIT_1	Init String 1
INI_MODEM_INIT_2	NONE	2-50	RW	String	BTMP_INIT_2	Init String 2
INI_MODEM_INIT_3	NONE	2-50	RW	String	BTMP_INIT_3	Init String 3
INI_MODEM_INIT_4	NONE	2-50	RW	String	BTMP_INIT_4	Init String 4
INI_MODEM_INIT_5	NONE	2-50	RW	String	BTMP_INIT_5	Init String 5
INI_MODEM_INIT_6	NONE	2-50	RW	String	BTMP_INIT_6	Init String 6
INI_MODEM_INIT_7	NONE	2-50	RW	String	BTMP_INIT_7	Init String 7
INI_MODEM_INIT_8	NONE	2-50	RW	String	BTMP_INIT_8	Init String 8
INI_MODEM_INIT_9	NONE	2-50	RW	String	BTMP_INIT_9	Init String 9
INI_MODEM_INIT_10	NONE	2-50	RW	String	BTMP_INIT_10	Init String 10
INI_MODEM_BAUD	9	0-13	RW	Number	BTMP_BAUD	Baud rate. Baud rate values numeric representation: 0 = 300bps 1 = 600bps 2 = 1200bps 3 = 2400bps

Table 27 Mobile Phone (GSM) over Bluetooth Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
						4 = 4800bps 5 = 9600bps 6 = 19200bps 7 = 38400bps 8 = 57600bps 9 = 115200bps 10 = 12000bps (not supported in Conexant and Silabs DDI Drivers) 11 = 14400bps 12 = 28800bps (for Conexant DDI Driver only) 13 = 33600bps (not supported in Conexant and Silabs DDI Drivers)
INI_MODEM_FORMAT	4	0-5	RW	Number	BTMP_FORMAT	Format. Format numeric representation: 0 = Fmt_A7E1 1 = Fmt_A7N1 (for Silabs only) 2 = Fmt_A7O1 3 = Fmt_A8E1 (for Silabs only) 4 = Fmt_A8N1 5 = Fmt_A8O1 (for Silabs only)
INI_MODEM_AT_SEND_TIMEOUT	20000	500-60000	RW	Number	BTMP_ATSNDTO	Timeout to send the AT command
INI_MODEM_INIT_TIMEOUT	20000	500-60000	RW	Number	BTMP_INIT_TO	Response timeout used for all init AT commands in milliseconds.
INI_MODEM_CONNECT_TIMEOUT	60000	45000-120000	RW	Number	BTMP_CONN_TO	Data connection timeout in milliseconds
INI_MODEM_DISCONNECT_TIMEOUT	1400	500-60000	RW	Number	BTMP_DISC_TO	Data disconnect timeout in milliseconds

Table 27 Mobile Phone (GSM) over Bluetooth Configuration Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_MODEM_INTERCHAR_ TIMEOUT	500	500- 60000	RW	Number	BTMP_ICHAR_ TO	Inter-character timeout for all AT commands in milliseconds
INI_DIAL_PRIMARY	NONE	RW	1-128	String	BTMP_PHO_ PRI	Primary phone number to dial
INI_DIAL_SECONDARY	NONE	RW	1-128	String	BTMP_PHO_ SEC	Secondary phone number to dial

Table 28 lists the available IOCTL calls for WiFi.

Table 28 IOCTL Calls Available for WiFi

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_SEARCH	None	NA	String	Initiates a scan of WiFi networks at all channels. Response is "OK" when search is successfully initiated.
IOCTL_GET_SEARCH_CNT	None	NA	String	Response is the number of WiFi networks found after a scan operation.
IOCTL_GET_SEARCH_RESULT	None	NA	ddi_bss_info *	Response is a list of WiFi networks found after a scan operation. The WiFi network list displays up to 25 scanned networks only. The Application must call the IOCTL multiple times to receive the complete list of scanned WiFi networks. When all WiFi networks are displayed, the next IOCTL call returns -1047.

Table 28 **IOCTL Calls Available for WiFi** (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_ADD_NW_PROFILE	None	NA	String	<p>Add a new network profile with empty parameters. Response is the network ID of the new network profile.</p> <p>Note: EOS memory can handle up to 10 network profiles. System crash can occur if more than 10 profiles are present.</p>
IOCTL_SET_NW_PROFILE	Network profile information	ddi_nw_profile *	NA	Set the parameters for a network profile. Selection of network profile to set is based on the network ID.
IOCTL_SET_NW_ID	Network ID	String	NA	Select the network profile to use for IOCTL_GET_NW_PROFILE operation.
IOCTL_GET_NW_PROFILE	None	NA	ddi_nw_profile *	Response is the network parameters for a network profile.
IOCTL_GET_NW_CNT	None	NA	String	Response is the number of network profiles.
IOCTL_GET_NW_PROFILE_LIST	None	NA	ddi_nw_list *	Response is a list of network profiles with some parameter information.
IOCTL_DEL_NW_PROFILE	Network ID	String	NA	Remove a network profile.
IOCTL_SET_DEFAULT_PROFILE	Network ID	String	NA	Set the network profile to use when the WiFi network interface is started.
IOCTL_GET_DEFAULT_PROFILE	None	NA	String	Response is the network ID of the network profile that is used when the WiFi network interface is started.
IOCTL_GET_RSSI	None	NA	String Ex: 100 (means a signal strength of approximately -60 dBm)	Response is a percentage equivalent of the RSSI value retrieved from the WiFi module.

Table 28 IOCTL Calls Available for WiFi (continued)

Param Name (Type:macro)	Data Accepted	Input Data Type	Output Parameter Data Type	Description
IOCTL_GET_RSSI_ABS			String Ex: 32 (means a signal strength of approximately -60 dBm)	Response is a range indicator of the RSSI, measured in dBm.
IOCTL_GET_RSSI_DBM			String Ex: -65 (means a signal strength percentage of 85%)	Response is a dBm approximate of the RSSI value retrieved from the WiFi module.
IOCTL_GET_BSSINFO	None	NA	ddi_bss_info	Response is the network parameters for the currently connected network profile.
IOCTL_GET_AP_MAC	None	NA	String	Response is the MAC address of the connected access point.
IOCTL_GET_FW_VERSION	None	NA	String	Response is the firmware version of the WiFi driver.
IOCTL_GET_MAC	None	NA	String	Response is the MAC address of the terminal's WiFi module.

Table 29 lists the ddi_bss_info parameters.

Table 29 ddi_bss_info Member Variables

Parameter	Data Type	Description
BSSID	Unsigned Char[6]	12 octet value of the basic service set identifier (BSSID). In an infrastructure network, this is the access point's MAC address. In an ad-hoc network, this is a locally administered IEEE MAC address generated from a 46-bit random number.
SSID	Char[34]	String value of the network name (SSID).
freq	Integer	Holds the channel in frequency. This frequency value will be converted in channel internally. Example: Channel: 1 corresponds to Frequency: 2412
RSSI	Integer	Contains the approximate RSSI level returned by the WiFi module in dBm. Application can equate the value to a signed char variable to get the actual negative value. A value of -60 and higher indicates 100% signal level, while a value of -92 and lower indicates 1% signal level.

Table 29 **ddi_bss_info Member Variables** (continued)

Parameter	Data Type	Description
NWAuth	Char[14]	String value of the supported authentication type. The following values are possible: <ul style="list-style-type: none"> • WEP • WPA-PSK or WPA2-PSK • WPA-EAP or WPA2-EAP • WPA-CCKM or WPA2-CCKM • WPA-None String is empty when authentication is disabled.
DataEncrypt	Char[10]	String value of the supported encryption type. The following values are possible: <ul style="list-style-type: none"> • WEP • TKIP • CCMP String is empty when encryption is disabled.
bss_type	Integer	Network type whether infrastructure or ad-hoc. The following values are possible: <ul style="list-style-type: none"> • INFRASTRUCTURE(1) • ADHOC(2)
SSID_len	Integer	Length of the SSID string.

Table 30 lists the `ddi_nw_profile` parameters.

Table 30 **ddi_nw_profile Member Variables**

Parameter	Data Type	Description
network_id	Integer	Unique numeric ID of the profile in the profile list.
BSSID	Unsigned Char[8]	Contains BSSID information for internal driver processing. Values are ignored during <code>IOCTL_SET_NW_PROFILE</code> and unset during <code>IOCTL_GET_NW_PROFILE</code> .
ssid	Char[34]	String value of the network name (SSID).
flags	Char[64]	Contains the flags needed for internal driver processing. Flag values are processed bitwise.
ssid_len	Integer	Length of the SSID string.
bssid_len	Integer	Length of the BSSID string.
auth_type	Integer	Contains the numeric value for the network authentication type. This ranges from but not limited to: <ul style="list-style-type: none"> • Open (No authentication) • WEP/WEP-shared • WPA-PSK/WPA-EAP/WPA-CCKM • WPA2-PSK/WPA2-EAP/WPA2-CCKM For the complete list of possible values, an enum is provided in <code>vos_ioctl.h</code> .

Table 30 **ddi_nw_profile Member Variables** (continued)

Parameter	Data Type	Description
wsec	Integer	Contains the numeric value for network security type. These are the possible values: <ul style="list-style-type: none"> • Off • WEP • TKIP • WEP128 • AES-CCM For the complete list of possible values, an enum is provided in <code>vos_ioctl.h</code> .
bss_type	Integer	Determines whether network is infrastructure or ad-hoc. The following values are possible: <ul style="list-style-type: none"> • INFRASTRUCTURE(1) • ADHOC(2)
channel	Integer	Contains the channel ranging from 1 - 14.
uKey	Union	Holds the each network authentication type's details such as keys and passwords. One authentication type is only used at a time.
wpa_auth	Struct	Structure used for EAP.
eap_type	Integer	Contains the numeric value for the EAP type. This ranges from but not limited to: <ul style="list-style-type: none"> • No EAP • PEAP • TLS • FAST For the complete list of possible values, an enum is provided in <code>vos_ioctl.h</code> .
password	Char[128]	String value holding the password needed for EAP authentication.
identity	Char[128]	String value holding the identity needed for EAP authentication.
ca_cert	Char[128]	String value holding the file path to the CA certificate file.
client_cert	Char[128]	String value holding the file path to client certificate file.
private_key	Char[128]	String value holding the file path to private certificate file.
private_key_passwd	Char[128]	String value holding the password for the private key.
wep_auth	Struct	Structure used for WEP.
key	Char[4][36]	Contains the four WEP key passphrase.
key_len	Integer	Length of the current WEP key indicated by index.
index	Integer	Contains the current WEP index.
wpa_psk	Struct	Structure used for PSK.
key	Char[64]	Contains the PSK password.
key_len	Integer	Length of the PSK password.

Table 31 lists the `ddi_nw_list` parameters.

Table 31 **ddi_nw_list Member Variables**

Parameter	Data Type	Description
network_id	Integer	Unique numeric ID of the profile in the profile list.
SSID	Char[34]	String value of the network name (SSID).

Table 31 ddi_nw_list Member Variables

Parameter	Data Type	Description
BSSID	Char[6]	12 octet value of the basic service set identifier (BSSID). In an infrastructure network, this is the access point's MAC address. In an ad-hoc network, this is a locally administered IEEE MAC address generated from a 46-bit random number.
flags	Char[64]	Contains the flags needed for internal driver processing. Flag values are processed bitwise.
bssid_len	Integer	Length of the BSSID string.
ssid_len	Integer	Length of the SSID string.

Table 32 lists the parameters for WiFi DDI DLM driver configuration.

Table 32 WiFi DDI DLM Driver Configuration Parameters

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_DRIVER_VERSION	Driver version number	NA	R	String	NA	DDI DLM version number.
INI_SUPPORTS_AT	0	NA	R	Boolean	NA	AT commands are supported by DDI DLM.
INI_SUPPORTS_RSSI	1	NA	R	Boolean	NA	RSSI will be provided by DDI DLM.
INI_WI_PM	2	0 - 2	RW	Integer	WI_PM	Power saving mode. 0 (PM_OFF): driver never goes to sleep mode. 1 (PM_MAX): driver always goes to sleep mode. 2 (PM_FAST): driver goes to sleep mode when there are no active data transfers.
INI_WI_80211N	TRUE	0 or 1	RW	Boolean	WI_80211N	Enable 802.11N
INI_WI_KEEP_ALIVE_TO	55000	500-120000	RW	Integer	WI_KEEP_ALIVE_TO	Time interval in milliseconds the terminal sends keep-alive packets to the AP.
INI_WI_ROAM	0	NA	RW	Boolean	WI_ROAM	Enable roaming in the device.
INI_WI_ROAM_MINRSSI	10	0-100	RW	Integer	WI_ROAM_MINRSSI	Percent RSSI value to trigger roaming.

Table 32 **WiFi DDI DLM Driver Configuration Parameters** (continued)

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Attribute R= read only RW= read/ write	Data Type	Config.sys	Description
INI_WI_BEACON_TM	60000	500- 120000	RW	Integer	WI_BEACON_ TM	Time interval in milliseconds after beacons are lost to report link is down.
INI_WI_CONN_TO	60000	500- 120000	RW	Integer	WI_CONN_TO	Timeout in milliseconds for a data link connection attempt.
INI_WI_DISC_TO	60000	500- 12000	RW	Integer	WI_DISC_TO	Timeout in milliseconds for the data link disconnection attempt.
INI_WI_ROAM_SCAN_PERIOD	5000	1000- 120000	RW	Integer	WI_ROAM_ SCAN_ PERIOD	Timeout in milliseconds after association that background scan for roaming is started.
INIT_WI_ROAM_RENEW_IP	0	NA	RW	Boolean	WI_ROAM_ RENEW_IP	Enable renewal of IP address when terminal has roamed to another AP.
INI_WI_MINRSSI	10	1 - 99	RW	Integer	WI_MINRSSI	RSSI level in percentage that EOS shifts from good network to poor network conditions.
INI_WI_RSSI_STS_THLD	5	1 - 20	RW	Integer	WI_RSSI_ STS_THLD	RSSI threshold in dBm with reference from the last RSSI notification that a new RSSI notification is sent.

Bluetooth and WiFi Configuration Files

This section describes the standardized Bluetooth and WiFi configuration file format to use across all Verix eVo components requiring external configuration/customization.

Bluetooth Configuration File

Bluetooth NWIF users have the option to use the INI file BTDevices.ini for setting the Bluetooth devices to use for Ethernet, Dial or PPP over Bluetooth communication.

The INI file is read by EOS on startup to update its configuration. Automatic pairing is performed if the configured Bluetooth devices are not yet paired.

The INI file is updated each time a paired device is added or removed, as well as each time a paired device is selected as the default device, either via BT-DDI or VXBT.

Default Section

The DEFAULT section contains either the friendly name or the Bluetooth address of the devices to be used for media over Bluetooth communication. Users should add this as the first section of the INI file.

Table 33 lists the default section Bluetooth configuration file parameters.

Table 33 Default Section Bluetooth Configuration File Parameters

INI Parameter Name (Type:macro)	Default Values	Min- Max Values	Data Type	Config.sys	Description
DEF_ETH_DEV	NONE	1-32	String	BT_ETH_DEV	Friendly name or Bluetooth address of the default paired device for Ethernet (PAN) profile connection.
DEF_DIAL_DEV	NONE	1-32	String	BT_DIAL_DEV	Friendly name or Bluetooth address of the default paired device for dial (DUN) profile connection.
DEF_SER_DEV	NONE	1-32	String	BT_SER_DEV	Friendly name or Bluetooth address of the default paired device for serial (SPP) connection.

Devices Section

The succeeding sections after the DEFAULT section will contain the parameters for the Bluetooth devices. For each Bluetooth device in the DEFAULT section, user should create a device section for its parameters. The section name should follow the convention DEV_<name> where <name> can be any user defined name.

Table 34 lists the devices section Bluetooth configuration file parameters.

Table 34 Devices Section Bluetooth Configuration File Parameters

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Data Type	Description
NAME	NONE	1-32	String	Friendly name.
BD_ADDR	NONE	0-12	Hex String	12-digit hex value of the Bluetooth address. When this is empty, EOS performs a discovery operation and pairs with the 1st device found that matches the friendly name.
PIN	"0000"	0-16	String	Bluetooth PIN

Sample INI File

The following is a sample INI file for Bluetooth.

BTDevices.ini

(Click to view file attachments)

WiFi Configuration File

WiFi NWIF users have the option to use the INI file WiFiProf.ini for setting the network profile of access points and set which network profile to use for WiFi communication.

The INI file is read by EOS on startup to update its network profiles list.

The INI file is updated each time a network profile is modified, as well as each time a network profile is selected as the default profile, either via WiFi-DDI or VXWIFI.

Default Section

The DEFAULT section will contain the SSID of the network profile to be used for WiFi communication. Users should add this as the 1st section of the INI file.

Table 35 lists the default section WiFi configuration file parameters.

Table 35 Default Section WiFi Configuration File Parameters

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Data Type	Config.sys	Description
DEF_SSID	NONE	0-32	String	WI_DEFAULT_SSID	SSID of the default network profile.

Profiles Section

The succeeding sections after the DEFAULT section will contain the parameters for the network profiles. User can create several profiles section, one section for each network profile. The section name should follow the convention PROF_<name> where <name> can be any user defined name.

Table 36 lists profiles section WiFi configuration file parameters.

Table 36 Profiles Section WiFi Configuration File Parameters

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Data Type	Mandatory / Optional	Description
SSID	NONE	1-32	String	Mandatory	WiFi network name (SSID).
BSS_TYPE	INFRA	5	String		WiFi network type. Valid values: <ul style="list-style-type: none"> • INFRA • ADHOC
CHANNEL	1	1-14	Number	Mandatory	WiFi channel number.
AUTH_TYPE	Open	4-9	String	Mandatory if BSS_TYPE is ADHOC	WiFi authentication type. Valid values: <ul style="list-style-type: none"> • Open • Shared • WPA-PSK

Table 36 Profiles Section WiFi Configuration File Parameters (continued)

INI Parameter Name (Type:macro)	Default Values	Min-Max Values	Data Type	Mandatory / Optional	Description
WSEC	NONE or WEP	3-4	String	Mandatory	<ul style="list-style-type: none"> • WPA2-PSK • WPA-EAP • WPA2-EAP • WPA-CCKM • WPA2-CCKM WiFi encryption type. Valid values: <ul style="list-style-type: none"> • None • WEP • TKIP • CCMP Default value is WEP when AUTH_TYPE is set to Shared. Otherwise, the default value is None.
PSK	NONE	1-64	String	Mandatory if AUTH_TYPE is PSK.	WPA pre-shared key.
WEP1, WEP2, WEP3, WEP4	NONE	4-26	String	Mandatory if WSEC is WEP.	Static WEP keys.
WEP_INDEX	1	1-4	Number	Mandatory if WSEC is WEP.	Default WEP key index.
EAP_TYPE	TLS	3-4	String	Mandatory if AUTH_TYPE is EAP or CCKM.	Valid values: <ul style="list-style-type: none"> • TLS • PEAP • FAST
EAP_IDENTITY	NONE	1-128	String	Mandatory if AUTH_TYPE is EAP or CCKM.	Identity string for EAP
EAP_PWD	NONE	1-128	String	Mandatory if EAP_TYPE is PEAP or FAST.	Password string for EAP
EAP_CACERT	NONE	1-128	String	Mandatory if EAP_TYPE is TLS or PEAP.	File path to CA certificate file
EAP_CLIENTCERT	NONE	1-128	String	Mandatory if EAP_TYPE is TLS or PEAP.	File path to client certificate file
EAP_PRIVKEY	NONE	1-128	String	Mandatory if EAP_TYPE is TLS.	File path to private key file
EAP_PRIVKEY_PWD	NONE	1-128	String	Mandatory if EAP_TYPE is TLS.	Password for private key file

Sample INI File

The following is a sample INI file for WiFi.

[WiFiProf.ini](#)

(Click to view file attachments)

WiFi Configuration Reset

WiFi NWIF users have the option to reset the WiFi configuration via `config.sys` parameter `EOS_WPROF_RST`. The reset operation removes all network profiles. Below are the steps for performing a WiFi configuration reset:

- 1 Set `config.sys` parameter `EOS_WPROF_RST` in GID 1 to 1.
- 2 Delete `WiFiProf.ini`.
- 3 Start WiFi NWIF.

`EOS_WPROF_INI` is removed by EOS after detecting its presence during the NWIF start-up.



CommEngine Interface API (CEIF.lib)

The Verix eVo Communication Engine or CommEngine (VXCE.OUT) provides services to applications. Applications can query, for example, the IP Address of the terminal, start and stop a network connection, etc. These services are provided by an API referred as the CommEngine Interface API (ceAPI). The complete scope and list of application services is the purpose of this section.

The ceAPI is a shared library and this library is referred as the Communication Engine Interface library or CEIF.LIB residing in Verix eVo. All applications will have access to this shared library avail of the services of the CommEngine via ceAPI. Applications must first register using ceAPI before they can benefit from the services.

ceAPI covers a broad range of services. In addition applications may optionally choose to subscribe to unsolicited messages (or events) from CommEngine on the state of the network connection. The ceAPI application services are categorized as:

- Registration
- Device Management
- Communication infrastructure control
- Broadcast Messages
- IP Configuration
- Device Driver (DDI) configuration
- Connection status and configuration query

Message Exchange mxAPI & Message Formatting mfAPI

ceAPI under the covers operates by exchanging messages with CommEngine. ceAPI, to provide application services, constructs a message, referred as the “Request Message” and delivers it to CommEngine. CommEngine reads this message and in response sends the “Response Message”. ceAPI reads this response from CommEngine, analyzes it and provides the received information to the application.

Messages exchanged with CommEngine have a specific format and follow certain simple rules. The messages are a sequence of tag, length and value (TLV). Applications need to create the Request message and have the ability to read and parse the Response message both of which are in TLV format.

To summarize, `ceAPI` to provide application services must be able to:

- Send and receive messages with CommEngine
- The messages exchanged are in the TLV format that CommEngine understands. Similarly `ceAPI` must be able to under the TLV formatted message sent by CommEngine in response.

`ceAPI` takes advantage of two general purpose API to exchanges and format messages:

- `mxAPI` – Message eXchange API. Send and receive messages.
- `mfAPI` – Message Formatting API. Create and parse TLV messages.

NOTE



Both `mxAPI` and `mfAPI` libraries are part of `CEIF.LIB` — a shared library residing in Verix eVo. It is important to note that both `mxAPI` and `mfAPI` are general purpose API and can be used by any application. Application developers are encouraged to use both these libraries in their applications. Using these two APIs it is possible to for any two applications to communicate and exchange messages and data.

ceAPI Concepts

Device & Device Ownership

The `ceAPI` provides applications the capability and flexibility to configure and manage its network interfaces. This section provides the background and core concepts behind the design of `ceAPI`.

In Verix eVo, a device is physical entity such as a modem. Each device has name such as `/DEV/COM1` and referred as the device name.

A device is owned by the process or by the task that opens the device. Once the task is complete, it closes the device and ownership ceases. The device is then available for any other device to use. Device ownership can be explicitly transferred from one task to another. Once the device is transferred no operation can be performed by the task that owned the device.

The key points are: a device has a name and it is owned by the task that opens it. Device ownership can be transferred by the device that currently owns it.

In the VMAC environment, `VMACIF` is required to be downloaded to the terminal.

Device Driver

In Verix eVo, a device driver is a software component that manages a device (described in the [Device & Device Ownership](#) section). This device driver is distinct and different from the OS device driver. A device driver supports a published interface referred as device driver interface or DDI. It is not uncommon to refer to this device driver as DDI driver in common usage.

Each device driver has a configuration file and applications using `ceAPI` can alter this configuration file. Device drivers read this configuration file at start up and configure themselves. The rest of this section provides additional information on the device driver concept and at the end revisit how applications can obtain / alter device driver configuration.

A device driver supports a specific communication technology such as Ethernet, WiFi, GPRS, CDMA, etc. More specifically, it supports the combination of the device (modem) and the communication technology. For example, a device driver supports GPRS on the Vx610 and the device name is `/DEV/COM2`.

It is possible to have another driver that supports mobile phone (GSM) on `/DEV/COM2`. However, since both drivers share the same device, both cannot be simultaneously supported. So if the device driver for GPRS is running, it cannot run the driver for mobile phone (GSM) and vice versa. The device driver to device is a one-to-one relationship, i.e., a device driver is associated with one and only one device. The relationship between a device and the device driver is a one-to-many relationship.

In the previous example the two drivers for GPRS and mobile phone (GSM) share the same device and their operation is mutually exclusive from each other. Other factors also contribute to two device drivers operating mutually exclusive. If two devices are mixed on the same serial communications port (COM port), then only one device is accessible at a time. Alternately, two devices cannot operate at the same time due to radio interference or power requirements or any other number of reasons. When two (or more) drivers have a mutually exclusive relationship, then it is necessary to turn off a driver if the other needs to be turned on.

As described earlier in this section, each device driver has a configuration file. This file contains a list of parameters and corresponding values. At start up, the driver reads the configuration parameters and configures itself. Applications using ceAPI can query and modify device driver configuration. In addition to configuration parameters, the configuration file contains “radio” parameters. When an application requests the value of a radio parameter, the value is fetched from the device. The wireless signal strength is an example of a “radio” parameter. Applications can query and obtain the current value but it cannot change it.

To summarize, a device driver supports device driver interface (DDI) and is associated with a specific device and communication technology. Each driver has a configuration file with parameters and values. Applications may query and modify the parameter value. For the driver to function, the device must be available and owned by the task.

Network Interface (NWIF)

A Network Interface, or NWIF for short, is an entity by which a network connection can be managed. A terminal (or network device) can either be connected or not connected to the network. When connected, the terminal has a network connection. Similarly, when the terminal is disconnected, it has no network connection.

An application using ceAPI can connect a terminal to the network or disconnect a terminal from the network by managing the network interface. The application using ceAPI can query and modify the NWIF configuration, and manage the connection or disconnection processes.

The network interface consists of the following elements:

- NWIF handle, a mechanism to refer to a network interface.
- Device, as described in [Device & Device Ownership](#).
- Device driver, as described in [Device Driver](#).
- Communication technology identifier. This is a property of driver and NWIF inherits it.
- NWIF configuration.
- NWIF connection state and error.

The NWIF works with two Verix eVo components – device driver and the TCP/IP stack. The device driver is started first and after this component is successfully established, the TCP/IP stack is initialized. When both these components are up and functioning, the network connection is established. Similarly, when the network connection is torn down, both the components (driver and stack) must be closed before the network connection is closed.

Network Connection Process & States

CommEngine's objective is to bring up the network interfaces and notify applications where there a change in state. It notifies application via events. This section describes the process of bringing and tearing down the connection. The events are described in the [NWIF Events](#) section.

Bringing up the connection is a multi-step process and is depicted in [Figure 1](#). These steps are transparent to applications. Similarly bringing down the connection is depicted in [Figure 2](#). Using `ceAPI`, an application can start or stop the network interface.

Some applications have requirements to manage the connection process and, using `ceAPI`, it is possible to incrementally bring up or down the connection. For example, a CDMA or GPRS connection first brings up the link by dialing to the service provider then establishes the PPP session. In certain geographical regions, the duration of the PPP connection is billed and there is incentive to tear down the connection when inactive. A practice has evolved to tear down the PPP session but to maintain the link layer up. This approach has performance advantages as the link layer need not be re-established and is cost saving as the PPP session is not idle. Re-establishing the PPP session takes a few seconds. To accomplish this, applications must have the capability to control the connection process to pause at intermediate states and proceed in either direction, to establish the connection or tear it down.

NOTE



It is not uncommon for ISPs and Wireless Service Providers to terminate the link layer when the PPP session is terminated. In such situations, the application making the request will receive additional events that it should manage. It is also possible to incrementally bring up the connection. However, the same may not be true for bringing the connection down as this depends on the service provider.

Figure 1 and Figure 2 below depict this connection and disconnection process.

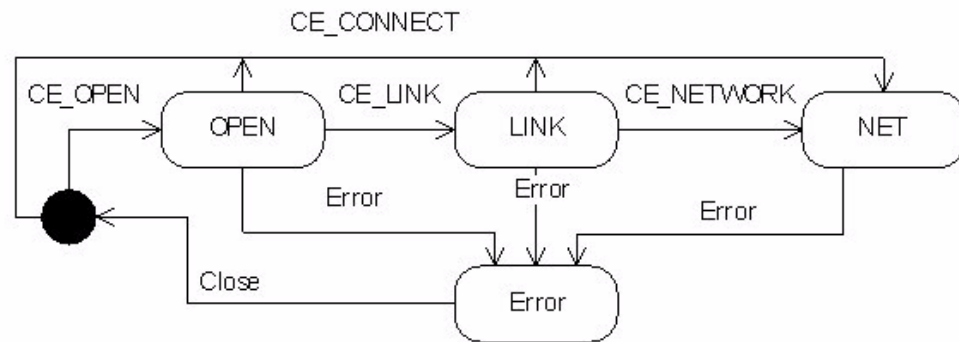


Figure 1 Connection-up state transition diagram (Left to Right)

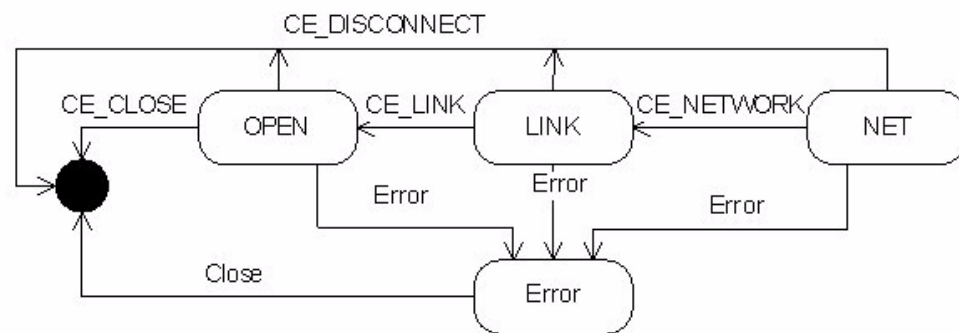


Figure 2 Connection-down state transition diagram (Right to Left)

The connection process takes time and applications need to know when the network connection is up so they can start processing transactions. When the connection reaches an end (terminal) state, it broadcasts an event to all registered applications notifying the applications of the state of the connection. This notification is sent out when the network connection is established, or torn down, or reaches an error state. Applications can also query the status of the connection using `ceAPI`.

The [Handling CommEngine Events](#) section expands on the concept of connection management. An understanding of the `ceAPI` is recommended prior to reviewing topics in the [Application Developer Notes](#) section.

Network Connection State Descriptions

Network Connection State	Description
OPEN	In an OPEN state, this means that CommEngine has successfully initialized the device driver. Once the device driver is initialized, any queries to it will now be allowed. In this state, any AT commands and IOCTL requests will now be possible.
LINK	In a LINK state, the device driver is successfully connected to the network. In a PPP connection, a data link has been established. In the case of a GPRS device driver, this state includes attaching to the network and setting and activating the PDP context.
NET	In a NET state, TCP/IP stack is now up and running. Socket functionalities can now be used. In a PPP connection, this means that the PPP session has been established. For Ethernet DHCP, the IP address has been obtained.
ERROR	In the event that an error occurs while CommEngine is trying to establish the target state requested, CommEngine will bring the connection down until CommEngine's initial state. There is no way to recover the previous target state if an error occurs. Applications will have to manually set the previous state.

Once CommEngine has successfully reached the target state requested, it is CommEngine's responsibility to maintain that target state by default. If for example an application requested CommEngine to establish a connection until NET state, and in the middle of the connection the network was down, CommEngine will always try to re-establish the connection until the previous target state is reached or until re-establishing the network is no longer possible. If re-establishing the connection is no longer possible, CommEngine tears the connection down to its initial state. Changing the target state while CommEngine is trying to re-establish the network is not allowed.

However, CommEngine can be configured to not re-establish the target state requested on certain scenarios via `ceSetNWParamValue()`. If `ceSetNWParamValue()` is used with the `paramNameStr` as `AUTO_RECONNECT` and `paramValue` set to 0 (false), CommEngine does not re-establish the target state, instead CommEngine will remain in an OPEN state. For a GPRS scenario, this would mean that if `AUTO_RECONNECT` is OFF and connection was lost, GPRS would only remain in an initialized and attached state.

NOTE



When working with GPRS and incrementally bringing the network either up or down, the GPRS driver performs the GPRS Attach in the LINK state, while GPRS Detach is done in OPEN state. If the GPRS driver is in the OPEN state and the connection is lost it remains attached. If `ceStartNWIF()` is called to transit into OPEN state, the GPRS driver will not attach to the network.

NWIF Events

Events are posted to registered applications by CommEngine when interesting network events occur. For example, when the network connection is established, an event is broadcasted to all applications that have registered for events. This lets applications know that network connectivity is established and can start processing transactions or any other network activity. Similarly, if the network connection is lost, an event is posted indicating the network connection is down.

Network connection being down is distinct from a failed network connection. When the connection goes down, there is a possibility (not a certainty) that the connection may recover back in the future. A WiFi terminal (e.g. Vx670 WiFi) going out and coming back in range illustrates this situation. When a network connection fails, it usually is configuration that is incompatible with the current environment. If the GPRS APN (Access Point Name) is incorrectly specified, this leads to a failed network connection. The APN value must be corrected before attempting to connect again. See the [List of CommEngine Events](#) section for additional details.

As described in the [Network Connection Process & States](#) section, the connection or disconnection can be incremental. The application moves the connection from one state to the other using ceAPI. Here, states refer to OPEN, LINK and NET as shown in [Figure 1](#) and [Figure 2](#). The state transition process involves multiple steps – first the application calls ceAPI, either `ceStartNWIF()` or `ceStopNWIF()`, to move the connection to the next step. These API calls are non-blocking. They return an operation that is not complete where the request has been accepted and is being processed. When the operation is complete, CommEngine posts an event notifying the application that the requested operation is completed.

The [List of CommEngine Events](#) section provides the list of events generated by CommEngine.

NOTE



Applications are responsible for reading the events. If an application does not read an event, the pipe may become full. If pipe is full, the application will not be able to receive any further events.

Multiple NWIFs

CommEngine supports running multiple NWIFs at the same time, as long as they can work independently of each other. An example of this is running Ethernet and PPP Dial NWIFs on the VX 520 at the same time. However, on the VX 680, both mobile phone (GSM) and GPRS NWIFs cannot be run at the same time. Both mobile phone (GSM) and GPRS NWIFs share the same physical device limiting it only to one NWIF running at a time.

When two or more NWIFs are running at the same time, how the TCP/IP stack is configured (more details below) is affected thus applications should take this into consideration.

NWIFs have implicit hierarchy and when more than one NWIF is running, the TCP/IP stack is configured with gateway and DNS of the highest running NWIF.

On the V*510/VX 520 the hierarchy is:

- Ethernet
- PPP/Dial

In this case, Ethernet is higher on the hierarchy. When both are running, Ethernet takes precedence. Consider the situation when both are running and the Ethernet NWIF is stopped. Since PPP/Dial is the only NWIF running, the TCP/IP stack is now configured with its parameters. When Ethernet is restarted, the TCP/IP stack is reconfigured as Ethernet ranks higher in the hierarchy over PPP/Dial.

This set-up will be dependent on the network cloud that each NWIF is using and how the network is set-up. Also, as only one active default gateway is allowed, the hierarchy of which gateway per NWIF will be using is automatically done by CommEngine. Primary and secondary DNS that will be used will also be done automatically.

Refer to the `ReadMe.txt` document packaged along with the software for a list of terminals supported and their hierarchy.

Below is a sample scenario on how CommEngine will handle multiple running NWIFs.

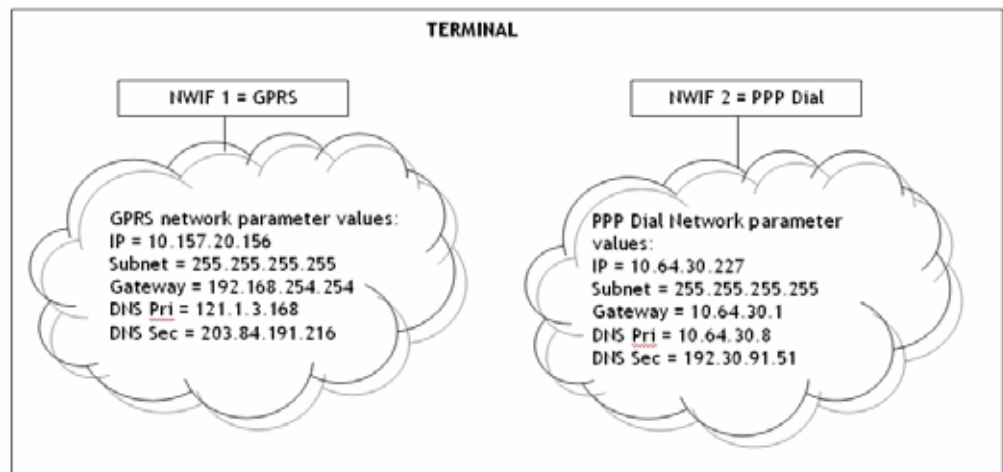


Figure 3 CommEngine Handling Multiple Running NWIFs

If both interfaces are up and the application would want to PING the secondary DNS of the PPP Dial Network interface, this would fail. This is because the packet would pass thru the default gateway of the GPRS network, as GPRS is the primary NWIF. If applications want to make sure that the packet would go thru the default gateway of the PPP Dial network, the GPRS should be brought down. This would force the packet to use the default gateway for the PPP Dial Network.

ceAPI Summary

The ceAPI provides applications services to all registered applications. There are no pre-conditions to registration. The API is categorized based on the type of service it provides, see [Table 37](#) for more information.

Table 37 ceAPI Application Services

Service	API	Description
Registration	<code>ceRegister()</code>	Registers application with CommEngine.
	<code>ceUnregister()</code>	Cancels prior successful registration with CommEngine.
	<code>ceSetCommDevMgr()</code>	Registers application for device management.
Device management	<code>ceRequestDevice()</code>	Requests device from CommEngine.
	<code>ceReleaseDevice()</code>	Releases the device to CommEngine.
Device Driver (DDI) configuration	<code>ceGetDDParamValue()</code>	Retrieves device driver (DD) parameter.
	<code>ceSetDDParamValue()</code>	Sets device driver (DD) parameter.
Sending commands to device	<code>ceExCommand()</code>	Executes a command on the device and obtain response string.
Communication infrastructure configuration and management control	<code>ceGetNWIFCount()</code>	Obtains the number of network interfaces (NWIF) supported.
	<code>ceGetNWIFInfo()</code>	Obtains NWIF information.
	<code>ceStartNWIF()</code>	Starts a specified network interface.
	<code>ceStopNWIF()</code>	Stops a specified network interface.
	<code>ceSetNWIFStartMode()</code>	Specifies if the NWIFs are started at start up (auto mode) or need to be explicitly started (manual mode).
	<code>ceGetNWIFStartMode()</code>	Obtains the current network interfaces start mode.
	<code>ceSetNWParamValue()</code>	Specifies IP and PPP configuration parameter value.
	<code>ceGetNWParamValue()</code>	Retrieves current IP and PPP configuration parameter value.
	<code>ceSetBattThreshold()</code>	Specifies the maximum battery threshold wherein CommEngine allows network operations.
	<code>ceGetBattThreshold()</code>	Retrieves battery threshold.
	<code>ceStartDialIF()</code>	Starts the dial interface and provides the handle to device.
	<code>ceStopDialIF()</code>	Stops the dial interface.
Dial interface support	<code>ceSetDialMode()</code>	Sets <code>ceStartDialIF()</code> / <code>ceStopDialIF()</code> behavior to blocking or non-blocking.
	<code>ceGetDialMode()</code>	Retrieves <code>ceStartDialIF()</code> / <code>ceStopDialIF()</code> behavior.

Table 37 ceAPI Application Services (continued)

Service	API	Description
Event management	<code>ceGetDialHandle()</code>	Retrieves the dial handle value to the application that called <code>ceStartDialIF()</code> .
	<code>ceEnableEventNotification()</code>	Enables notification of events from CommEngine. Application also registers callback function to be called when an event is posted.
	<code>ceDisableEventNotification()</code>	Disables notification of events from CommEngine. After this call the application will no longer receive events.
	<code>ceGetEventCount()</code>	Returns the number of pending CommEngine Events.
	<code>ceGetEvent()</code>	Returns first pending CommEngine notification event.
	<code>ceIsEventNotificationEnabled()</code>	Returns the current state of event notification.
Miscellaneous	<code>ceSetSignalNotification()</code>	Sets flag for signal strength notification events from CommEngine to application.
	<code>ceActivateNCP()</code>	Activates application VxNCP. On successful execution of this API, VxNCP has the focus and ownership of the console.
	<code>ceGetVersion()</code>	Obtains version information for CommEngine Interface library and CommEngine.

Table 38 ceAPI Application Services

Service	API	Description
Registration	<code>ceRegister()</code>	Registers application with CommEngine.
	<code>ceUnregister()</code>	Cancels prior successful registration with CommEngine.
	<code>ceSetCommDevMgr()</code>	Registers application for device management.
Device management	<code>ceRequestDevice()</code>	Requests device from CommEngine.
	<code>ceReleaseDevice()</code>	Releases the device to CommEngine.
Device Driver (DDI) configuration	<code>ceGetDDParamValue()</code>	Retrieves device driver (DD) parameter.
	<code>ceSetDDParamValue()</code>	Sets device driver (DD) parameter.
Sending commands to device	<code>ceExCommand()</code>	Executes a command on the device and obtain response string.

Table 38 **ceAPI Application Services** (continued)

Service	API	Description
Communication infrastructure configuration and management control	<code>ceGetNWIFCount()</code>	Obtains the number of network interfaces (NWIF) supported.
	<code>ceGetNWIFInfo()</code>	Obtains NWIF information.
	<code>ceStartNWIF()</code>	Starts a specified network interface.
	<code>ceStopNWIF()</code>	Stops a specified network interface.
	<code>ceSetNWIFStartMode()</code>	Specifies if the NWIFs are started at start up (auto mode) or need to be explicitly started (manual mode).
	<code>ceGetNWIFStartMode()</code>	Obtains the current network interfaces start mode.
	<code>ceSetNWParamValue()</code>	Specifies IP and PPP configuration parameter value.
	<code>ceGetNWParamValue()</code>	Retrieves current IP and PPP configuration parameter value.
	<code>ceSetBattThreshold()</code>	Specifies the maximum battery threshold wherein CommEngine allows network operations.
	<code>ceGetBattThreshold()</code>	Retrieves battery threshold.
Dial interface support	<code>ceStartDialIF()</code>	Starts the dial interface and provides the handle to device.
	<code>ceStopDialIF()</code>	Stops the dial interface.
	<code>ceSetDialMode()</code>	Sets <code>ceStartDialIF()</code> / <code>ceStopDialIF()</code> behavior to blocking or non-blocking.
	<code>ceGetDialMode()</code>	Retrieves <code>ceStartDialIF()</code> / <code>ceStopDialIF()</code> behavior.
	<code>ceGetDialHandle()</code>	Retrieves the dial handle value to the application that called <code>ceStartDialIF()</code> .
Event management	<code>ceEnableEventNotification()</code>	Enables notification of events from CommEngine. Application also registers callback function to be called when an event is posted.
	<code>ceDisableEventNotification()</code>	Disables notification of events from CommEngine. After this call the application will no longer receive events.
	<code>ceGetEventCount()</code>	Returns the number of pending CommEngine Events.
	<code>ceGetEvent()</code>	Returns first pending CommEngine notification event.
	<code>ceIsEventNotificationEnabled()</code>	Returns the current state of event notification.

Table 38 ceAPI Application Services (continued)

Service	API	Description
Miscellaneous	<code>ceSetSignalNotification()</code>	Sets flag for signal strength notification events from CommEngine to application.
	<code>ceActivateNCP()</code>	Activates application VxNCP. On successful execution of this API, VxNCP has the focus and ownership of the console.
	<code>ceGetVersion()</code>	Obtains version information for CommEngine Interface library and CommEngine.

ceRegister()

Registers application with CommEngine (CE).

Prototype `int ceRegister(void);`

Return Values

0	Registration successful.
ECE_REGAGAIN	Application is attempting to register again to CommEngine even though application is already registered.
ECE_CREATEPIPE	CommEngine failed to create a pipe for the application.
ECE_REGFAILED	Attempt to register to CommEngine failed as CommEngine is not running.
< 0	Error. See List of Error Codes for other return values.

NOTE



This is the first API that must be called by an Application. A pipe is created and message is sent to CE to register. CE responds either to confirm or deny the registration.

`ceRegister()` must be called just once. Calling it multiple times has no affect and results in an error. CE must be running for successful registration. For a multi-threaded application, only one thread needs to register to CE and all the other threads will be able to use any `ceAPI`.

ceUnregister()

Cancels the application's prior successful registration with CE.

Prototype `int ceUnregister(void);`

Return Values

0	Successfully unregistered. Prior registration successfully cancelled.
ECE_NOTREG	Application is not registered with CE.
ECE_CANNOT_UNREG	Application is not allowed to un-register with CE. Possible reasons are: <ul style="list-style-type: none"> • Application was registered as a device management application • Application has devices checked-out or has pending requests for devices
< 0	Error. See List of Error Codes .

`ceRegister()` creates a pipe, registers with CE and allocates necessary resources (memory). `ceUnregister()` does the reverse: it cancels the registration with CE, closes the pipe and frees any other allocated resources.

Calling any other API after `ceUnregister()` will result in failure as the application is no longer registered. The application may register again by calling API `ceRegister()`.

NOTE



If this application is the CE communication device management application and has successfully registered itself via API `ceSetCommDevMgr()`, it should not unregister. Attempts to unregister will fail.

ceSetCommDevMgr()

Registers the application for communication device management.

Prototype `int ceSetCommDevMgr(void);`

Return Values

0	Application successfully registered as CE communication device management application.
ECE_NOTREG	Application has not successfully registered with CE.
ECE_NOCDM	No device management was required.
ECE_CDMAPP	Possible reasons are: <ul style="list-style-type: none"> • A device management application has already been set • Application is attempting to register as the device management application twice
< 0	Error. See List of Error Codes .

NOTE



An application may optionally register itself as CE's communication device management application. *CEDM must be set to 1 if device management is required.

ceRequestDevice()

Requests device from CE.

Prototype `int ceRequestDevice(const char deviceName[]);`

Parameters

In:	deviceName	VeriXV device name. This is the device being request from CommEngine. Device information is returned by API <code>ceGetNWIFInfo</code> in structure element <code>niDeviceName</code> of parameter <code>stArray</code> . This should be a null terminated ASCII string.
-----	------------	--

Return Values

0	Successful. Application may open the device.
ECE_DEVNAMELEN	Length of device name is above the maximum length for a device name. Maximum length of a device is 31.
ECE_DEVNAME	Unknown device name.
ECE_DEVOWNER	Device is being used by another application or is unavailable with CE.
ECE_DEVBUSY	Device is currently busy as it is being used by CommEngine.
< 0	Error. See List of Error Codes .

The calling application is requesting CE for the device `deviceName`. CE transfers the ownership of device to the requesting application. The application is expected to bring down the active network associated with this device before the device is requested.

It is important to distinguish between device and network interface. Usually there is a one to one relationship between network interface and device but there are exceptions. The most common example is that of GPRS devices. Usually these devices also support mobile phone (GSM) and it is possible to support PPP over a mobile phone (GSM) call. A device may support more than one network interfaces but only one network interface may be active at a time.

This API is a blocking call and returns `ECE_SUCCESS` when the requested device is available.

ceReleaseDevice()

Releases the device to CE.

Prototype `int ceReleaseDevice(const char deviceName[]);`

Parameters

In: `deviceName` VerixV device name. This is the device being returned to CE. For Device information refer to API `ceGetNWIFInfo` in structure element `niDeviceName` of parameter `stArray`.

Return Values

0	Successful. CE has accepted the device.
ECE_DEVNAME	Unknown device name.
ECE_NOTDEVOWNER	Possible reasons are: <ul style="list-style-type: none"> • Application did not check out this device • Application is attempting to check in a device that it did not check out
< 0	Failed. See List of Error Codes .

This is the mechanism for an application to “return” a device back to CE. If an application is trying to release a device it does not have ownership on, `ECE_NOTDEVOWNER` is returned. `ECE_DEVNAME` is returned if the device name is unknown. CE brings up the network interface associated with this device if its start mode is `CE_SM_AUTO`.

ceGetDDParamValue()

Retrieves device driver (DD) parameter.

Prototype

```
int ceGetDDParamValue(const unsigned int niHandle, const char
paramNameStr[], const unsigned int paramValueSize, void *paramValue,
unsigned int *paramValueLen);
```

Parameters

In:	niHandle	Handle to network interface. Returned by API <code>ceGetNWIFInfo</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
In:	paramStr	To fetch value associated with parameter <code>paramStr</code> . <code>paramStr</code> is a null terminated string. This parameter cannot be NULL ^[1] .
Out:	paramValue	The value associated with <code>paramStr</code> is returned in <code>paramValue</code> . This parameter cannot be NULL. The type of <code>paramValue</code> is dependent on the data type of <code>paramNameStr</code> . If it is a numeric type, an integer type of <code>paramValue</code> should be passed. In the case of a type Boolean, this will be handled as an integer, this is to be able to support C/C++ applications.
In:	paramValueSize	Size of buffer <code>paramValue</code> . Its value must be greater than zero
Out:	paramValueLen	Number of bytes containing data in <code>paramValue</code> . This value is returned and is always less than or equal to <code>paramValueSize</code> . In the case of a string type, the NULL terminator is also counted as part of the <code>paramValueLen</code> .

Return Values

0	Parameter value obtained.
ECE_PARAM_INVALID	Possible reasons are: <ul style="list-style-type: none"> • <code>iHandle</code> passed is out of range • <code>paramValueSize</code> passed is not valid • IOCTL query for parameter failed
ECE_DD_NOT_READY	Parameter associated with <code>paramNameStr</code> might be a real time parameter. Such queries will only be possible if NWIF is ready for an IOCTL call. Applications can start NWIF to at least an OPEN state for this query to be successful.
ECE_PARAM_NOT_FOUND	Parameter associated with <code>paramNameStr</code> can not be found in the Driver's metadata file. This parameter may not be supported by device driver corresponding to <code>iHandle</code> passed.
< 0	Failed. See List of Error Codes .

NOTE

This API is for fetching the value associated with parameter `paramStr`. Device driver configuration parameters can be obtained (such as timeouts, waiting times, etc.) even if the device is unavailable or network interface is disabled or not running.

Real-time parameters, such as signal strength, wireless carrier, etc. can be obtained from the device/modem. Such queries will only be possible if NWIF is ready for an IOCTL call. Applications can start NWIF to at least an OPEN state for this query to be successful. A list of Device Driver parameters and their definitions are discussed in [Configuration Management](#).

ceSetDDParamValue()

Sets device driver (DD) parameter.

Prototype

```
int ceSetDDParamValue(const unsigned int niHandle, const char
paramNameStr[], const void *paramValue, const unsigned int paramValueLen);
```

Parameters

In:	niHandle	Handle to network interface. Returned by API <code>ceGetNWIFInfo</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
In:	paramStr	To set value associated with parameter <code>paramStr</code> . <code>paramStr</code> is a null terminated string. This parameter cannot be NULL. ^[2]
In:	paramValue	The value associated with <code>paramStr</code> . This parameter cannot be NULL.
In:	paramValueLen	Number of bytes containing data in <code>paramValue</code> . In the case of a string type, The NULL terminator should be included in the count of <code>paramValueLen</code> . In the case of a type Boolean, this will be handled as an integer, this is to be able to support C/C++ applications.

Return Values

0	Parameter value set.
ECE_PARAM_INVALID	Possible reasons are: <ul style="list-style-type: none"> • <code>niHandle</code> passed is out of range • <code>paramValueSize</code> passed is not valid • IOCTL query for parameter failed • <code>paramValue</code> is out of range or not within the minimum or maximum value or length allowed • string is not NULL terminated • <code>paramValue</code> contains non-valid characters
ECE_DD_NOT_READY	Parameter associated with <code>paramNameStr</code> might be a real time parameter. Such queries will only be possible if NWIF is ready for an IOCTL call. Applications can start NWIF to at least an OPEN state for this query to be successful.
ECE_PARAM_NOT_FOUND	Parameter associated with <code>paramNameStr</code> can not be found in the Driver's metadata file. This parameter may not be supported by device driver corresponding to <code>niHandle</code> passed.
ECE_PARAM_READ_ONLY	Parameter associated with <code>paramNameStr</code> is a read-only parameter.
< 0	Failed. See List of Error Codes .

This API is for assigning a value associated with parameter `paramStr`. Device driver configuration parameters can be set (such as timeouts, waiting times, etc.) even if the device is unavailable or network interface is not up. However, some parameters can only be set if a network interface is up. A list of Device Driver parameters and their definitions can be seen in the [Configuration Management](#) section.

NOTE

Applications can also revert to factory defaults by passing `FACTORY_DEFAULT` as `paramNameStr`. Through this setting, all previous settings will be restored to their original values.

ceExCommand()

Executes a command on the device and obtain response string.

Prototype

```
int ceExCommand(const unsigned int niHandle, const void *cmdStr, const
unsigned int cmdStrLen, const unsigned int cmdRespSize, void *cmdResp,
unsigned int *cmdRespLen, const unsigned long timeoutMS);
```

Parameters

In:	niHandle	Handle to network interface. Returned by API <code>ceGetNWIFInfo</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
In:	cmdStr	<code>cmdStr</code> contains the command string to execute. <code>cmdStr</code> is of length <code>cmdStrLen</code> . This parameter cannot be NULL.
In:	cmdStrLen	Length of <code>cmdStr</code> .
In:	cmdRespSize	Size of buffer <code>cmdResp</code> in bytes. Its value must be greater than zero.
Out:	cmdResp	The response in consequent of executing <code>cmdStr</code> .
Out:	cmdRespLen	Number of bytes containing response data in <code>cmdResp</code> . This value is returned and is always less than or equal to <code>cmdRespSize</code> .
In:	timeoutMS	This command is executed by the device driver. Parameter <code>timeoutMS</code> is the duration in milliseconds the device driver will wait for a response before it gives up. This timeout value should be reasonable and long enough to obtain a response. If this parameter is zero, the device driver will wait indefinitely until a response is obtained.

Return Values

0	Command executed successfully.
ECE_EX_CMD_FAILED	IOCTL for specified command failed.
ECE_NOTREG	Application must register before attempting this API.
ECE_TIMEOUT	Timeout occurred while waiting for response
ECE_DD_NOT_READY	Command will only be possible if NWIF is ready for an IOCTL call. Applications can start NWIF to at least an OPEN state for this query to be successful.
ECE_NOT_SUPPORTED	Device Driver does not support sending AT commands.
< 0	Failed. See List of Error Codes for other return values.

This API can be successfully executed if network interface associated with `niHandle` is enabled and running. There are certain device driver states where running of this command is not permissible/advisable.

In the case of an Ethernet, PPP, and BT device driver type, interface should be in an initialized state by calling `ceStartNWIF()` with `CE_OPEN` parameter before calling `ceExCommand()`.

In the case of a Dial device driver type, `ceStartDialIF()` should already be done and the application has the dial device handle before calling `ceExCommand()`.

NOTE

This API is provided as an option for applications where running a specific command is necessary. Usage of this API is **strongly discouraged** and should only be used as a last resort.

ceEnableEventNotification()

Subscribes to notification events from CommEngine. This API is non-blocking and sets the environment for handling notification events from CE.

On calling this API, a pipe is created internally for CE to post notification events. The application receives a pipe event whenever CE posts a notification event. The application can fetch the notification event via ceAPI `ceGetEvent()`.

There are two events – the first is the CE notification event which the application registered for and the second, the OS pipe event. Though both are referred as events, the delivery mechanism for both is different. The OS pipe event lets the application know that a CE notification event is available which can be fetched by using API `ceGetEvent()`.

An application may have multiple pipes open and on receiving a pipe event it needs to determine the pipe that caused the OS pipe event to be posted. The application may use ceAPI `ceGetEventCount()` to determine the number of CE notification events pending. If the value returned by API `ceGetEventCount()` is zero, the application may choose to check other pipes.

See API [ceGetEvent\(\)](#) for details on event structure.

Prototype	<code>int ceEnableEventNotification(void);</code>	
Return Values	0	Successful.
	ECE_EVENTS_AGAIN	Application is trying to enable notification events from CommEngine again.
	< 0	Failed. See List of Error Codes .

ceDisableEventNotification()

Unsubscribes to CE notification events. After this call, the application will no longer receive CE notification events.

Prototype `int ceDisableEventNotification(void);`

Return Values

0	Successful.
< 0	Failed. Error. See List of Error Codes .

NOTE



Event notification is disabled by default. Use API `ceEnableEventNotification()` to subscribe to CE notification events.

ceGetEventCount()

Returns the number of pending CE events.

Prototype `int ceGetEventCount(void);`

Return Values

<code>>= 0</code>	Number of pending messages
----------------------	----------------------------

This API returns the number of pending CE notification events. If there are none or the application has not enabled notification events, the value returned is zero.

ceGetEvent()

Returns first pending CE notification event.

Prototype

```
int ceGetEvent(stceNWEvt *ceEvt, const int applExtEvtDataSz, void
*applExtEvtData, int *applExtEvtDataLen);
```

Parameters

In:	ceEvt	Pointer to CommEngine notification event structure. See CommEngine Notification Event Structure for details of the structure.
In:	applExtEvtDataSz	Size of buffer pointed by applEvtData. Set this zero if no extended application events are expected or desired by the application.
Out:	applExtEvtData	Extended application event data will be loaded in this buffer. Set this parameter to NULL if no application events are expected or desired by the application.
Out:	applExtEvtDataLen	Length of buffer populated with extended application event data. Set this parameter to NULL if no application events are expected or desired.

Return Values

0	Success
ECE_NO_CE_EVENT	No pending CommEngine notification events are present.
< 0	Failed. See List of Error Codes .

Use API `ceGetEvent()` to fetch CE notification events. [CommEngine Notification Event Structure](#) describes structure `stceNWEvt` where the event information is returned. [List of CommEngine Events](#) contains the list of all events returned by CE.

CE works with device drivers and some of these drivers generate extended application events (`CE_EVT_DDI_APPL`). The extended application event data is returned in parameter `applExtEvtData`. The size of buffer `applExtEvtData` is specified in parameter `applExtEvtDataSz`. The actual length populated is returned in `applExtEvtDataLen`.

In the case of a `CE_EVT_NET_FAILED`, `CE_EVT_NET_OUT` and `CE_EVT_NET_POOR` event, CE also sends out extended errors so that applications can determine why a `CE_EVT_NET_FAILED`, `CE_EVT_NET_OUT` or `CE_EVT_NET_POOR` was sent by CE. The extended error string is returned in parameter `applExtEvtData`. Handling of the extended error string on a `CE_EVT_NET_FAILED`, `CE_EVT_NET_OUT` or `CE_EVT_NET_POOR` will just be like handling a `CE_EVT_DDI_APPL`.

CommEngine Notification Event Structure

```

typedef struct
{
    unsigned int niHandle;        // Handle to Network Interface
    unsigned int neEvt;          // Network Interface event, see List of CommEngine Events.
    int neParam1;                // Populated where applicable. Zero otherwise. Used with CE_EVT_SIGNAL (signal in percentage), CE_EVT_NET_FAILED (error code) and CE_EVT_DDI_APPL (application event).
    int neParam2;                // Populated where applicable. Zero otherwise. Used with CE_EVT_SIGNAL (signal in dBm).
    int neParam3;                // Populated where applicable. Zero otherwise. Used with CE_EVT_SIGNAL (signal in RSSI).
} stceNWEvt;                    //

```

ceSetSignalNotification()

Sets the flag of signal strength notification events from CE to the application. For an application to receive signal strength notifications, it must call this API with the required flag. This API enables delivery of all network events to the application, including signal strength. Signal strength events are sent by CommEngine only if the underlying hardware supports wireless and signal strength makes sense. In addition to this, the network interface associated with wireless must be initialized.

CE sends signal strength updates at flag chosen by the application. For example, if the application chooses ON (CE_SF_ON) then it is sent. CE starts sending signal strength notifications once device driver has been initialized, and stops signal strength notifications once driver has been released.

Signal strength notifications are received when:

- Driver has just initialized
- If there are changes to the signal strength
- If any application turns their signal strength notifications on while the driver has initialized. This is to ensure that if any applications turned on their signal strength notification, they will immediately receive an update.
- If any application unregisters or turns-off event notifications.
- CE instantly sends CE_EVT_SIGNAL event to the application that recently subscribed for signal notification.

If applications were not able to process the signal strength notifications immediately, it can query current signal level by calling API ceGetNWParamValue() and passing SIGNAL_LEVEL as the paramStr.

When the application is preparing to go on power saving mode, it is recommended that all notifications be disabled by calling API ceDisableEventNotification(). To obtain the current flag of signal strength notification setting, use API ceIsEventNotificationEnabled(). Refer to [Signal Event Notification](#) for details of the event structure.

Prototype

```
int ceSetSignalNotification(const unsigned int flag);
```

Parameters

In:	flag	Specify the flag at which notifications are sent. The possible flag values are CE_SF_ON or CE_SF_OFF. Flag values are defined in Table 41 Constants .
-----	------	---

Return Values

0	Successful.
ECE_PARAM_INVALID	Parameter flag passed is not a valid flag value.
< 0	Failed. See List of Error Codes .

Signal Event Notification

The above section describes the process of enabling (and disabling) signal strength notification events. The signal event, similar to all events, is obtained by calling `ceAPI ceGetEvent()`. The event structure is described in the [CommEngine Notification Event Structure](#) section and is populated with signal data (see table below). The extended event data will hold the actual RSSI value with the data type of an integer. The actual RSSI value will be in the range of 0 to 31. If the value is 99, it implies the signal value is unavailable.

Table 39 **Signal Event**

Structure Element	Description
<code>unsigned int niHandle</code>	Handle to Network Interface
<code>unsigned int neEvt</code>	<code>CE_EVT_SIGNAL</code>
<code>int neParam1</code>	Signal strength as percentage. Values are in the range 0 to 100%. If the value is -1, it implies the signal value is unavailable.
<code>int neParam2</code>	Signal strength in dBm. Usually in the range of -51dBm (Excellent) to -113dBm (Poor). These are typical values may change depending on the geography and wireless medium (GPRS / CDMA / WiFi, etc.). If the value is 99, it implies the signal strength is unavailable or unknown.
<code>int neParam3</code>	Actual RSSI value. Usually in the range of 0 to 31. This is the actual value that the GPRS modem is giving. If the value is 99, it implies the signal strength is unavailable or unknown.

celsEventNotificationEnabled()

Returns the current state of event notification and flag of signal strength notification.

Prototype `int ceIsEventNotificationEnabled(unsigned int *flag);`

Parameters

out: flag	Specify the flag at which notifications are sent. The possible flag values are CE_SF_ON or CE_SF_OFF. Flag values are defined in Table 41 Constants .
--------------	---

Return Values

1	Event notification enabled.
0	Event notification disabled.
ECE_PARAM_NULL	Pointer flag passed in NULL.
ECE_NOTREG	Application needs to register to use this API.
< 0	Failed. See List of Error Codes .

NOTE



When event notification is disabled, no signal strength notifications are sent regardless of flag value. Only when event notifications are enabled can signal strength notifications be sent at the chosen flag.

ceGetNWIFCount()

Obtains the number of network interfaces (NWIF) supported.

Prototype

```
int ceGetNWIFCount(void);
```

Return Values

count	Number of network interfaces
< 0	Failed. See List of Error Codes .

Terminals support multiple communication devices (or modems) referred to as network interfaces. This API returns the total number of network interfaces supported. For example, on the Vx570, the two network interfaces are Ethernet and the Eisenhower modem (over PPP).

NOTE



It is recommended that this API be called prior to API `ceGetNWIFInfo()`. The count provided by this API is required as input to API `ceGetNWIFInfo()`.

ceGetNWIFInfo()

Obtains Network Interface information. See description of structure `stNIInfo` for details.

Prototype `int ceGetNWIFInfo(stNIInfo stArray[], const unsigned int stArraySize, unsigned int *arrayCount);`

Parameters

In / Out:	stArray	Pointer to array of stArraySize elements of stNIInfo.
In:	stArraySize	The number of elements in parameter stArray. Its value should be the value returned by API ceGetNWIFCount()
Out:	arrayCount	Number of elements in stArray populated. The value of parameter arrayCount is the same as stArraySize if stArraySize is same as the value returned by API ceGetNWIFCount().

Return Values

0	Successful.
< 0	Failed. See List of Error Codes .

Example This API returns the information associated with each network interface in structure `stNIInfo`. The sample code is shown below.

Configuration Structure stNIInfo

```
typedef struct
{
    unsigned int niHandle;                // Handle to Network Interface
    char niCommTech[16];                 // Null terminated str. See Table 41 Constants for the complete list.

    char niDeviceName[32];               // /DEV/COMx /DEV/WLN1 /DEV/ETH1
    unsigned int niRunState;             // 1 - Running; 2 - Ready; 3 - Not Ready; 0 - Failed to run

    int niErrorCode;                    // Associated error code
    unsigned int niStartUpMode;          // 0 - Auto startup; 1 - Manual startup
    char niDeviceDriverName[18+1];      // Format
                                        // [2 char media type][2 char vendor ID][4 char device id]>.lib
    int niDeviceDriverType;              // 1 - Ethernet; 2 - PPP; 3 - Dial
} stNIInfo;
```

ceStartNWIF()

Starts network interface associated with the handle.

Prototype

```
int ceStartNWIF(const unsigned int niHandle, const int cl);
```

Parameters

In:	niHandle	Handle to network interface. Returned by API <code>ceGetNWIFInfo</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
In:	cl	Specify the extent to which the connection is established. To connect all the way, set parameter <code>cl</code> to <code>CE_CONNECT</code> . To perform incremental connection, set <code>cl</code> to <code>CE_OPEN</code> , <code>CE_LINK_</code> or <code>CE_NETWORK</code> . For definitions, see Table 41 Constants .

Return Values

0	Successful.
ECE_START_STOP_BUSY	<code>ceStartNWIF()</code> is still busy processing a previous <code>ceStartNWIF()</code> or <code>ceStopNWIF()</code> request. Possible scenarios are: <ul style="list-style-type: none"> CommEngine is still bringing up or tearing down the network CommEngine is busy re-establishing the network In the middle of the connection, the battery was removed so CommEngine is busy tearing down the network
ECE_EVENTS_AGAIN	Enabling of events is required when using <code>ceStartNWIF()</code> . Not enabling events will produce this error code.
ECE_STATE_ERR	Application passed an invalid state. Possible scenarios are: <ul style="list-style-type: none"> CommEngine state has already reached target connection state target connection state is greater than CommEngine's current state
ECE_PARAM_INVALID	Application passed an invalid network interface such as a dial only network interface.
ECE_DEVOWNER	CommEngine is currently not the owner of this device. Another application might have requested this device and has not yet released it to CommEngine.
ECE_DEVBUSY	Device is currently busy processing another request. If the device is a shared device, it most likely is being used by another NWIF.

ECE_INIT_FAILURE	Device failed to initialize. The device driver library file does not exist.
< 0	Failed. See List of Error Codes .

The network interface can either start the connection incrementally or completely. See [Network Connection Process & States](#) for details on bringing up the connection.

This API is non-blocking. When it returns successfully, the request has been accepted by CommEngine. When the network interface is started (or moves to the next state), CommEngine posts an event. If the interface cannot be started for any reason, CommEngine posts an event. See [List of CommEngine Events](#).

In the case of having static IP configurations, CommEngine will always post CE_EVT_NET_UP. It is up to the user to ensure validity of the set environment.

In the case of a PPP Authentication failure, CommEngine will try to re-establish the network three (3) times before finally tearing the network down. Applications can change the PPP configurations at this time, and no call to ceStartNWIF() will be needed to re-connect. If changes to PPP configurations were done, after the three retries have been exhausted, ceStartNWIF() should be called again.

If the battery is removed during the middle of a connection, CommEngine will tear the network down and will automatically re-establish communication based on the last target state once the battery is reinserted. If the battery level goes below the minimum threshold level set during the middle of a connection, CommEngine will tear the network down and will automatically re-establish communication based on the last target state once the terminal is docked.

ceStartNWIF() and ceStopNWIF() calls for a BT device driver type, will not be accepted. Just like a DIAL device driver type, this would return ECE_NOT_SUPPORTED. This is because the interface for BT should always be running. For an interface running over Bluetooth, ceSetNWParamValue(BT_CONFIG) should be called first before starting the interface.

ceStopNWIF()

Stops network interface associated with handle.

Prototype `int ceStopNWIF(const unsigned int niHandle, const int cl);`

Parameters

In:	niHandle	Handle to network interface. Returned by API <code>ceGetNWIFInfo</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
In:	cl	Specify the extent to which the connection is established. To disconnect all the way, set parameter <code>cl</code> to <code>CE_DISCONNECT</code> . To perform incremental disconnect set <code>cl</code> to <code>CE_NETWORK</code> , <code>CE_LINK_</code> or <code>CE_CLOSE</code> . For definitions, see Table 41 Constants .

Return Values

0	Successful.
ECE_START_STOP_BUSY	<code>ceStartNWIF()</code> is still busy processing a previous <code>ceStartNWIF()</code> or <code>ceStopNWIF()</code> request. Possible scenarios are: <ul style="list-style-type: none"> CommEngine is still bringing up or tearing down the network CommEngine is busy re-establishing the network In the middle of the connection battery was removed so CommEngine is busy tearing down the network
ECE_EVENTS_AGAIN	Enabling of events is required when using <code>ceStartNWIF()</code> . Not enabling events will produce this error code.
ECE_STATE_ERR	Application passed an invalid state. Possible scenarios are: <ul style="list-style-type: none"> CommEngine state has already reached target connection state Target connection state is greater than CommEngine's current state
ECE_PARAM_INVALID	Application passed an invalid network interface such as a dial only network interface.
ECE_DEVOWNER	CommEngine is currently not the owner of this device. Another application might have requested this device and has not yet released it to CommEngine.
ECE_DEVBUSY	Device is currently busy processing another request. If the device is a shared device, it most likely is being used by another NWIF.
< 0	Failed. See List of Error Codes .

The network interface can either start the connection incrementally or completely. See [Network Connection Process & States](#) for details on bringing up the connection.

This API is non-blocking. When it returns successfully, the request has been accepted by CommEngine. When the network interface is stopped (or moves to the next state), CommEngine posts an event. See [List of CommEngine Events](#).

If the battery was removed during the middle of a connection and the target state is not `NWIF_CONN_STATE_INIT`, CommEngine will tear the network down and will automatically re-establish communication based on last target state once battery is re-inserted. If battery level goes below minimum threshold set during the middle of a connection and target state is not `NWIF_CONN_STATE_INIT`, CommEngine will tear the network down and will automatically re-establish communication based on the last target state once the terminal is docked.

`ceStartNWIF()` and `ceStopNWIF()` calls for a BT device driver type, will not be accepted. Just like a DIAL device driver type, this would return `ECE_NOT_SUPPORTED`. This is because the interface for BT should always be running.

ceSetNWIFStartMode()

Specifies if a network interface is automatically initialized during start up (auto mode) or explicitly started (manual mode).

Prototype

```
int ceSetNWIFStartMode(const unsigned int niHandle, const unsigned int startmode)
```

Parameters

In:	niHandle	Handle to network interface. Returned by API <code>ceGetNWIFInfo()</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
In:	startmode	<ul style="list-style-type: none"> 1 – Manual mode (<code>CE_SM_MANUAL</code>). Network interfaces must be explicitly started. 0 – Auto mode (<code>CE_SM_AUTO</code>). At power up or restart, this network interfaces will automatically start.

Return Values

0	Successful.
ECE_PARAM_INVALID	Application passed an invalid startmode or <code>niHandle</code> passed is out of range.
< 0	Failed. See List of Error Codes .

NOTE



Setting the auto mode affects the next time the terminal is either cold or warm booted. There is no immediate effect.

In auto mode, the connection is established completely (it is equivalent to `ceStartNWIF(handle, CE_CONNECT)`). For starting incrementally, set the start mode to manual (`CE_SM_MANUAL`). In the case of a Dial NWIF, setting of start mode does not apply. Any start mode value set to Dial NWIF will be ignored.

ceGetNWIFStartMode()

Obtains the network interface start mode.

Prototype `int ceGetNWIFStartMode(const unsigned int niHandle);`

Parameters

In:	niHandle	Handle to network interface. Returned by API <code>ceGetNWIFInfo()</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
-----	----------	--

Return Values

CE_SM_AUTO	Auto mode. At power up or restart, all enabled network interfaces are started.
CE_SM_MANUAL	Manual mode. Network interfaces must be explicitly started by using API <code>ceStartNWIF()</code> .
ECE_PARAM_INVALID	Application <code>niHandle</code> passed is out of range.
< 0	Failed. See List of Error Codes .

In the case of a Dial NWIF, start mode will always return CE_SM_MANUAL.

ceSetNWParamValue()

Sets the network (NW) parameter.

Prototype

```
int ceSetNWParamValue(const unsigned int niHandle, const char paramStr[],
const void *paramValue, const unsigned int paramValueLen);
```

Parameters

In:	niHandle	Handle to network interface. Returned by API <code>ceGetNWIFInfo</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
In:	paramStr	To set value associated with parameter <code>paramStr</code> . <code>paramStr</code> is a null terminated string. This parameter cannot be NULL. The list of parameters is available in the List of Network Parameters section.
In:	paramValue	The value associated with <code>paramStr</code> . This parameter cannot be NULL.
In:	paramValueLen	Number of bytes containing data in <code>paramValue</code> .

Return Values

0	Parameter value set.
ECE_NOTREG	Application must register before attempting this API.
ECE_PARAM_INVALID	Possible reasons are: <ul style="list-style-type: none"> • application passed a dial only network interface • application passed an Ethernet driver type for a PPP_CONFIG • <code>niHandle</code> passed is out of range • <code>paramValue</code> passed is out of range
ECE_PARAM_NULL	<code>paramValue</code> passed is missing a required field. In the case of an IP_CONFIG, IP address, Subnet Mask and gateway are required.
ECE_SUBNET_MASK_INVALID	Subnet Mask passed is invalid.
ECE_NOT_SUPPORTED	<code>niHandle</code> passed does not support <code>paramStr</code> .
< 0	Failed. See List of Error Codes .

This API is for assigning a value associated with parameter `paramStr`. Certain network configuration parameters can be even if the device is unavailable or network interface is disabled or not running.

In the case of setting a `ncSubnet`, an error code of `ECE_SUBNET_MASK_INVALID` will be returned if it is invalid. An error code of `ECE_PARAM_NULL` will be returned if a required parameter is not set. When setting `ncDNS1`, `ncDNS2` may be NULL. However, the reverse is not allowed. If `ncDNS2` is not NULL and `ncDNS1` is NULL, `ncDNS2` and `ncDNS1` will be reversed internally by CE.

When setting for a static IP configuration, make sure to set the variable `ncDHCP` to 0. As for using DHCP configuration, make sure to set the variable `ncDHCP` to 1. When setting `AUTO_RECONNECT`, if the value passed is not 1 (true), it will be treated as 0 (false). `AUTO_RECONNECT` is only supported for a PPP device driver type and is only applicable when a connection is lost. For an interface running over Bluetooth, `ceSetNWParamValue(BT_CONFIG)` should be called first before starting the interface.

ceGetNWParamValue()

Retrieves the network (NW) parameter value.

Prototype

```
int ceGetNWParamValue(const unsigned int niHandle, const char
paramStr[], void *paramValue, const unsigned int paramValueSize, unsigned
int *paramValueLen);
```

Parameters

In:	niHandle	Handle to network interface. Returned by API <code>ceGetNWIFInfo</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
In:	paramStr	To fetch value associated with parameter <code>paramStr</code> . <code>paramStr</code> is a null terminated string. This parameter cannot be NULL. The list of parameters is available in the List of Network Parameters section.
Out:	paramValue	The value associated with <code>paramStr</code> is returned in <code>paramValue</code> . This parameter cannot be NULL.
In:	paramValueSize	Size of buffer <code>paramValue</code> . Its value must be greater than zero.
Out:	paramValueLen	Number of bytes containing data in <code>paramValue</code> . This value is returned and is always less than or equal to <code>paramValueSize</code> .

Return Values

0	Parameter value obtained.
ECE_NOTREG	Application must register before attempting this API.
ECE_PARAM_INVALID	Possible reasons are: <ul style="list-style-type: none"> • Application passed a dial only network interface • Application passed an Ethernet driver type for a <code>PPP_CONFIG</code> • <code>niHandle</code> passed is out of range
ECE_SIGNAL_FAILURE	Query of signal level to DDI Driver failed.
< 0	Failed. See List of Error Codes .

This API fetches the value associated with parameter `paramStr`. Certain network configuration parameters may be obtained even if the device is unavailable or the network interface is not running.

NOTE



Certain parameters, such as DHCP lease time are meaningful only when the network interface is running.

ceSetBattThreshold()

Specifies the maximum battery threshold where CommEngine allows network operations.

Prototype `int ceSetBattThreshold(const unsigned int iLevel);`

Parameters

In:	iLevel	Maximum battery level to set. Values must be in the range of CE_BATT_LEVEL_MIN and CE_BATT_LEVEL_MAX.
-----	--------	--

Return Values

0	Successful.
ECE_PARAM_INVALID	iLevel value is out of range.
ECE_NOT_SUPPORTED	Setting of battery threshold is not supported. Terminal must be portable.
ECE_BATTERY_THRESHOLD_FAILURE	OS API <code>get_battery_sts()</code> or <code>set_battery_value()</code> returned -1. The most common reason for failure is when the battery is not attached to the terminal.
< 0	Failed. See List of Error Codes .

Depending on the terminal, Battery Threshold may not always be supported. When setting the battery threshold, always check if the battery is attached first to avoid any errors. The configuration parameter *EOSMINBATT in GID 1 may also be used to set battery threshold. CommEngine at start up looks for parameter *EOSMINBATT and sets the minimum battery threshold. If this parameter is not present or incorrect, CE_BATT_LEVEL_MIN is assumed.

ceGetBattThreshold()

Retrieves the battery threshold.

Prototype `int ceGetBattThreshold(void);`

Return Values

<code>>= CE_BATT_LEVEL_MIN</code>	The Battery Threshold CommEngine is using.
<code>ECE_NOT_SUPPORTED</code>	Battery Threshold in the terminal is not supported.
<code>< 0</code>	Failed. See List of Error Codes .

NOTE



Depending on the terminal, battery threshold may not always be supported.

ceStartDialIF()

Starts the dial interface and provides the handle to device.

Prototype

```
int ceStartDialIF(const unsigned int diHandle, const unsigned int deSize,
char *dialErrorStr);
```

Parameters

In:	diHandle	Handle to dial interface. Returned by API ceGetNWIFInfo in structure element niHandle of parameter stArray.
In:	deSize	Size of buffer dialError.
Out	dialErrorStr	Null terminated error string provided by the dial device driver.

Return Values

> 0	Dial device handle
ECE_DIAL_ONLY_FAILED	Dial interface failed. See dialErrorStr for error details.
ECE_PARAM_INVALID	diHandle passed may not be a Dial driver type or is out of range
ECE_DD_NOT_READY	Device driver is currently not in the proper state to process request.
ECE_DEVBUSY	Device is currently busy processing another request. If the device is a shared device, it most likely is being used by another NWIF.
ECE_DEVOWNER	CommEngine is currently not the owner of this device. Another application might have requested this device and has not yet released it to CommEngine.
ECE_START_STOP_BUSY	CommEngine is tearing down the connection. Possible causes may be: <ul style="list-style-type: none"> • In the middle of the connection battery was removed • Connection was lost
< 0	Failed. See List of Error Codes .

After the dial interface is successfully started, application uses the device handle returned to perform `read()` and `write()` operations. Even if the applications share the dial device handle that was returned by `ceStartDialIF()`, only the application that called `ceStartDialIF()` will have a successful `read()` / `write()`. When complete, close the interface using `ceAPI ceStopDialIF()`.

If a `Connection Lost` occurs in the middle of a `DIAL ONLY` or a `GSM ONLY` connection, CommEngine will revoke the ownership of the application on the device handle and will automatically tear down the connection. Applications will experience write / read error when this happens.

By default, `ceStartDialIF()` is a blocking call, but can be a non-blocking API. Blocking or non-blocking behavior of `ceStartDialIF()` can be set via `ceSetDialMode()`. An application will receive `CE_EVT_START_DIAL` event and dial device handle can be extracted from `neParam1`, if `ceStartDialIF()` is set to non-blocking. If `ceStartDialIF()` is set to non-blocking, `CE_EVT_STOP_CLOSE` will be received.

Event handling may change if `ceStartDialIF()` is used. Broadcast events can still be received but single application events are only received once `ceStartDialIF()` is set to non-blocking mode. No automatic reconnection or bringing up of interfaces is done for a `DIAL ONLY` driver type because the dial device handle may change if a reconnection is performed.

ceStopDialIF()

Stops the previously started dial interface. Use ceAPI ceStopDialIF() to close the dial interface that was previously started using ceStartDialIF(). By default, ceStopDialIF() is a blocking call, but can be a non-blocking API. The blocking or non-blocking behavior of ceStopDialIF() can be set using ceSetDialMode().

If ceStartDialIF() is still busy processing a request and an application called ceStopDialIF(), the first ceStopDialIF() call will be accepted and CommEngine will disregard the ceStartDialIF() request and will start to bring down the interface.

Prototype

```
int ceStopDialIF(const int short diHandle);
```

Parameters

In:	diHandle	Handle to dial interface. Returned by API ceGetNWIFInfo in structure element niHandle of parameter stArray. This is the same handle provided in ceStartDialIF().
-----	----------	--

Return Values

0	Interface successfully closed
ECE_PARAM_INVALID	diHandle passed may not be a Dial driver type or is out of range.
ECE_DD_NOT_READY	Device driver is currently not in the proper state to process request.
ECE_NOTASKID	ceStopDialIF() is called by a different application instead of the one that called ceStartDialIF().
ECE_START_STOP_BUSY	CommEngine is tearing down the connection. Possible causes may be: <ul style="list-style-type: none"> • In the middle of the connection battery was removed • Connection was lost
< 0	Failed. See List of Error Codes .

NOTE



The first application that called ceStartDialIF() successfully will be the only application that can call ceStopDialIF() successfully. No other application will be allowed to bring down the PPP connection.

ceSetDialMode()

Sets `ceStartDialIF()` or `ceStopDialIF()` behavior to blocking or non-blocking.

Prototype `int ceSetDialMode (const unsigned int mode);`

Parameters

In:	Mode	Behavior of <code>ceStartDialIF()</code> / <code>ceStopDialIF()</code>
		<ul style="list-style-type: none"> 0 – <code>ceStartDialIF()</code> or <code>ceStopDialIF()</code> will be blocking (CE_DIALONLY_BLOCKING) 1 – <code>ceStartDialIF()</code> or <code>ceStopDialIF()</code> will be non-blocking (CE_DIALONLY_NONBLOCKING)

Return Values

0	Dial mode successfully set
ECE_PARAM_INVALID	Mode passed may not be 1 or 0
< 0	Failed. See List of Error Codes .

NOTE



Use ceAPI `ceGetDialMode()` to verify the behavior for `ceStartDialIF()` or `ceStopDialIF()`. By default it is set to CE_DIALONLY_BLOCKING.

ceGetDialMode()

Retrieves `ceStartDialIF()` or `ceStopDialIF()` behavior.

Prototype `int ceSetDialMode(void);`

Return Values

0	Dial mode successfully set
ECE_PARAM_INVALID	Mode passed may not be 1 or 0
< 0	Failed. See List of Error Codes .

NOTE



Use ceAPI `ceSetDialMode()` to modify the behavior for `ceStartDialIF()` or `ceStopDialIF()`. By default it is set to `CE_DIALONLY_BLOCKING`.

ceGetDialHandle()

Retrieves the dial handle value to the application that called `ceStartDialIF()`.

Prototype

`ceGetDialHandle(const unsigned diHandle, const unsigned int deSize, char *dialErrorStr);`

Parameters

In:	diHandle	Handle to dial interface. Returned by API <code>ceGetNWIFInfo</code> in structure element <code>niHandle</code> of parameter <code>stArray</code> .
	deSize	Size of buffer <code>dialError</code>
	dialErrorStr	Null terminated error string provided by the dial device driver.

Return Values

ECE_START_STOP_BUSY	<code>ceStartDialIF()</code> is still processing as it maybe in non-blocking mode.
ECE_PARAM_INVALID	One possible reason is that the <code>diHandle</code> passed is not a DIAL type
ECE_DD_NOT_READY	Device is not in the state to provide the dial handle.
ECE_NOTASKID	<code>ceGetDialHandle()</code> is called a different application instead of the one that called <code>ceStartDialIF()</code> .
< 0	Failed. See List of Error Codes .

Only the application that called `ceStartDialIF()` may be able to call `ceGetDialHandle()`. Application that was not able to extract dial handle from event `CE_EVT_START_DIAL` can use this API instead.

In case `ceStartDialIF()` failed, `diHandle` will be set to `ECE_DIAL_ONLY_FAILED` or error code set by DDI Driver but error code can only be extracted by application that called `ceStartDialIF()`.

ceDualSIMSwitch()

Initiates manual switching of SIM when network is connected.

Prototype

```
int ceDualSIMSwitch(const unsigned int niHandle, const int niSIM);
```

Parameters

In: niHandle	Handle to Network/Dial interface.
In: niSIM	Indicates which SIM to switch to. Valid values are: <ul style="list-style-type: none"> • 1: SIM in slot 1. • 2: SIM in slot 2. • 0: Switch to unconnected SIM.

Return Values

ECE_SUCCESS	Successful.
ECE_NOT_SUPPORTED	Terminal does not support dual SIM.
ECE_START_STOP_BUSY	CE is still busy with processing a ceStartNWIF() or ceStopNWIF() request.
ECE_PARAM_INVALID	Parameter passed is invalid.
ECE_STATE_ERR	Network interface is still not up.

NOTE



When network is down, application should use the DDI parameter GP_DSIM_PRI (GPRS) or GS_DSIM_PRI (GSM) to select the SIM to be used before starting the network interface.

ceActivateNCP()

Activates application VxNCP. On successful execution of this API, VxNCP has the focus and ownership of the console.


Prototype

int ceActivateNCP(void);

Return Values	
0	Successful. Verix eVo application VxNCP activated.
ECE_VXNCP_CONSOLE	Application does not own console. Applications must own console before NCP can be activated.
ECE_VXNCP_PIPE	VxNCP is not running.
ECE_VXNCP_PIPECR	Unable to create pipe to communicate with NCP.
< 0	Failed. See List of Error Codes .

Any application that owns the console (/dev/console) can activate VxNCP by calling this API. When user exits VxNCP, it returns back to this application, i.e., the application that activated it.

NOTE



VxNCP sets printer font to default. If application wants to use previous settings of font, application must set the printer font again after VxNCP exits.

ceGetVersion()

Obtains the version information for CommEngine Interface library (CEIF.LIB) and CommEngine (VxCE.OUT).

Prototype

```
int ceGetVersion(const unsigned short component, const unsigned short verStrSize, char *verStr)
```

Parameters

In:	component	Verix eVo component to obtain version information. See Table 41 Constants for the list of values.
In:	verStrSize	Size of buffer verStr.
Out:	verStr	Buffer where the version string is returned. A null terminated string is returned provided the buffer size verStrSize is sufficient.

Return Values

0	Success. Value returned in parameter verStr.
ECE_VER_UNAVAIL	CommEngine version is unavailable.
ECE_PARAM_INVALID	Parameter component passed is invalid.
< 0	Failed. See List of Error Codes .

Application must register to obtain version information. The version information is returned in the form “mm.nn.pp.bb” where:

- mm – Major number
- nn – Minor number
- pp – Patch number
- bb – Build number

For example “01.08.04.07” is valid version string. In this example, the size of this version string is 11. Using a buffer of reasonable size (ex. 32) to account for future changes or expansions is recommended.

Constants, Defines & Miscellaneous

Use the following information to determine constants, definitions, and other variables.

List of CommEngine Events

Table 40 contains the list of CommEngine events that an application is expected to receive. The event may optionally contain event data.

Table 40 List of CommEngine Events

Event ID	Event Value	Dist. Scope	Description
CE_EVT_START_FAIL	0x000A	Single App	Event posted to requesting application after failed <code>ceStartNWIF()</code> .
CE_EVT_STOP_NW	0x000B	Single App	Event posted to requesting application after successful <code>ceStopNWIF(CE_NETWORK)</code> .
CE_EVT_STOP_LINK	0x000C	Single App	Event posted to requesting application after successful <code>ceStopNWIF(CE_LINK)</code> .
CE_EVT_STOP_CLOSE	0x000D	Broadcast	Sent once the interface has been de-initialized. This is received after a successful <code>ceStopNWIF(CE_CLOSE)</code> or <code>ceStopNWIF(CE_DISCONNECT)</code> . This can also be received if there are errors encountered during <code>ceStartNWIF()</code> , <code>ceStopNWIF()</code> , or <code>ceStartDialIf()/ceStopDialIf()</code> if it is set to non-blocking mode
CE_EVT_STOP_FAIL	0x000E	Single App	Event posted to requesting application after failed <code>ceStopNWIF()</code> .
CE_EVT_DDI_APPL	0x000F	Subscribed	The extended event data contains the device driver application event data. For a mobile phone (GSM) / GPRS driver, this can also contain reasons for a failed start-up. The application event is stored in the event structure in the variable <code>neParam1</code> .
CE_EVT_NET_UP	0x0001	Broadcast	Sent when network is up and running.
CE_EVT_NET_DN	0x0002	Broadcast	Sent when the network is torn down.
CE_EVT_NET_FAILED	0x0003	Broadcast	Sent when attempting to establish the network, the connection failed. The event data contains the error code, the reason for failure. Error codes will be stored in <code>neParam1</code> and extended errors can also be extracted from <code>applExtEvtData</code> .
CE_EVT_NET_OUT	0x0004	Broadcast	The network is up but not active. This event is sent when there is either no coverage (cellular) or out of range (WiFi) or cable has

Table 40 List of CommEngine Events (continued)

Event ID	Event Value	Dist. Scope	Description
CE_EVT_NET_RES	0x0005	Broadcast	been removed (Ethernet). This is a temporary 'outage'. Network outage string can be extracted from <code>applExtEvtData</code> . The network is restored and is running. This event usually follows <code>CE_EVT_NET_OUT</code> . This event occurs after the outage has been fixed.
CE_EVT_SIGNAL	0x0006	Subscribed	The event data contains the signal strength in percentage stored in <code>neParam1</code> . Signal strengths are sent once device driver has been initialized and will stop once device driver has been released.
CE_EVT_START_OPEN	0x0007	Single App	Event posted to requesting application after successful <code>ceStartNWIF(CE_OPEN)</code> .
CE_EVT_START_LINK	0x0008	Single App	Event posted to requesting application after successful <code>ceStartNWIF(CE_LINK)</code> .
CE_EVT_START_NW	0x0009	Single App	Event posted to requesting application after successful <code>ceStartNWIF(CE_NETWORK)</code> or <code>ceStartNWIF(CE_CONNECT)</code> .
CE_EVT_START_DIAL	0x0010	Single App	Posted to requesting app after <code>ceStartDialIF()</code> if it is non-blocking. <code>neParam1</code> may contain the dial device handle or the error code in case of failure.
CE_EVT_NET_POOR	0x0011	Broadcast	The network is UP but is in a poor condition. This event is sent when the coverage is below the minimum threshold.
CE_EVT_PROCESSING_NET_UP	0x0012	Broadcast	The network interface has been started to fully establish a connection. This is triggered by a call to <code>ceStartNWIF(CE_CONNECT)</code> .
CE_EVT_PROCESSING_NET_DN	0x0013	Broadcast	The network interface has been started to fully teardown the connection. This is triggered by a call to <code>ceStopNWIF(CE_DISCONNECT)</code> .
CE_EVT_RECONNECT_PENDING	0x0014	Broadcast	Sent when there is a pending reconnection attempt for an NWIF that failed to connect.

Constants

The table below displays the constants, their values and description.

Table 41 Constants

Constant	Value	Description
Communication Technology		Communication Technology Descriptor String
CE_COMM_TECH_ETH	Ethernet	Ethernet
CE_COMM_TECH_WIFI	WiFi	Wireless Fidelity
CE_COMM_TECH_GPRS	GPRS	General Packet Radio Service
CE_COMM_TECH_CDMA	CDMA	Code Division Multiple Access
CE_COMM_TECH_PPPDIAL	PPP/Dial	Point to Point Protocol over Dial
CE_COMM_TECH_DIALONLY	Dial	Dial over phone line (POTS)
CE_COMM_TECH_PPPGSM	PPP/Mobile Phone (GSM)	Point to Point Protocol over Mobile Phone (GSM)
CE_COMM_TECH_GSMONLY	Mobile Phone (GSM)	Global system for mobile communications
CE_COMM_TECH_BT	Bluetooth	Bluetooth
CE_COMM_TECH_WIFI	WiFi	WiFi
CE_COMM_TECH_ETHBT	Ethernet/BT	Ethernet over Bluetooth
CE_COMM_TECH_PPPBT	PPP/BT	Point to Point Protocol over Bluetooth
CE_COMM_TECH_DIALBT	Dial/BT	Dial over Bluetooth
Component Version		Component Name for Version Information
CE_VER_CEIF	1	CommEngine Interface Library
CE_VER_VXCE	2	CommEngine
Starting the Connection		Start connection states.
CE_CONNECT	10	Establish the full connection (normal operation).
CE_OPEN	12	Open and prep the communication device.
CE_LINK	13	On PPP devices the data connection is established.
CE_NETWORK	14	Establish the network connection.

Table 41 **Constants** (continued)

Constant	Value	Description
Stopping the Connection		
CE_DISCONNECT	11	Disconnect and close the communication device (normal operation)
CE_NETWORK	14	Close the network connection.
CE_LINK	13	On PPP devices the data connection is torn down
CE_CLOSE	15	Close the device.
NWIF Start Mode		
CE_SM_AUTO	0	Start Mode - Automatic
CE_SM_MANUAL	1	Start Mode - Manual
Signal Notification Flag		
		Flag at which signal strength notifications are sent to applications by CommEngine.
CE_SF_ON	1	Signal Strength notification are sent.
CE_SF_OFF	0	No notifications are sent.
Run State		
		The run state of the network interface.
CE_RUN_STATE_FAILED	0	NWIF failed to start
CE_RUN_STATE_RUNNING	1	NWIF is successfully running
CE_RUN_STATE_READY	2	NWIF is not running and is pending start
CE_RUN_STATE_NOT_READY	3	NWIF cannot run as device is unavailable
Device Driver Type		
		The driver type of network interface.
CE_DRV_TYPE_ETHERNET	1	Driver type is Ethernet
CE_DRV_TYPE_PPP	2	Driver type is PPP
CE_DRV_TYPE_DIAL	3	Driver type is Dial
CE_DRV_TYPE_BT	4	Driver Type is Bluetooth
Pre-Dial Options		
		Controls <code>ceStartDialIF()</code> behavior
CE_DIALONLY_BLOCKING	0	<code>ceStartDialIF()</code> is blocking
CE_DIALONLY_NONBLOCKING	1	<code>ceStartDialIF()</code> is non-blocking

Table 41 **Constants** (continued)

Constant	Value	Description
NWIF Connection States		NWIF target or current states.
NWIF_CONN_STATE_INIT	0	DDI Driver is not yet loaded.
NWIF_CONN_STATE_OPEN	1	NWIF has now been initialized.
NWIF_CONN_STATE_LINK	2	NWIF has established data link connection.
NWIF_CONN_STATE_NET	3	TCP/IP is up and running.
NWIF_CONN_STATE_ERR	-1	An error has occurred.
Configuration Section Names		Config.sys variable prefix. <prefix>.<suffix>
CE_CT_CFG_PREFIX_ETH	ETH	Ethernet
CE_CT_CFG_PREFIX_PPPDIAL	PPPDIAL	PPP/Dial
CE_CT_CFG_PREFIX_DIAL	DIAL	Dial only
CE_CT_CFG_PREFIX_GPRS	GPRS	GPRS
CE_CT_CFG_PREFIX_BT	BT	Bluetooth
CE_CT_CFG_PREFIX_ETHBT	ETHBT	Ethernet/Bluetooth
CE_CT_CFG_PREFIX_PPPBT	PPPB	PPP/Bluetooth
CE_CT_CFG_PREFIX_DIALBT	DIALBT	Dial/Bluetooth
CE_CT_CFG_PREFIX_PPPGSM	PPPGSM	PPP/Mobile Phone (GSM)
CE_CT_CFG_PREFIX_WIFI	WIFI	WiFi
CE_CT_CFG_PREFIX_GSM	GSM	Mobile Phone (GSM) only
CE_CT_CFG_PREFIX_CDMA	CDMA	CDMA
Configuration Key Names		Config.sys variable suffix. <prefix>.<suffix>
CE_CT_CFG_SUFFIX_STARTMODE	startmode	Automatic (A) or Manual (M).
CE_CT_CFG_SUFFIX_DHCP	dhcp	DHCP (1) or Static (0).
CE_CT_CFG_SUFFIX_IPADDRESS	ipaddr	IP address to be used if static.
CE_CT_CFG_SUFFIX_SUBNETMASK	subnet	Subnet to be used if static.
CE_CT_CFG_SUFFIX_GATEWAY	gateway	Gateway to be used if static.
CE_CT_CFG_SUFFIX_DNS1	dns1	Primary DNS to be used if static.
CE_CT_CFG_SUFFIX_DNS2	dns2	Secondary DNS to be used if static.
CE_CT_CFG_SUFFIX_USERNAME	username	PPP Username.
CE_CT_CFG_SUFFIX_PASSWORD	password	PPP Password.
CE_CT_CFG_SUFFIX_AUTHTYPE	authtype	PPP Authentication Type.
Battery Threshold		These are the min and max values allowed.
CE_BATT_LEVEL_MIN	10	Minimum allowed. This is also the default.

Table 41 Constants (continued)

Constant	Value	Description
CE_BATT_LEVEL_MAX	100	Maximum allowed.
Device Driver Parameter		
FACTORY_DEFAULT	FACTORY_DEFAULT	Passed to <code>ceSetDDParamValue()</code> to revert to factory defaults.

List of Network Parameters

The table below lists the configuration parameters that can be either set or fetched using `ceAPI ceSetNWParamValue()` or `ceGetNWParamValue()`, respectively.

Table 42 Network Configuration Parameters

Parameter	Data Type	Get	Set	Description
AUTO_RECONNECT	Boolean	✓	✓	Set to 1 (true) for automatic reconnection. Set to 0 (false) for CE to remain in CE_OPEN state.
BT_CONFIG	stNi_BTConfig	✓	✓	See Configuration structure stNi_BTConfig sample code
BT_DIAL_DEVICE	Integer	✓	✓	Set to BT_VX680_BASE(1) to connect to a VX 680 Bluetooth base. Set to BT_MOBILE_PHONE(2) to connect to a mobile phone with DUN support. Terminal restart is required for the changes to take effect.
DSIM_AUTO_SWITCH_MODE	Integer	✓	✓	Enables / disables automatic switching. Valid values are: <ul style="list-style-type: none"> 0: Disabled (Default) 1: Enable switching on poor or no network and during connection failures. Also enables switching during GSM registration failures. During connection failures, auto reconnection is first completed with one SIM. When all attempts fail, auto reconnection is then completed with the other SIM. 2: Enable switching on poor or no network and during connection failures. Also enables switching during GSM registration failures. During connection failures, the SIM used is toggled during CE tier-1 reconnection. Note: When mode is 1 and RECONN_TO is 0, switching will never occur during connection failures since reconnection attempts on one SIM is infinite. When mode is 2 and RECONN_MAX is 0, switching will never occur since tier-1 reconnect is not performed.

Table 42 Network Configuration Parameters (continued)

Parameter	Data Type	Get	Set	Description
DSIM_AUTO_SWITCH_NET_RES_INTERVAL	Integer	✓	✓	<p>Once terminal enters poor or no network, this is the time in seconds to wait before switching to the other SIM. The use case for this parameter are described below:</p> <ul style="list-style-type: none"> When terminal stays in poor or no network until the wait time completes, switching is performed. <p>If terminal enters good network before the wait time elapses, switching is not performed.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> Minimum: 5. Maximum: 3600. Default: 20.
DSIM_AUTO_SWITCH_REVERT_INTERVAL	Integer	✓	✓	<p>Once terminal switches to the other SIM but both SIM has actually entered poor or no network, this is the time in seconds to wait before switching back to the previous SIM. The use case for this parameter are discussed below:</p> <ul style="list-style-type: none"> When terminal stay in poor or no network until the wait time elapses, switching back to the previous SIM is performed. If terminal enters good network before the wait time elapses, switching back to the previous SIM is not performed. <p>Valid values are:</p> <ul style="list-style-type: none"> Minimum: 5. Maximum: 3600. Default: 60.
DSIM_AUTO_SWITCH_REVERT_TO	Integer	✓	✓	<p>When terminal is in an area where both SIM is in poor or no network, this is the time in seconds the automatic switching operation is repeatedly performed after the initial switch has failed.</p> <p>When terminal stays in poor or no network on both SIM until the timeout elapses, the network interface is torn down. Application needs to manually restart the network interface.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> Minimum: 15 Maximum: 7205 Default: 125
IP_CONFIG	stNI_IPConfig	✓	✓	See Configuration structure stNI_IPConfig sample code.
IP_CONFIG2	stNI_IPConfig	✓	✓	Configure IP settings for SIM in slot 2.
NWIF_STATE	stNI_NWIFState	✓	-	See Configuration structure stNI_NWIFState sample code.

Table 42 Network Configuration Parameters (continued)

Parameter	Data Type	Get	Set	Description
PPP_CONFIG	stNI_PPPConfig	✓	✓	See Configuration structure stNI_PPPConfig sample code .
PPP_CONFIG2	stNI_PPPConfig	✓	✓	Configure PPP settings for SIM in slot 2.
SIGNAL_LEVEL	stSignalLevel	✓	-	See Configuration structure stSignalLevel sample code .

NOTE



The list of network parameters is not exhaustive and will expand based on requirement. Verix eVo TCP/IP API provides a high degree of configurability. However not all options are applicable.

Example

Configuration structure sample codes are provided in the section below.

Configuration structure stNI_IPConfig sample code

```
typedef struct
{
    unsigned int ncDHCP;           // 1 = DHCP; 0 = Static
    unsigned long ncIPAddr;        // Mandatory if Static IP
    unsigned long ncSubnet;        // Mandatory if Static IP
    unsigned long ncGateway;       // Mandatory if Static IP
    unsigned long ncDNS1;          // Optional but usually required
    unsigned long ncDNS2;          // Optional
    char dhcpLeaseStartTime[14];   //YYYYMMDDHHMMSS format
                                   (read only)
    char dhcpLeaseEndTime[14];    //YYYYMMDDHHMMSS format
                                   (read only)
    unsigned int ncIsConnected;    //Yes or No (read only)
} stNI_IPConfig;
```

Variable `ncIsConnected` shows if an IP address was successfully obtained by CommEngine. It does not reflect CE's current state. Its value may differ on a `CE_EVT_NET_OUT` event depending on the device driver type.

Note the following scenarios:

- For an Ethernet type of driver:
Once the cable is unplugged, `ncIsConnected` will become false as the network connection is lost. CommEngine will try to reestablish the connection once the cable is plugged back in and the IP will be reacquired as a different LAN cable may have been plugged.
- For a GPRS device driver:

Once the driver detects that the signal has been lost, `CE_EVT_NET_OUT` will be sent out by CommEngine but `ncIsConnected` will still be true as the network was not torn down.

Configuration structure stNI_PPConfig sample code

```
typedef struct
{
    pppAuthType ncAuthType;        //Type of PPP authentication
    char ncUsername[MAX_USERNAME]; //PPP username, optional
    char ncPassword[MAX_PASSWORD]; //PPP password, optional
} stNI_PPConfig;
```

Configuration structure stNI_BTConfig sample code

```
typedef struct
{
    char devPort[16]; //Bluetooth device port to use
} stNI_BTConfig;
```

Configuration structure stSignalLevel sample code

```
typedef struct
{
    int iSignal1;    //signal in percentage
    int iSignal2;    //signal in dBm
    int iSignal3;    //signal in RSSI
} stSignalLevel;
```

Configuration structure stNI_NWIFState sample code

```
typedef struct
{
    int nsTargetState;           // Target State. See NWIF_CONN_STATE_*.
    int nsCurrentState;         // Current State. See NWIF_CONN_STATE_*.

    unsigned int nsEvent;       // Last Broadcast event

    int nsErrorCode;            // Error / failure code if an error occurred, zero otherwise.

    char nsErrorString[MAX_ERR_LEN]; // Holds the last known error string (if a CE_EVT_NET_
                                     // FAILED is sent) or network outage string (if a CE_EVT_
                                     // NET_OUT or CE_EVT_NET_POOR is sent) or network
                                     // restore string (if a CE_EVT_NET_RES is sent). Null if there
                                     // is no error.

    char nsTimeStamp[MAX_DATE_LEN+1]; // Timestamp for when the last event was sent. Timestamp
    } stNI_NWIFState;               // in YYYYMMDDhhmmss format. This is a null terminated
                                     // string.
```

List of Error Codes

Error codes returned by ceAPI are negative (< 0). ceAPIs are designed to return a non-negative return value (≥ 0) if the operation is successful. The return values are documented with each API. Negative return values (< 0) are listed in [Table 43](#).

Table 43 List of ceAPI Error Codes

Error ID	Error Value	Description
ECE_SUCCESS	0	Success return code
ECE_NO_MEMORY	-1000	No memory
ECE_REGAGAIN	-1001	Application is attempting to register again after a successful registration. If necessary unregister by calling ceAPI <code>ceUnregister()</code> and register again using API <code>ceRegister()</code> .
ECE_REGFAILED	-1002	Registration failed as CommEngine is not running.
ECE_CREATEPIPE	-1003	Application creates a pipe as part of the registration process. This error is returned if the API has failed to create a pipe.
ECE_CREATEMON	-1004	Application creates a monitoring thread as part of the registration. This error occurs if the monitor task has failed.
ECE_SENDCEREQ	-1005	Application unable to send request event to CommEngine.
ECE_RESPFAIL	-1006	Application failed to receive a response.
ECE_TIMEOUT	-1007	No response, timeout error.
ECE_NOTREG	-1008	Application has not successfully registered with CommEngine. This error returned if an application is attempting a ceAPI prior to successful registration.
ECE_DEVNAMELEN	-1009	Device name length is beyond limits. Verify device name is accurate and null terminated.
ECE_PARAM_NULL	-1010	Parameter is null when one is not expected.
ECE_PARAM_INVALID	-1011	Parameter is invalid or out of range.
ECE_VXNCP_PIPE	-1012	Pipe VxNCP is not running.
ECE_VXNCP_PIPECR	-1013	Unable to create temp pipe to create with VxNCP.
ECE_VXNCP_CONSOLE	-1014	Application is not console owner.
ECE_VER_UNAVAIL	-1015	CommEngine version information is unavailable.
ECE_CDMAPP	-1016	Application has failed to register as CommEngine's communication device management application. Either another

Table 43 List of ceAPI Error Codes (continued)

Error ID	Error Value	Description
ECE_NOCDM	-1017	<p>application has successfully registered or this application is attempting more than once. See API <code>ceSetCommDevMgr ()</code> for additional details.</p> <p>Application is attempting to register itself as CommEngine's communication device management application when none is required. See API <code>ceSetCommDevMgr ()</code> for additional details.</p>
ECE_DEVNAME	-1018	Unknown device name. This is not a communication device that CommEngine is aware of.
ECE_DEVOWNER	-1019	CommEngine is not the owner of the device. This is a communication device that CommEngine is aware of but currently not in its possession.
ECE_DEVBUSY	-1020	CommEngine is currently the owner of this device. CommEngine is unable to release this device as it is busy. This error is returned if there are open sockets.
ECE_NOTASKID	-1021	Task Id provided as parameter does not exist.
ECE_NODEVREQ	-1022	No prior device request was made. Nothing to cancel.
ECE_CANNOT_UNREG	-1023	<p>Application cannot unregister for one of the following reasons:</p> <ul style="list-style-type: none"> • Application has registered itself as CommEngine's device management application • This application has devices checked out or has pending requests for devices. Check in all devices or cancel all requests before attempting to unregister again.
ECE_NOT_SUPPORTED	-1024	The feature requested is not supported. Check version information and releases notes of ceAPI and CommEngine.
ECE_NWIF_BUSY	-1025	The configuration cannot be changed as the network interface is in a run state. To change configuration -- stop, change configuration and restart.

Table 43 List of ceAPI Error Codes (continued)

Error ID	Error Value	Description
ECE_DD_NOT_READY	-1026	The Device Driver is not in a state to provide response to parameter requests. In the case of an <code>ioctl()</code> command, CommEngine is expecting the NWIFs state to at least be in OPEN before it proceeds.
ECE_NO_CE_EVENT	-1027	No CommEngine Notification Event in queue or Event Notification was not enabled.
ECE_EX_CMD_FAILED	-1028	Execute Command Failed.
ECE_NOTDEVOWNER	-1029	Application did not check out this device or application is attempting to check if the device did not check out previously.
ECE_PARAM_NOT_FOUND	-1030	Parameter requested is not supported.
ECE_PARAM_READ_ONLY	-1031	Parameter is a read only. Parameter can not be set.
ECE_PARAM_SAVE_FAIL	-1032	Saving to Delta file failed.
ECE_STATE_ERR	-1033	<code>ceStart/StopNWIF()</code> state error. Target mode passed might be less than CommEngine's current state.
ECE_EVENTS_AGAIN	-1034	Application is attempting to enable events again after a successful event enabling.
ECE_DIAL_ONLY_FAILED	-1035	<code>ceStartDialIF()</code> failed.
ECE_SUBNET_MASK_INVALID	-1036	Invalid subnet mask was passed.
ECE_START_STOP_BUSY	-1037	A previous call to <code>ceStartNWIF()</code> or <code>ceStopNWIF()</code> is still being processed.
ECE_PPP_AUTH_FAILURE	-1038	PPP Authentication failure. This is returned in as a value of <code>neParam1</code> in the structure <code>stceNWEvt</code> .
ECE_INIT_FAILURE	-1039	This is returned when DDI driver can not be initialized.
ECE_SIGNAL_FAILURE	-1040	This is returned when query of signal level failed.
ECE_TCPIP_STACK_FAILURE	-1041	This is returned in as a value of <code>neParam1</code> in the structure <code>stceNWEvt</code> when a TCP/IP Stack error occurred. Possible reasons are: <ul style="list-style-type: none"> • TCP/IP stack APIs returned an error. • A PPP stack event of <code>LL_OPEN_FAILED</code> was received.
ECE_CONNECTION_LOST	-1042	This is returned in as a value of <code>neParam1</code> in the structure <code>stceNWEvt</code> when a connection lost happens.

Table 43 List of ceAPI Error Codes (continued)

Error ID	Error Value	Description
ECE_ETHERNET_HOSTDOWN	-1043	This is returned in as a value of neParam1 in the structure stceNWEvt when Ethernet Renewal failed because the DHCP renewal re-trans-mechanism timed out (and the server never responded).
ECE_ETHERNET_REBIND_TIMEOUT	-1044	This is returned in as a value of neParam1 in the structure stceNWEvt when rebind failed or the address expired before the rebind completed.
ECE_BATTERY_THRESHOLD_FAILURE	-1045	This is returned in the event that OS API set_battery_value() fails or if no battery is attached to the terminal.
ECE_PPP_TIMEOUT_FAILURE	-1046	This is returned as a value of neParam1 in the structure stceNWEvt.
ECE_IOCTL_FAILURE	-1047	This is returned if a call to ddi_ioctl() returned error.
ECE_RESTORE_DEFAULT_FAILURE	-1048	This is returned if restoring factory default has failed. Probable cause is due to IOCTL to restore defaults returned failure.
ECE_PARAM_WRITE_ONLY	-1049	Parameter is write only.

List of CE Error Codes with Corresponding Error Strings

When receiving CE Events, there are cases when neParam1 may be populated with an error code and application extended data with error string. This may be populated by CE or by DDI Driver / Manager. [Table 44](#) lists CE event with error codes and error strings.

Table 44 CE Event with Error Codes and Error Strings

Error ID	Error Code	Error String ID	Error String	Description
ECE_DIAL_ONLY_FAILED	-1035	ERR_DRIVER_INTERNAL_FAILURE	Driver failure	IOCTL call to get extended error failed.
ECE_PPP_AUTH_FAILURE	-1038	ERR_PPP_AUTH_FAILURE	PPP Auth Fail	This is received when an Authentication Failure happens.
ECE_DIAL_ONLY_FAILED	-1039	ERR_DLL_LOAD_ERROR	DLL Load Error	Call to ddi_init_dial() resulted in a failure.
ECE_TCPIP_STACK_FAILURE	-1041	ERR_TCPIP_FAILURE	TCP/IP failure	TCP/IP API net_addif() or AddInterface() returned failure

Table 44 CE Event with Error Codes and Error Strings (continued)

Error ID	Error Code	Error String ID	Error String	Description
		ERR_TCPIP_LANDLINE_OPEN_FAILURE	TCP PPP Open Fail	TCP/IP returned PPP event LL_OPEN_FAILED.
ECE_CONNECTION_LOST	-1042	ERR_CONNECTION_LOST	Connection Lost	Connection Lost was detected by DDI Driver.
ECE_ETHERNET_HOSTDOWN	-1043	ERR_ETH_HOSTDOWN	ETH Host Down	TCP/IP returned ETH event EHOSTDOWN.
ECE_ETHERNET_REBIND_TIMEOUT	-1044	ERR_ETH_TIMEOUT	ETH Timeout	TCP/IP returned ETH event ETIMEDOUT.

Extended Error Codes and Messages

When the DDI Driver encounters an error in its operation, a specific error is generated to CommEngine. The information passed from the driver is in two forms: an extended error code; and an extended error message.

The extended error code is a negative integer that numerically represents the type of error that has occurred. It can be used by applications to check what kind of DDI Driver error is passed from the driver through CommEngine. On the other hand, the extended error message is a short string that described the error that can be used by the application for display purposes (on the screen).

Table 45 lists the extended error codes generated by the DDI Driver together with their corresponding error message. For the application to use the extended error codes, it should include `vos_ddi_ext_err.h`, which is found in the Verix eVo SDK directory under the `include` sub-directory.

Table 45 Common Error Codes and Messages

Error Code (Macro)	Error Messages	Description
E_DDI_INVL_IOCTL	IOCTL not supported	The IOCTL request is not recognized by the DDI driver.
E_DDI_PORT_OPEN_FAIL	Port open fail	Communication port not opened successfully.
E_DDI_STAT_PORT_OPEN_FAIL	Stat port open fail	Status port not opened successfully.
E_DDI_NO_IOCTL_ID	No IOCTL ID	The IOCTL request ID is not set.
E_DDI_AT_CMD_FAIL	AT command fail	A generic failure when the DDI Driver sends an AT command.
E_DDI_INI_PARAM_NOT_FOUND	Missing INI param	A necessary INI parameter is not listed in the INI file (master or delta) and or INI parameter is not loaded into memory.
E_DDI_STAT_PORT_CLOSE_FAIL	Stat port close err	Status port not closed successfully.
E_DDI_PORT_CLOSE_FAIL	Port close fail	Communication port not closed successfully.
E_DDI_NO_CARRIER	NO CARRIER	A generic NO CARRIER modem response.
E_DDI_GENERIC_ERROR	ERROR	A generic ERROR modem response.
E_DDI_NO_DIAL_TONE	NO DIAL TONE	A generic NO DIAL TONE modem response.
E_DDI_BUSY	BUSY	A generic BUSY modem response.

Table 45 Common Error Codes and Messages (continued)

Error Code (Macro)	Error Messages	Description
E_DDI_NO_ANSWER	NO ANSWER	A generic NO ANSWER modem response.
E_DDI_INVL_IOCTL_DATA	Invalid IOCTL data	The IOCTL request data may be structured incorrectly, set to an incorrect data type, or format is invalid.
E_DDI_NO_IOCTL_DATA	No IOCTL data	The IOCTL request data is not set.
E_DDI_INIT_AT_CMD_FAIL	Init AT cmd fail	A generic initialization AT command failure.
E_DDI_START_AT_CMD_FAIL	Start AT cmd fail	A generic start-up AT command failure.
E_DDI_PORT_NOT_SET	Port not set	Communication parameters are not properly set.
E_DDI_DISCONNECTION_FAIL	Disconnection fail	A generic disconnection failure.
E_DDI_NO_CONNECTION_BAUD	No connection baud	When an IOCTL call for IOCTL_GET_CONNECTION_BAUD is made and the connection baud is unknown.
E_DDI_NOT_CONNECTED	Not connected	When an IOCTL call for IOCTL_GET_CONNECTION_BAUD is made and the device is not connected.
E_DDI_LOW_BATT	Low battery level	The terminal has reached a critical low battery level.
E_DDI_CONNECT_TIMEOUT	Connect time out	The connection attempt passes the allotted time for the driver to return a failure.
	WPA key incorrect	For WiFi only. All connection attempts failed due to wrong WPA key.
	Authentication timeout	For WiFi only. All connection attempts failed due to wrong WEP key.
E_DDI_POOR_NET_RESTORE_FAIL	Poor net ret expire	Poor network retries were exhausted
E_DDI_NO_NET_RESTORE_FAIL	No net retry expire	No network retries were exhausted
E_DDI_INVL_DDI_RESP	Invalid response	The response from the driver is not recognized by DDI Manager
E_DDI_OPEN_STATE	Open state error	Open state is requested without closing the device first.
E_DDI_INIT_STATE	Init state error	Init state is requested without deinitializing the device first.
E _DDI_DEINIT_STATE	Deinit state error	Deinit state is requested without closing the device first.
E_DDI_CLOSE_STATE	Close state error	Close state is requested without opening the device first
E_DDI_SIM_PIN_FAIL	Incorrect SIM PIN	Wrong SIM PIN is entered to the modem.

Table 46 lists the Dial error codes and messages.

Table 46 Dial Error Codes and Messages

Error Code (Macro)	Error Messages	Description
E_DDI_BAUDRATE_NOT_SUPPORTED	Baud not supported	The baud rate for the serial line is not supported.
E_DDI_CALL_ABORT_FAIL	Call abort fail	Aborting the dial call failed.
E_DDI_CONNECT_RESPONSE_ERROR	Connect resp error	If the string length of the modem response is greater than the response buffer upon dialing.
E_DDI_DCD_SIGNAL_NOT_SET_TO_ON	DCD not set to on	DCD signal did not go to ON after a successful connect response.
E_DDI_FORMAT_NOT_SUPPORTED	Fmt not supported	The format for the serial line is not supported.
E_DDI_INVALID_DIAL_TYPE	Invalid dial type	Invalid dial type. If the dial type is set outside the range of 0-3.
E_DDI_LINE_IN_USE	LINE IN USE	The telephone line is currently in use
E_DDI_NO_LINE	NO LINE	No phone line cable connected
E_DDI_NO_PRIMARY_NUMBER	No primary number	Primary number is not set.
E_DDI_RESPONSE_CODE_NOT_FOUND	Resp code not found	If the connect response code cannot be found.
E_DDI_SDLC_INITIALIZATION_FAIL	SDLC init fail	SDLC initialization failed.
E_DDI_SWITCH_TO_SDLC_FAIL	Switch to SDLC fail	Switch from asynchronous to synchronous communication failed.

Table 47 lists the Ethernet error codes and messages.

Table 47 Ethernet Error Codes and Messages

Error Code (Macro)	Error Messages	Description
E_DDI_GET_MAC_FAIL	Getting of MAC fail	Getting of MAC failed.
E_DDI_SET_RX_FAIL	Set rx control fail	Setting of rx_control failed.
E_DDI_INVL_NET_STAT	Invalid net status	Invalid Ethernet link value.

Table 48 lists the GPRS and mobile phone (GSM) error codes and messages.

Table 48 GPRS and Mobile Phone (GSM) Error Codes and Messages

Error Code (Macro)	Error Messages	Description
E_DDI_GSM_MUX_NOT_OPEN	GSM MUX not open	Mobile phone (GSM) multiplexer protocol is not successfully turned on.
E_DDI_CYCLIC_POW_SAVE	No cyclic powersave	Cyclic power saving is not successfully set in the GPRS/mobile phone (GSM) modem.
E_DDI_URC_DISABLED	URC events disabled	URC events are not successfully enabled in the modem.

Table 48 GPRS and Mobile Phone (GSM) Error Codes and Messages (continued)

Error Code (Macro)	Error Messages	Description
E_DDI_URC_NOT_REG	URC not registered	URC events are not properly registered to the modem.
E_DDI_PDP_NOT_SET	PDP context not set	PDP context is not properly set in the modem. May indicate that the APN is incorrect.
E_DDI_NO_BATT	No battery detected	Battery is removed from the terminal.
E_DDI_NO_SIM	No SIM card	SIM card is not present.
E_DDI_AUTO_BAUD_FAIL	Auto-baud fail	Modem not responding to any acceptable baud rate.
E_DDI_BAUD_NOT_SET	Baud rate not set	Baud rate is not properly set in the modem.

Table 49 lists Bluetooth error codes and messages.

Table 49 Bluetooth Error Codes and Messages

Error Code (Macro)	Error Messages	Description
E_DDI_DEV_INQUIRY_FAIL	Inquiry fail	Device search/inquiry fails
E_DDI_PAIR_FAIL	BT Pair fail	Pairing to a remote BT device was not successful
E_DDI_PAIR_TIMEOUT	Pair timeout	When pairing reaches the maximum pair timeout
E_DDI_INITIATE_PAIRING_FAIL	Initiate pairing fail	When the BT stack fails when initiating the pairing process
E_DDI_PAIR_AUTHENTICATION_FAIL	Pair authentication fail	When the link key passed does not match the link key saved on the remote BT device or when the BT device is not pairable
E_DDI_BT_ADDRESS_NOT_PRESENT	BT Address not present	No Bluetooth device specified when pairing
E_DDI_GET_SEARCH_COUNT_FAIL	Get inquiry count fail	When getting the inquiry count fails
E_DDI_GET_PAIR_COUNT_FAIL	Get pair count fail	When getting the pair count fails
E_DDI_REMOVE_PAIR_FAIL	Remove pair fail	When removing a paired device fails
E_DDI_SET_CONNECT_INFO_FAIL	Set connect info fail	When setting the port configuration fails
E_DDI_GET_SEARCH_RESULT_FAIL	Get inquiry result fail	When getting the inquiry result info fails
E_DDI_GET_PAIR_DEVICES_FAIL	Get pair devices fail	When getting the paired device info fails
E_DDI_EXCEEDS_PAIR_MAX	Exceeds pair max	It exceeds the maximum paired devices
E_DDI_CREATE_PIPE_FAIL	Create pipe fail	When creating a pipe fails

Table 49 Bluetooth Error Codes and Messages (continued)

Error Code (Macro)	Error Messages	Description
E_DDI_CLOSE_PIPE_FAIL	Close pipe fail	When closing a pipe fails
E_DDI_END_BOND_FAIL	End bond fail	Ending a bond was not successful
E_DDI_BT_CONNECT_FAIL	BT connect fail	When bonding and connecting to the remote BT device fails
E_DDI_LOAD_MDM_PROFILE_FAIL	Load mdm profile fail	When the modem profile was not loaded successfully
E_DDI_REMOTE_BT_DEVICE_NOT_READY	Remote BT device not ready	When pairing to an unplugged device
E_DDI_BT_FIRMWARE_UPGRADE_FAIL	Firmware upgrade fail	When the firmware was not loaded successfully to the remote BT device

Application Developer Notes

This section covers two topics of interest to Application Developers. [Handling CommEngine Events](#) describes how applications can manage events coming from CommEngine.

Handling CommEngine Events

Managing and Controlling the Connection

In this sample, 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.

This sample code illustrates how an application enables events using `ceAPI ceEnableEventNotification()` and manages them.

[Handling_CommEngine_Events.txt](#)

(Click to view file attachments)

CE_EVT_DDI_APPLICATION for a GPRS / Mobile Phone (GSM) Driver

Some device drivers may send application events directly to the application. In the case of a GPRS / mobile phone (GSM) driver, it sends strings to application stating reasons for failure. The sample code below shows an example on how applications can extract the raw data sent with `CE_EVT_DDI_APPL`.

[CE_EVT_DDI_APPL_GPRS_GSM.txt](#)

(Click to view file attachments)

Sample code on `ceSetNWParamValue()` / `ceGetNWParamValue()`

The sample code below illustrates how applications can set network parameters using `ceSetNWParamValue()` and get network parameters using `ceGetNWParamValue()`.

Function `SetParametersEthernet()` illustrates how applications can set static ip configurations.

Function `SetParametersPPP()` illustrates how applications can set username, password and authentication type for PPP connections.

[ceSetNWParamValue_ceGetNWParamValue.txt](#)

(Click to view file attachments)

**Sample code on
`ceSetDDParamValue()`
/
`ceGetDDParamValue()`**

The sample code below illustrates how applications can set device driver parameters using `ceSetDDParamValue()` and get device driver parameters using `ceGetDDParamValue()`.

[ceSetDDParamValue_ceGetDDParamValue.txt](#)

(Click to view file attachments)

**Sample code on
`ceEXCommand()`**

Sample code below illustrates how applications can send `ioctl()` calls directly to device driver using `ceExCommand()`.

[ceEXCommand.txt](#)

(Click to view file attachments)

**Sample code on
`ceStartDialIF()`**

Sample code below illustrates how `ceStartDialIF()` works in blocking and non-blocking mode.

[ceStartDialIF_Blocking.txt](#)

(Click to view file attachments)

[ceStartDialIF_NonBlocking.txt](#)

(Click to view file attachments)

Message Exchange mxAPI & Introduction

The Message Exchange API is designed for applications to communicate via pipes. Any set applications intending to communicate with each other can take advantage of this API. For example, for application A and application B to communicate with each other, each application creates a pipe and communicates with each other by sending and receiving messages.

Payload Size

The Payload field is the application data that is sent by the sending application to the receiving application. The contents and size are provided by the sending application. This field is considered as a sequence of n bytes, where n is the size of the payload. The maximum payload is 4096 bytes.

Message Exchange mxAPI – Summary

The Message Exchange API consists of the functions listed in the table below. The following section provides detailed description of each API.

API	Description
<code>mxCreatePipe()</code>	Creates a message pipe
<code>mxClosePipe()</code>	Closes the pipe created by API <code>mxCreatePipe()</code>
<code>mxGetPipeHandle()</code>	Obtain pipe handle associated with pipe name
<code>mxSend()</code>	Send message to destination pipe
<code>mxPending()</code>	Determines number of messages in the queue that are pending read
<code>mxRecv()</code>	Reads pending message from queue
<code>mxRecvEx()</code>	Reads pending message and extended pipe header information from queue

mxCreatePipe()

Creates a Verix eVo message pipe and returns handle to pipe.

Prototype `int mxCreatePipe(const char *pipeName, const short messageDepth);`

Parameters

In:	pipeName	Name of pipe to create. pipeName is up to 8 characters long. The pipe name must have the P: prefix. Pipe name may be "P:" for anonymous pipe.
In:	messageDepth	Maximum number of incoming messages buffered before they are read. messageDepth must be greater than 0 and less than or equal to 10.

Return Values

> = 0	Success. Pipe created and handle returned
< 0	See mxAPI Error Codes .

mxClosePipe()

Close Verix eVo message pipe created via `mxCreatePipe()`.

Prototype `int mxClosePipe(const int pipeHandle);`

Parameters

In:	<code>pipeHandle</code>	Pipe handle returned by <code>mxCreatePipe()</code> .
-----	-------------------------	---

Return Values

<code>= 0</code>	When the pipe is closed successfully.
<code>< 0</code>	When the pipe handle is not valid.

mxGetPipeHandle()

Obtains pipe handle associated with pipe name. Use this API to obtain the handle of a named pipe. API `mxSend()` requires the handle of the destination pipe.

Prototype `int mxGetPipeHandle(const char *pipeName);`

Parameters

In:	pipeName	Target pipe name whose handle to be obtained. Only the first 8 characters are considered in the comparison. The pipe name cannot be NULL or an empty string ("").
-----	----------	---

Return Values

> = 0	Pipe handle. API <code>mxGetPipeHandle()</code> fetches the handle associated with a pipe created with name <code>pipeName</code> .
< 0	<code>pipeName</code> does not match currently open pipes. See mxAPI Error Codes .

mxSend()

Sends message to destination pipe.

Prototype `int mxSend(const int pHSrc, const int PHDest, const char *dataPayload, const short dataLength);`

Parameters

In:	pHSrc	Source pipe handle. Created using <code>mxCreatePipe()</code> .
In:	PHDest	Destination pipe handle. Obtained handle via <code>mxGetPipeHandle()</code> or provided by other means.
In:	dataPayload	Data to send to destination pipe.
In:	dataLength	Size of data in dataPayload. Size cannot exceed 4096 bytes.

Return Values

> = 0	Returns dataLength. Returns length value on successful send.
< 0	See mxAPI Error Codes .

mxPending()

Determines number of messages in the queue that are pending read.

Prototype `int mxPending(const int pH);`

Parameters

In: pH Pipe handle. Created using `mxCreatePipe()`.

Return Values

> = 0 Count of number of unread / pending messages. Will return zero if no messages are pending.

< 0 If the handle is not valid. See [mxAPI Error Codes](#).

mxRecv()

Reads pending message from queue.

Prototype

```
int mxRecv(const int pH, int *pHFrom, void *dataPayload, const unsigned
int dataPayloadSz, unsigned int *dataPayloadLen);
```

Parameters

In:	pH	Pipe handle. Created using <code>mxCreatePipe()</code>
Out:	pHFrom	Sender pipe handle
Out:	dataPayload	Pointer to buffer to receive payload data from sender
In:	dataPayloadSz	Size of buffer <code>dataPayload</code> in bytes
Out:	dataLoadLen	Length of data in payload buffer

Return Values

> 0	Message read and number of bytes in payload
= 0	Message read and has no payload
< 0	Read error or no pending message. See mxAPI Error Codes

NOTE



API `mxRecv()` does not wait for incoming messages. It checks the queue and if one is present reads it and returns. This is a non-blocking call. Applications can wait on pipe event and then read the waiting message using this API.

mxRecvEx()

Reads pending message from queue. Structure `pipe_extension_t` is defined in VFSDK file `svc.h`. Refer to [mxRecv\(\)](#) for additional details.

Prototype

```
int mxRecvEx(const int pH, pipe_extension_t *extPI, void *dataPayload,
const unsigned int dataPayloadSize, unsigned int *dataPayloadLen);
```

Parameters

In:	pH	Pipe handle. Created using <code>mxCreatePipe()</code>
Out:	dataPayload	Pointer to buffer to receive payload data from sender
In:	dataPayloadSz	Size of buffer dataPayload in bytes
Out:	dataLoadLen	Length of data in payload buffer

Return Values

> 0	Message read and number of bytes in payload
= 0	Message read and has no payload
< 0	Read error or no pending message. See mxAPI Error Codes .

mxAPI Error Codes

mxAPI errors are negative and returned by the API. The list presented here is not exhaustive and is subject to change.

Error ID	Error Value	Description
EMX_UNKNOWN	-2000	Undetermined error
EMX_PIPE_NAME	-2001	Pipe name too long. May not exceed 8 characters. Pipe name has invalid non-ASCII characters.
EMX_MSG_DEPTH	-2002	The message depth parameter should be in the range 1 to 10. Any value outside this range will result in this error.
EMX_PIPE_NOMEM	-2003	No memory to create pipe.
EMX_PIPE_MATCH	-2004	Pipe name does not match currently open named pipe(s)
EMX_PIPE_HANDLE	-2005	Pipe handle not valid
EMX_PAYLOAD_SIZE	-2006	Payload size should be positive non-zero value and less than limit
EMX_NULL_PARAM	-2007	Input parameter is unexpected null
EMX_DEST_PIPE	-2008	Target pipe for <code>mxSend()</code> does not exist or not configured for messages
EMX_PIPE_FULL	-2009	The destination pipe is full and the message cannot be sent

Message Formatting mfAPI

Applications need to exchange data. To successfully do so, participating applications must have a common format. Applications use Message Exchange API (mfAPI) format data as a TLV list for exchange.

The data exchanged between applications usually consists of a list of name value pairs. This combination of name and value is referred to as TLV or its expansion Tag, Length and Value. TLV is sometimes referred to as *Type*, Length and Value.

The Message Formatting API provides the necessary functionality to create a TLV list and the reverse, i.e., convert a TLV List to individual TLVs. Consider two applications, a client and server application. The client application obtains service from the server application by sending a request message and the server application responds by sending the response message. The client application creates a request message as a TLV list and dispatches it to the server application. The server sends a response message as a TLV list and the client application converts it to individual TLVs. The *mfAPI* provides API for constructing the request message as a TLV list and to extract individual TLVs from the response message.

Tag, Length & Value (TLV)

A TLV consists of a fixed size Tag name size field followed by a variable length Tag name field. This pair of fields is followed by a fixed size Value length field and terminated by the variable length Value field. Representing it as a sequence of fields:

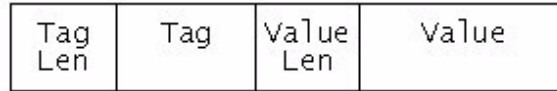


Figure 4 Fixed Length TLV

The Tag Length and Value Len fields are fixed sizes while the Tag and Value fields are variable size. For example if Tag name length is 9 the size of the Tag is 9. Taking this example further, the string "CITY_NAME" is 9 characters long.

The Value field is a variable length field of data whose size is dictated by the Length field.

Consider this example:

```
#define TAG_CITY_NAME "CITY_NAME" // Tag for city name, 9bytes
```

The city name tag is "CITY_NAME" and its value is "BOSTON" which is 6 characters long and the length is 6. The TLV for this example represented as:

```
09 | CITY_NAME | 06 | BOSTON
```

The '|' separator and spaces are only for readability.

TLV List This section describes the message format. The TLV list is a sequence of TLVs preceded by a Header (Figure 5).

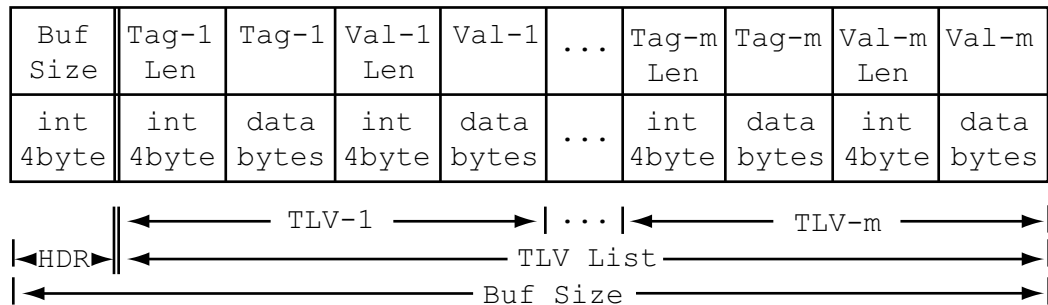


Figure 5 Fixed length TLV list

The header consists of one field:

Field Name	Data Type	Size	Description
Buffer Size	int	sizeof(int) 4bytes	Size of the Message - computed by adding the size of the header (4 bytes) to the size of the TLV List. For example, if the size of TLV List is 96 bytes then the value of this field is 100bytes.
		Total	4 bytes

Message Formatting mfAPI – Summary

The Message Formatting API consists of the functions listed in the table below. The following section has detailed descriptions of each API.

Table 50 Message Formatting APIs

API	Description
Handle Management	
mfCreateHandle	Obtains a new handle
mfDestroyHandle	Releases a handle previously created by mfCreateHandle().
Add Operations	
mfAddInit()	Initializes the Message buffer for Add operations
mfAddClose()	Closes the session and returns length of Message buffer.
mfAddTLV()	Adds TLV to the buffer.
Fetch Operations	
mfFetchInit()	Initializes the Message buffer for Fetch (read) operations and returns the session handle.
mfFetchClose()	Closes the session for Fetch operations.
mfFetchReset()	Repositions the "Next" pointer to the top of the TLV list.
mfPeekNextTLV()	Obtains the next Tag length and the size of its Value. The "Next" pointer is not advanced.
mfFetchNextTLV()	Fetches the next TLV in sequence from list. Moves the "Next" pointer to the next TLV in the list.

Table 50 **Message Formatting APIs** (continued)

API	Description
mfFindTLV()	Fetches the first TLV from list that matches tag.
Helper Function	
mfEstimate()	Computes memory requirement for a set of tags.

mfCreateHandle()

Obtains a new handle.

Prototype `void *mfCreateHandle(void);`

Return Values

hF (non-zero)	Operation successful. Pointer to session handle.
0	Operation failed.

NOTE



This must be the first API. All other API require session handle (hF). Use this handle for mfAddInit() or mfFetchInit(). Release the handle using mfDestroyHandle().

mfDestroyHandle()

Releases a handle previously created by `mfCreateHandle()`.

Prototype `void mfDestroyHandle(void *hF);`

Return Values

<code>hF</code>	Session Handle created by <code>mfCreateHandle()</code> .
<code>None</code>	Operation successful. Pointer to session handle.
<code>0</code>	Operation failed.

Release the handle using `mfDestroyHandle()`. This is the last API to be called as the handle is relinquished.

mfAddInit()

Initializes the handle and input buffer for Add operations.

Prototype `int mfAddInit(const void *hF, char *tlvBuffer, const unsigned short tlvBufferSize);`

Parameters

In:	hF	Session Handle returned by mfCreateHandle().
In:	*tlvBuffer	Pointer to TLV buffer. TLVs are added in this buffer. This may be NULL.
In:	tlvBufferSize	Size of tlvBuffer in bytes.

Return Values

= 0	Success. This handle and buffer are initialized for add operation.
< 0	Invalid session handle or invalid operation or invalid parameters. See mfAPI Error Codes .

This must be the first API before calling mfAddTLV(). The session must be closed with mfAddClose().

mfAddClose()

Closes the Session Handle and returns the length of `tlvBuffer`. The `tlvBuffer` is ready for further processing after `mfAddClose`. After a successful `mfAddClose()`, the handle is no longer available for Add operations. To reuse the handle, it should be initialized again with either `mfFetchInit()` or `mfAddInit()`.

Prototype `int mfAddClose(const void *hP);`

Parameters

In:	<code>hP</code>	Session Handle returned by <code>mfCreateHandle()</code> and initialized for <i>Add</i> operations by <code>mfAddInit()</code> .
-----	-----------------	--

Return Values

<code>> = 0</code>	Successful closure. Returns length of <code>tlvBuffer</code> .
<code>< 0</code>	Invalid handle or invalid operation. See mfAPI Error Codes .

mfAddTLV()

Adds TLV to buffer. This API must be called after `mfAddInit()`. Successful execution results in adding the TLV in to the `tlvBuffer`. `mfAddTLV` does not check for duplicates and does not overwrite an existing Tag with the same name.

Prototype

```
int mfVarTLV(const void *hP, const char *tag, unsigned short tagLen, const char *value, const unsigned short valueLen);
```

Parameters

In:	hP	Session handle returned by <code>mfCreateHandle()</code> and initialized for add operations by <code>mfAddInit()</code> .
In:	tagLen	Size of field tag
In:	*tag	Pointer to Tag field. Its size must be as specified in <code>tagLen</code> . This parameter may not be NULL.
In:	valueLen	Size of value field
In:	*value	Pointer to value field. Its length must be as specified in parameter <code>valueLen</code> . This parameter may not be NULL.

Return Values

0	Success. Tag added.
< 0	Invalid session handle or parameter <code>valueLen</code> exceeds limits or no space available in <code>tlvBuffer</code> . See mfAPI Error Codes .

mfFetchInit()

Initializes the handle and `tlvBuffer` for Fetch operations. It positions the next pointer at the top of the TLV list.

Prototype `int mfFetchInit(const void *hF, const char *tlvBuffer)`

Parameters

In:	<code>hF</code>	Session Handle returned by <code>mfCreateHandle()</code> .
In:	<code>tlvBuffer</code>	Pointer to message buffer. TLVs are fetched from this buffer. This parameter cannot be NULL.

Return Values

<code>= 0</code>	Success. The handle is initialized for Fetch operations.
<code>< 0</code>	Invalid parameters or operation not supported. See mfAPI Error Codes .

This must be the first API for fetch operations – `mfFetchClose()`, `fmFetchReset()`, `mfPeekNextTLV()`, `mfFetchNextTLV()`, `mfFindTLV()`. The Fetch session must be closed with `mfFetchClose()`.

mfFetchClose()

Closes the Session Handle for fetch operations.

Prototype `int mfFetchClose(const void *hF)`

Parameters

In:	hF	Session Handle returned by <code>mfCreateHandle()</code> and initialized for <i>Fetch</i> operations by <code>mfFetchInit()</code> .
-----	----	--

Return Values

0	Successful closure.
< 0	Invalid handle or operation. See section 12 for list of error codes. See mfAPI Error Codes .

After successful `mfFetchClose()` call, the handle should be initialized again for with either `mfFetchInit()` or `mfAddInit()`.

mfFetchReset()

Repositions the “Next” pointer to the top of the TLV list.

Prototype `int mfFetchReset(const void *hF)`

Parameters

In:	hF	Session Handle returned by <code>mfCreateHandle()</code> and initialized for <i>Fetch</i> operations by <code>mfFetchInit()</code> .
-----	----	--

Return Values

0	Successful operation.
< 0	Invalid handle or no operation. See mfAPI Error Codes .

mfPeekNextTLV()

Obtains the next Tag length and the size of its Value. The “Next” pointer is not advanced.

Prototype

```
int mfPeekNextTLV(const void *hF, unsigned short *tagLen, unsigned short *valueLen)
```

Parameters

In:	hF	Session Handle returned by mfCreateHandle() and initialized for <i>Fetch</i> operations by mfFetchInit().
Out:	tagLen	Pointer to tagLen. The length of tag is set on return. This parameter must not be NULL.
Out:	valueLen	Pointer to valueLen. The length of value field is set on return. This parameter must not be NULL.

Return Values

0	Successful operation.
< 0	End of TLV list, no TLV returned or incorrect parameters or invalid operation. See mfAPI Error Codes .

Applications using this API can foresee the next tag without “fetching” it. This API does not advance the “Next” pointer. Applications can plan for space requirements by knowing the size. This call is usually followed by mfFetchNextTLV().

mfFetchNextTLV()

Fetches the next TLV from the list. Moves the “Next” pointer to the next TLV tag. The “Next” pointer moves to the next TLV in the list.

Prototype

int mfFetchNextTLV(const void *hF, const unsigned short tagSize, char *tag, unsigned short *tagLen, const unsigned short valueSize, char *value, unsigned short *valueLen);

Parameters		
In:	hF	Session Handle returned by mfCreateHandle() and initialized for <i>Fetch</i> operations by mfFetchInit().
In:	tagSize	Size of buffer tag in bytes. Its value must be greater than zero.
Out:	tag	Pointer to tag populated by mfFetchNextTLV. The size of this tag is returned by parameter tagLen. This parameter must not be NULL.
Out:	tagLen	Pointer to tagLen. The size of parameter tag is returned. This parameter must not be NULL.
In:	valueSize	Size of buffer value in bytes. Its value must be greater than zero.
Out:	value	Size of value in bytes. mfFindVarTLV sets this parameter when a match is found. This parameter must not be NULL.
Out:	valueLen	Pointer to valueLen set by mfFetchNextTLV. This parameter must not be NULL. Its value does not exceed valueSize.

Return Values		
0		Successful fetch. Tag and value returned.
< 0		End of TLV list, no TLV returned or invalid operation. See mfAPI Error Codes .

mfFindTLV()

Find and fetch the first TLV from list that matches tag. This function call searches for tags from the top of the TLV list and returns on the first successful match. The tag lengths are compared first and if equal, the tags are matched.

Prototype

```
int mfFindTLV(const void *hF, const char *tag, const unsigned short
tagLen, const unsigned short valueSize, char *value, unsigned short
*valueLen,)
```

Parameters

In:	hF	Session Handle returned by mfCreateHandle() and initialized for <i>Fetch</i> operations by mfFetchInit().
In:	tag	Pointer to tag to fetch. mfFindTLV will search for tag that matches the one pointed by parameter tag. This parameter must not be NULL.
In:	tagLen	Length of field tag. This field must be great than zero.
In:	valueSize	Size of buffer value in bytes. Its value must be greater than zero.
Out:	value	Value of tag. mfFindTLV sets this parameter when a match is found. Its length is as returned by valueLen. This parameter must not be NULL.
Out:	valueLen	Length of value in bytes. mfFindTLV sets this parameter when a match is found. This parameter must not be NULL. The returned value will not exceed valueSize.

Return Values

0	Successful match and tag returned.
< 0	No match or invalid operation. No TLV returned. See mfAPI Error Codes .

mfEstimate()

Compute memory requirement for a set of tags.

Prototype

```
int mfEstimate(const unsigned short tagCount, const unsigned short  
sigmaTag, const unsigned short sigmaValue);
```

Parameters

In:	tagCount	Number of tags.
In:	sigmaTag	Size of all the tag fields.
In:	sigmaValue	Size of all the value fields.

Return Values

> 0	Estimate of memory buffer to accommodate all the tags.
-----	--

mfAPI mfAddInit() requires a buffer (tlvBuffer) of size (tlvBufferSize). The size of the buffer is dependent on the number of tags, their cumulative size and the cumulative size of the value fields. The API provides the size of the TLV buffer required to accommodate all tags.

mfAPI Error Codes

mfAPI errors are negative and returned by the API. The list presented in Table 51 is not exhaustive and is subject to change.

Table 51 mfAPI Error codes

Error ID	Error Value	Description
EMF_TAG_SIZE	-3001	Tag size should be greater than zero.
EMF_VALUE_SIZE	-3002	Value size should be greater than zero.
EMF_PARAM_NULL	-3003	Parameter is null when one is not expected.
EMF_PARAM_INVALID	-3004	Parameter is invalid or out of range
EMF_TLV_BUF_SIZE	-3005	The buffer size is less than minimum required size
EMF_UNKWN_HANDLE	-3006	Unknown session handle
EMF_BUFFER_FULL	-3007	No space in buffer to add tag.
EMF_OP_NOT_SUPP	-3008	Operation not supported.
EMF_TAG_NO_MATCH	-3009	No matching tag was found.
EMF_TAG_EOL	-3010	End of list. No more tags in list.



3G Radio Firmware Update

On terminals that have the PHS8 3G radio module, Verix eVo provides support for updating (firmware down or firmware up) the radio's firmware.

Requirements for Firmware Update

The following requirements must be met to initialize the firmware update operation:

- Terminal contains PHS8 3G radio.
- Terminal is connected to the power supply.
- Firmware files are complete and authenticated in GID 46.
- `Config.sys` parameter `*UGPHS8FW=1` is set in GID 1.

Firmware Files

The firmware files are provided as a separate package from EOS. The readme file in the package contains the instructions for downloading the files to the terminal.

Firmware Update Operation

The requirements for firmware update are checked during terminal startup. Firmware update operation is initiated when all requirements are met.

User Prompt For Firmware Update

The firmware update operation is started with a confirmation from the user to either proceed or cancel the firmware update. The user prompt displays the following the screen:

- GID location of the firmware file.
- Number of files.
- Version number of firmware in file.
- Version number of firmware in radio.
- User selection menu:
 - 1 – Proceed (Proceed with firmware update)
 - 2 – Cancel (Postpone firmware update)
 - 3 – Abort (Cancel firmware update)

The `config.sys` parameter `*UGPHS8FW` is removed after user selection so that firmware update operation is no longer initiated on next terminal startup.

Console Notification for Incomplete Files

When incomplete firmware files in GID 46 are detected, a notification is displayed so that user can download the missing files. The following are displayed on screen:

- GID location of firmware file.
- Number of files.
- Version number of firmware in file.
- List of missing files separated by comma.

The notification is displayed until the user presses any key and then the terminal restarts. After restart, the notification is no longer displayed. The notification is displayed again when the list of missing files has changed on succeeding startups.

Firmware Update Status

The radio firmware update starts when the user selects '1 - Proceed' from the user prompt. While the update is in progress, the console displays the following information:

- Current file being processed.
- Number of bytes sent to the radio.
- Warning message to not remove the power supply.

NOTE



EOS will not be able to handle failed firmware update when the power supply is removed and when the battery is drained before firmware update is completed.

In the case of failed firmware update, it is possible that the radio will become inoperative except to perform firmware update again. In this case, EOS can be used to re-initiate the firmware update. Ensure that all requirements are met and that the power supply is/remains connected while the firmware update is in progress.

Firmware Update Result

After the radio firmware update is completed, the radio firmware version is retrieved and compared with the firmware version in the file. An error is displayed when the version numbers does not match.

On successful firmware update, the firmware files and signature files are deleted from the flash memory.

On failed firmware update or when the user selects "2 – Cancel" from the user prompt, the firmware files and signature files are retained in the flash memory. The user can re-initiate the firmware update operation by re-adding the `config.sys` parameter `*UGPHS8FW`.

When the user selects "3 – Abort" from the user prompt, the firmware files and signature files are deleted from the flash memory. User will need to re-download the files and re-add the `config.sys` parameter `*UGPHS8FW` to re-initiate the firmware update operation.

GPS

GPSD Application

On terminals that support GPS, Verix eVo provides libraries for GPS support.

GPSD is Verix eVo's engine for reading and processing NMEA sentences from a GPS port. GPSD is started by applications calling `IOCTL_GPS_START` and is stopped by calling `IOCTL_GPS_STOP`.

Data collected from the GPS port is available to applications via TCP/IP socket on port 2947 at localhost. The following diagram presents how GPSD is executed and how data is passed from the GPS port to applications.

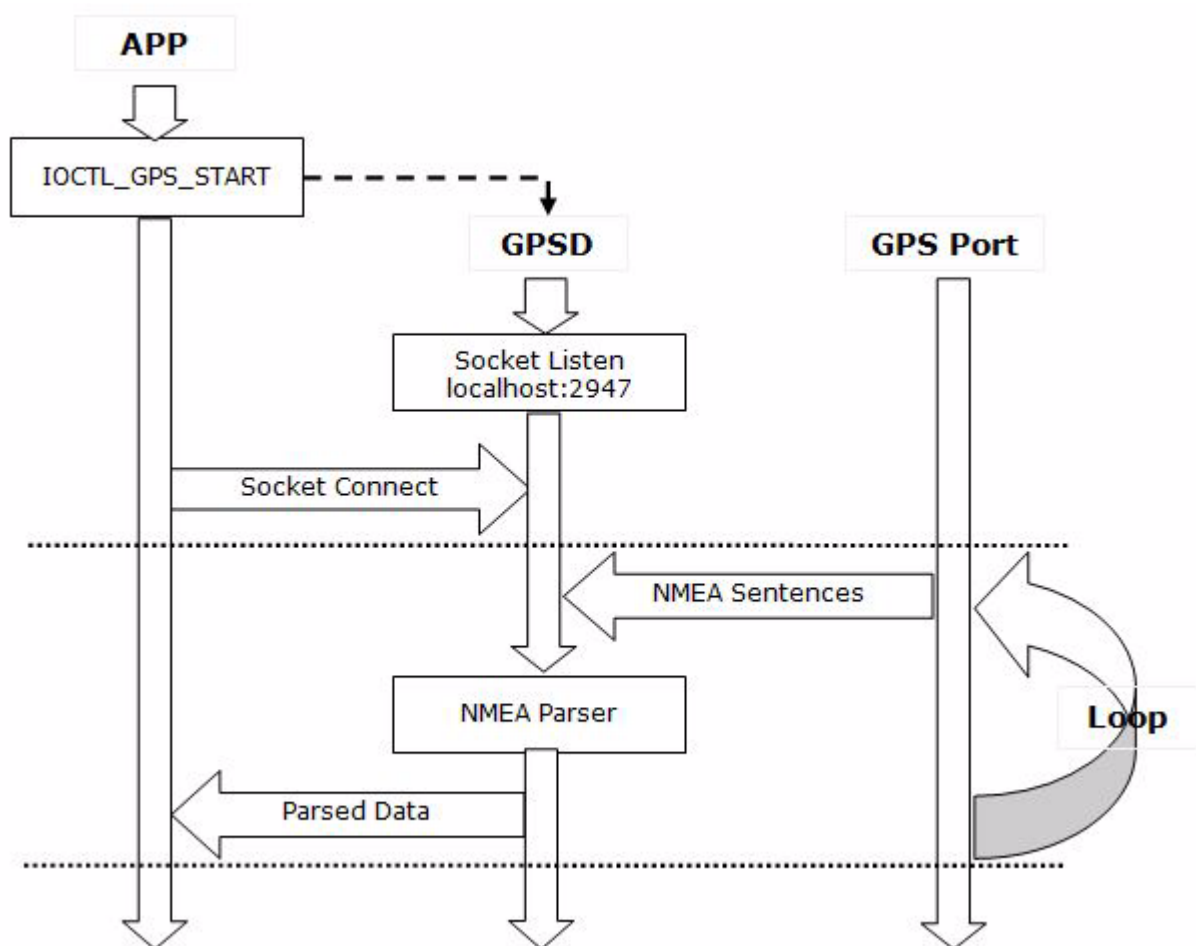


Figure 6 GPSD Application Process

GPS APIs

GPS support provides APIs to interface with the GPS engine. [Table 52](#) provides a summary of these APIs.

Table 52 **GPS API Summary**

API	Description
<code>int gps_open(char *server, char *port, struct gps_data_t *gpsdata);</code>	Initializes the GPS data structure <code>gpsdata</code> to hold data collected by the GPS and performs a socket connection to GPSD.
<code>int gps_close(struct gps_data_t *gpsdata);</code>	Ends the GPSD session and closes the socket connection.
<code>int gps_send(struct gps_data_t *gpsdata, char *fmt, ...);</code>	Writes a command to the GPS engine.
<code>int gps_read(struct gps_data_t *gpsdata);</code>	Accepts and interprets a response or series of responses from the GPS engine. When successful, the number of bytes read is returned.
<code>int gps_unpack(char *buf, struct gps_data_t *gpsdata);</code>	Provided for applications that wishes to manage socket I/O by itself. Parses JSON from the argument buffer into the target GPS data structure.
<code>bool gps_waiting(struct gps_data_t *gpsdata, int timeout);</code>	Checks whether there is new data from the GPS engine. Returns true when new data is available, false when no new data is available.
<code>int gps_stream(struct gps_data_t *, unsigned int flags, void *data);</code>	Commands the GPS engine to stream the reports that it has and make them available after polling.
<code>char *gps_data(struct gps_data_t *gpsdata);</code>	Returns the contents of the client data buffer.
<code>char *gps_errstr(int err);</code>	Return an ASCII string (in English) describing the error indicated by a non-zero return value from <code>gps_open()</code> .
<code>int gps_engine_start(int poll_time, char *server, char *port, int agps_set, char *agps_file);</code>	This is a wrapper API that starts GPRS network interface and then starts GPS normally done via ceAPI calls.
<code>int gps_engine_stop(void);</code>	This is a wrapper API that stops GPS and tears down GPRS network interface normally done via ceAPI calls.

gps_open()

Initializes the GPS data structure `gpsdata` to hold data collected by the GPS and performs a socket connection to GPSD.

Prototype `int gps_open(char *server, char *port, struct gps_data_t *gpsdata);`

Parameters

<code>server</code>	IP address of GPSD.
<code>port</code>	Listening port of GPSD.
<code>gpsdata</code>	GPS data structure.

Return Values

0	No error.
-1	Error occurred.

gps_close()

Ends the GPSD session and closes the socket connection.

Prototype `int gps_close(struct gps_data_t *gpsdata);`

Parameters

<code>gpsdata</code>	GPS data structure.
----------------------	---------------------

Return Values

0	No error.
-1	Error occurred.

gps_send()

Writes a command to the GPS engine.

NOTE

Use of this function is discouraged. Please use [gps_stream\(\)](#) as an alternative.

Prototype `int gps_send(struct gps_data_t *gpsdata, char *fmt, ...);`

Parameters

gpsdata	GPS data structure.
fmt	Format string containing elements from the command set documented at <code>gpsd(1)</code> .

Return Values

0	No error.
-1	Error occurred.

gps_read()

Accepts and interprets a response or series of responses from the GPS engine.

Prototype `int gps_read(struct gps_data_t *gpsdata);`

Parameters

<code>gpsdata</code>	GPS data structure.
----------------------	---------------------

Return Values

0	No error.
-1	Error occurred.

gps_unpack()

Provided for applications that wishes to manage socket I/O by itself. Parses JSON from the argument buffer into the target GPS data structure.

Prototype `int gps_unpack(char *buf, struct gps_data_t *gpsdata);`

Parameters

<code>buf</code>	Buffer containing the received socket data from GPSD.
<code>gpsdata</code>	GPS data structure.

Return Values

0	No error.
-1	Error occurred.

gps_waiting()

Checks if there is new data from the GPS engine.

Prototype `bool gps_waiting(struct gps_data_t *gpsdata, int timeout);`

Parameters

<code>gpsdata</code>	GPS data structure.
<code>timeout</code>	Currently unused.

Return Values

<code>true</code>	New data available.
<code>false</code>	No new data available.

gps_stream()

Commands the GPS engine to stream the reports that it has and make them available after polling.

Prototype `int gps_stream(struct gps_data_t *, unsigned int flags, void *data);`

Parameters

<code>gpsdata</code>	GPS data structure.
<code>flags</code>	Flag mask that sets various policy bits.
<code>data</code>	Currently unused.

Return Values

0	No error.
-1	Error occurred.

gps_data()

Returns the content of the client data buffer.

Prototype `char *gps_data(struct gps_data_t *gpsdata);`

Parameters

<code>gpsdata</code>	GPS data structure.
----------------------	---------------------

Return Values

<code>>NULL</code>	Pointer to data buffer of GPS data structure.
<code>NULL</code>	Error occurred.

gps_errstr()

Return an ASCII string (in English) describing the error indicated by a non-zero return value from [gps_open\(\)](#).

Prototype `char *gps_errstr(int err);`

Parameters

Err	Return value from gps_open() .
-----	--

Return Values

>NULL	Pointer to a static string containing the ASCII string.
NULL	Error occurred.

gps_engine_start()

This is a wrapper API that starts GPRS network interface and then starts GPS normally done via ceAPI calls.

Prototype

```
int gps_engine_start(int poll_time, char *server, char *port, int
agps_set, char *agps_file);
```

Parameters

<code>poll_time</code>	Poll time in seconds for receiving NMEA sentences from the GPS port.
<code>server</code>	IP address of GPSD. Set this to <code>NULL</code> to use the default value <code>localhost</code> .
<code>port</code>	Listening port of GPSD. Set this to <code>NULL</code> to use the default value 2947.
<code>agps_set</code>	Enables or disables A-GPS.
<code>agps_file</code>	Complete path and filename of the <code>GpsOneXTRA</code> binary file.

Return Values

0	No error.
< 0	Error occurred.

gps_engine_stop()

This is a wrapper API that stops GPS and tears down GPRS network interface normally done via ceAPI calls.

Prototype `int gps_engine_stop(void);`

Return Values

0	No error.
< 0	Error occurred.

Geo-Fence APIs

GPS support provides APIs to perform geo-fencing. Table provides a summary of these APIs.

Table 53 **Geo-Fence API Summary**

API	Description
<code>int geof_open(geofINFO *info);</code>	Initializes geo-fencing object.
<code>void geof_destroy(geofINFO *info);</code>	Destroys geo-fencing object.
<code>int geof_set(geofINFO *info, geof_cfg *cfg);</code>	Adds or edits a geo-fence.
<code>int geof_get_count(geofINFO *info);</code>	Returns the number of geo-fences.
<code>int geof_get_by_index(geofINFO *info, int index, geof_cfg *cfg);</code>	Retrieves a geo-fence based on its list index as specified by parameter <code>index</code> . Application can use this with <code>geof_get_count()</code> to do a traverse of all geo-fences.
<code>int geof_get_by_name(geofINFO *info, char *name, geof_cfg *cfg);</code>	Retrieves a geo-fence based on its name as specified by parameter <code>name</code> .
<code>int geof_del(geofINFO *info, char *name);</code>	Deletes a geo-fence. <code>name</code> indicates the name of the geo-fence to delete.
<code>int geof_del_all(geofINFO *info);</code>	Deletes all geo-fences.
<code>int geof_set_from_file(geofINFO *info, char *file);</code>	Adds geo-fences based on content of an INI file.
<code>int geof_save_to_file(geofINFO *info, char *file)</code>	Saves geo-fences to an INI file.
<code>int geof_check(geofINFO *info, double lat, double lon, char *name)</code>	Checks latitude and longitude point if inside or outside of any geo-fence.

geof_open()

Initializes geo-fencing object.

Prototype `int geof_open(geofINFO *info);`

Parameters

<code>info</code>	Geo-fencing object.
-------------------	---------------------

Return Values

<code>0</code>	No error.
<code>-2</code>	Invalid parameter.

geof_destroy()

Destroys geo-fencing object.

Prototype `void geof_destroy(geofINFO *info);`

Parameters

<code>info</code>	Geo-fencing object.
-------------------	---------------------

Return Values

<code>0</code>	No error.
<code>-2</code>	Invalid parameter.

geof_set()

Adds or edits a geo-fence.

Prototype `int geof_set(geofINFO *info, geof_cfg *cfg);`

Parameters

<code>info</code>	Geo-fencing object.
<code>cfg</code>	Geo-fence data to add or edit. Add is performed when <code>cfg->name</code> does not yet exist, otherwise edit is performed.

Return Values

0	No error.
-2	Invalid parameter.
-3	No heap memory.
-7	Maximum number of geo-fences reached.

geof_get_count()

Returns the number of geo-fences.

Prototype `int geof_get_count(geofINFO *info);`

Parameters

<code>info</code>	Geo-fencing object.
-------------------	---------------------

Return Values

<code>>= 0</code>	Number of geo-fences.
<code>-2</code>	Invalid parameter.

geof_get_by_index()

Retrieves a geo-fence object based on its list index as specified by parameter `index`. Application can use this with [geof_get_count\(\)](#) to do a traverse of all geo-fences.

Prototype `int geof_get_by_index(geofINFO *info, int index, geof_cfg *cfg);`

Parameters

<code>info</code>	Geo-fencing object.
<code>index</code>	Index of geo-fence to retrieve.
<code>cfg</code>	Output parameter for retrieved geo-fence object.

Return Values

0	No error.
-2	Invalid parameter.
-4	Geo-fence not found.

geof_get_by_name()

Retrieves a geo-fence object based on its list index as specified by parameter name.

Prototype `int geof_get_by_name(geofINFO *info, char *name, geof_cfg *cfg);`

Parameters

<code>info</code>	Geo-fencing object.
<code>name</code>	Name of geo-fence to retrieve.
<code>cfg</code>	Output parameter for retrieved geo-fence object.

Return Values

<code>0</code>	No error.
<code>-2</code>	Invalid parameter.
<code>-4</code>	Geo-fence not found.

geof_del()

Deletes a geo-fence.

Prototype `int geof_del(geofINFO *info, char *name);`

Parameters

info	Geo-fencing object.
name	Name of geo-fence to delete.

Return Values

0	No error.
-2	Invalid parameter.
-4	Geo-fence not found.

geof_del_all()

Deletes all geo-fences.

Prototype `int geof_del_all(geofINFO *info);`

Parameters

<code>info</code>	Geo-fencing object.
-------------------	---------------------

Return Values

<code>0</code>	No error.
<code>-2</code>	Invalid parameter.

geof_set_from_file()

Adds geo-fences based on content of an INI file.

Prototype `int geof_set_from_file(geofINFO *info, char *file);`

Parameters

<code>info</code>	Geo-fencing object.
<code>file</code>	Full path and filename of the INI file.

Return Values

0	No error.
-2	Invalid parameter.
-5	File access error.
-6	File content error.

geof_save_to_file()

Saves geo-fences to an INI file.

Prototype `int geof_save_to_file(geofINFO *info, char *file);`

Parameters

<code>info</code>	Geo-fencing object.
<code>file</code>	Full path and filename of the target INI file.

Return Values

0	No error.
-2	Invalid parameter.
-5	File access error.

geof_check()

Checks latitude and longitude point if inside or outside of any geo-fence.

Prototype `int geof_check(geofINFO, double lat, double lon, char *name);`

Parameters

info	Geo-fencing object.
lat	Latitude point to check.
lon	Longitude point to check.
name	When point is inside a geo-fence, geo-fence name will be copied to this output parameter.

Return Values

1	Point is inside a geo-fence.
0	Point is outside of all geo-fences.

Geo-Fence Downloadable File

Details for geo-fences to be used by the Geo-fence API can be defined in a user downloadable INI file. On download of INI file, application can call `geof_set_from_file()` to load the contents of the INI file. Table 54 presents the section and keys that are available for the user downloadable INI file.

Table 54 Section and Key Names for the User Downloadable INI File

Key Name	Value Type	Range	Description
[GEOFENCE-<Index>]			Section for a geo-fence setting where <Index> specifies the index of the section. Range for <Index> is 1 to 5 so up to 5 geo-fences can be specified.
NAME	String	1-11	Name of the geo-fence.
METHOD	Number	1-2	Geo-fencing method. The following values are allowed: <ul style="list-style-type: none"> • 1: Circular geo-fence • 2: Polygon geo-fence
RADIUS	Number		Radius of the circular geo-fence.
CENTER	Number		Latitude and longitude of the circular geo-fence's center following the format <LAT>,<LON>.
NUM_POINTS	Number	3-12	Number of points or vertices of the polygon geo-fence.
POINT_01	Number		Latitude and longitude of the polygon geo-fence's first point following the format <LAT>,<LON>.
POINT_02	Number		Latitude and longitude of the polygon geo-fence's second point following the format <LAT>,<LON>.
...			
POINT_NN	Number		Latitude and longitude of the polygon geo-fence's last point following the format <LAT>,<LON>.
			The maximum number of points that can be specified is 12.

Data Structures

The data structures used for GPS operations are discussed in the tables below.

Table 55 presents some of the common members of `gps_data_t`. For description on other parameters, please refer to the `gps.h` header file.

Table 55 `gps_data_t` Member Variables

Parameter	Data Type	Description
set	<code>gps_mask_set (unsigned long long)</code>	Flag bit-mask that indicates which variables in the structure are set. Valid bit values: <ul style="list-style-type: none"> • <code>TIME_SET</code>: PVT data time is set. • <code>MODE_SET</code>: PVT data mode is set. • <code>LATLON_SET</code>: PVT data latitude and longitude is set. • <code>ALTITUDE_SET</code>: PVT data altitude is set. • <code>TRACK_SET</code>: PVT data track is set. • <code>SPEED_SET</code>: PVT data speed is set. • <code>CLIMB_SET</code>: PVT data climb is set.
fix	<code>struct gps_fix_t</code>	Accumulated PVT data.
status	<code>int</code>	Indicator for <code>fix</code> . Valid values are: <ul style="list-style-type: none"> • <code>STATUS_NO_FIX(0)</code> • <code>STATUS_FIX(1)</code> • <code>STATUS_DGPS_FIX</code>
satellites_used	<code>int</code>	Number of satellites used in solution.
used	<code>int [MAXCHANNELS]</code>	PRNs of satellites used in solution.
dop	<code>struct_dop_t</code>	Dilution of precision.
skyview_time	<code>timestamp_t</code>	Sky view timestamp.
satellites_visible	<code>int</code>	Number of satellites in view.
PRN	<code>int [MAXCHANNELS]</code>	PRNs of in view satellites.
elevation	<code>int [MAXCHANNELS]</code>	Elevation of in view satellites.
azimuth	<code>int [MAXCHANNELS]</code>	Azimuth of in view satellites.
ss	<code>int [MAXCHANNELS]</code>	Signal-to-noise ratio of in view satellites.

Table 56 presents the members of `struct gps_fix_t`.

Table 56 `Struct gps_fix_t` Member Variables

Parameter	Data Type	Description
mode	<code>int</code>	Mode of fix: <ul style="list-style-type: none"> • <code>MODE_NOT_SEEN(0)</code>: Mode update not seen yet • <code>MODE_NOT_FIX(1)</code>: None • <code>MODE_2D(2)</code>: Good for latitude/longitude • <code>MODE_3D(3)</code>: Good for altitude/climb
time	<code>timestamp_t (double)</code>	Time of update.
ept	<code>double</code>	Expected time uncertainty.
latitude	<code>double</code>	Latitude in degrees.
epy	<code>double</code>	Latitude position uncertainty, in meters.

Table 56 Struct `gps_fix_t` Member Variables (continued)

Parameter	Data Type	Description
longitude	double	Longitude in degrees.
epx	double	Longitude position uncertainty, in meters.
altitude	double	Altitude in meters.
epv	double	Vertical position uncertainty, in meters.
track	double	Course made good, relative to true north.
epd	double	Track uncertainty, in degrees.
speed	double	Speed over ground in meters/sec.
eps	double	Speed uncertainty, in meters/sec.
climb	double	Vertical speed in meters/sec.
epc	double	Vertical speed uncertainty, in meters/sec.

Table 57 presents the member variables of `geof_cfg`.

Table 57 `geof_cfg` Member Variables

Parameter	Data Type	Description
name	char[12]	Geo-fence name. Each geo-fence requires a unique name.
method	int	Geo-fencing method. The following values are allowed: <ul style="list-style-type: none"> • 1: Circular geo-fencing • 2: Polygon geo-fencing
geof_info	union	Geo-fence information.

Table 58 presents the member variables of `geof_info`.

Table 58 `geof_info` Member Variables

Parameter	Data Type	Description
circle	<code>geof_circle</code>	Radius and center position for a circular geo-fence.
poly	<code>geof_poly</code>	Polygon points for a polygon geo-fence.

Table 59 presents the member variables of `geof_circle`.

Table 59 `geof_circle` Member Variables

Parameter	Data Type	Description
radius	double	Radius of the circular geo-fence in kilometers.
lat	double	Latitude of the circular geo-fence's center.
lon	double	Longitude of the circular geo-fence's center.

Table 60 presents the member variables of `geof_poly`.

Table 60 **geof_poly Member Variables**

Parameter	Data Type	Description
<code>num_points</code>	<code>int</code>	Number of points or vertices of the polygon geo-fence [3-12].
<code>lat</code>	<code>double[12]</code>	Latitude points of the polygon geo-fence. Up to 12 points can be specified.
<code>lon</code>	<code>double[12]</code>	Longitude points of the polygon geo-fence. Up to 12 points can be specified.



Verix eVo Communication Engine Application

The Verix eVo Communication Engine or CommEngine (`VXCE.OUT`) is the core component of the communication infrastructure. The boot strap application `VxEOS.OUT` starts `VXCE.OUT` which in turn starts the rest of the components that constitute the communication infrastructure consisting of device drivers and the TCP/IP stack. Both these components have defined interfaces that CommEngine uses.

Prior to starting the device drivers, CommEngine needs to determine the right drivers to load. It identifies the communication devices present on the terminal and then loads the right device driver. One of the key objectives of Verix eVo is to minimize the change when new communication devices are introduced. The purpose of this identification apart from the obvious purpose is to provide a level of abstraction to CommEngine and insulate it from future changes to new device introduction.

CommEngine also provides application services, i.e. it applications can obtain network events and status, configure and query device drivers, manage the network connection, etc. Applications must link with the CommEngine Interface Library (`CEIF.LIB`) for these services.

CommEngine Bootstrap Process

Use the following sections to employ the CommEngine bootstrap process.

CommEngine Invocation

On start up (both cold and warm boot) the OS starts Verix eVo. Verix eVo starts the executable files specified in the Verix eVo manifest file. The manifest contains CommEngine and VxNCP, which is the application with the User Interface (UI) and provides configuration and management services to Verix eVo components. CommEngine after start up brings up the device driver and the Verix eVo TCP/IP stack.

Conditionally Starting CommEngine

On startup Verix eVo looks for configuration variable `*NO.VXCE` in `GID 1`. If this variable is not present or its value is zero, CommEngine is started. If `*NO.VXCE` is set to 1, CommEngine is not started. Note that `*NO.VXCE` is a Verix eVo configuration parameter and is referenced here as it affects CommEngine.

VxNCP also has a conditional start-up variable `*NO.VXNCP`. The same rules apply to `*NO.VXCE`.

CommEngine Startup Operations

CommEngine Device Management

CommEngine, on start up, grabs the communication device and starts the necessary device drivers and the TCP/IP stack. In very rare cases, the communication device needs to be exchanged with other applications that are VMAC-compliant. In such situations, CommEngine needs to know that it should not grab the communication device and wait for a notification before it opens the communication device.

On start up, CommEngine looks at variable `*CEDM` (CommEngine Device Management) in `GID 1`. If not present or set to zero (`*CEDM=0`) then CommEngine considers this as normal operation and open the communication devices. If `*CEDM` is non-zero, then CommEngine waits for devices to be provided to it

CommEngine Pipe – CEIF

Applications use API provided by `CEIF.LIB` to obtain CommEngine services. This API is designed to exchange pipe messages with CommEngine. CommEngine is listening on a named pipe – CEIF.

CommEngine creates named pipe CEIF at start up.

CommEngine Configuration and Metadata File

CommEngine maintains its configuration in one configuration file:

Configuration Filename	Description
<code>VXCEHW.INI</code>	Configuration <i>data</i> file. This file contains the name and value of the configuration variable. A change to configuration value updates this file.
<code>VXCEHW.MTD</code>	Metadata file. This file contains the list of <code>config.sys</code> variables that CommEngine uses.

Network Configuration via Configuration Delta File

CommEngine configurations can be updated via downloading a configuration delta file. The configuration delta file can be described as a file containing a section name and having keys that the application users can define. Applications may download their own CommEngine Configuration Delta File named `VXCEHW.INI` in `GID 1`.

A configuration file may contain comment and blank lines to make the file self documenting and readable. A comment line starts with the semi-colon (;), is may be followed by any character and terminated by end of line (`CRLF 0x0D0A`). Similarly a blank line is any number (zero or more) tabs (`0x09`) and / or space (`0x20`) characters terminated with an end of line.

[Table 61](#) lists CE configuration section names.

Table 61 List of CommEngine Configuration Section Names

Configuration Section Name	Description
BT	Bluetooth
CDMA	Code Division Multiple Access
DIAL	Dial only (no PPP)
DIALBT	Dial over Bluetooth
ETH	Ethernet
ETHBT	Ethernet over Bluetooth
GSM	Mobile Phone (GSM) only (no PPP)
GPRS	General Packet Radio Service
PPPBT	PPP Dial over Bluetooth
PPPDIAL	PPP over Dial
PPPGSM	PPP over Mobile Phone (GSM)
WIFI	WiFi

Table 62 lists CEs configuration key names.

Table 62 List of CommEngine Configuration Key Names

Configuration Key Name	Description
authtype	Authentication type, applicable for PPP connection. <ul style="list-style-type: none"> Values can be 1 (NONE), 2 (PAP), 3 (CHAP), and 4 (MSCHAP). CommEngine uses NONE as a default if this is not specified.
auto_reconnect	States if automatic reconnection will be done by CE if a connection is lost.
devPort	Virtual port to be used over BT connection.
dhcp	This states if DHCP (1) or Static (0). Keys ipaddr, subnet, gateway, dns1 and dns2 will be ignored if dhcp is set to 1.
dns1	Primary DNS IP Address.
dns2	Secondary DNS IP Address.
gateway	Gateway IP Address.
ipaddr	IP address to be used.
password	Password, applicable for PPP connection. <p>Note: Password is case sensitive</p>
reconn_max	Total number of reconnection attempts.
reconn_interval	Time interval in seconds in between RECONN_MAX reconnection attempts.
reconn_to	Total time that CE is allowed to perform the two-tiered auto-reconnection.
reconn_exempt	String list of numeric error codes delimited by comma that will not initiate auto-reconnection.
startmode	This states if the NWIF is Auto or Manual

Table 62 List of CommEngine Configuration Key Names (continued)

Configuration Key Name	Description
subnet	Subnet Mask to be used.
username	Username, applicable for PPP connection.

Example A sample CommEngine Configuration Delta File is provided below:

```
[ETH]
startmode = A
dhcp = 0
ipaddr = 10.64.80.175
subnet = 255.255.252.0
gateway = 10.64.80.1
dns1 = 10.64.80.35
dns2 = 10.64.128.227

[PPPDIAL]
startmode = M
dhcp = 1
ipaddr =
subnet =
gateway =
dns1 =
dns2 =
username = verifone
// Note password is case sensitive. In this example the letters
are lower case
password = c4h9oh
authtype =
auto_reconnect = 1
```

Each Network Interface also has its own delta filename for the device driver specific configurations. To determine the device driver configuration filename, begin with "DDI_", then append the Configuration Section Name name, then add ".INI" at the end. For example:

```
"DDI_" + < configuration section name > + ".INI"
"DDI_" + "ETH" + ".INI"
```

Refer to the ReadMe.txt document packaged along with the software for the list of configuration master and delta filenames of each interface.

Network Configuration via GID 1 CONFIG.SYS parameters

CommEngine configuration can be specified via CONFIG.SYS parameters in GID 1. This is an alternate approach for users to configure CommEngine.

The configuration parameter is a combination of the section name (see [Table 61](#)) and key name (see [Table 62](#)). Taking the example of PPPDIAL username the corresponding configuration parameter name is: PPPDIAL.USERNAME. There is a period (.) between the section name and key name. In general, the configuration parameter is:

<SectionName>.<KeyName> = <KeyValue>

Where <SectionName> is defined in [Table 61](#) and <KeyName> is defined in [Table 62](#). The key value, <KeyValue> is same as specified in the configuration delta file (see [Network Configuration via Configuration Delta File](#)).

Here is another example where the ETH configuration parameters and its values are listed.

```
ETH.STARTMODE = A
ETH.DHCP = 0
ETH.IPADDR = 10.64.80.175
ETH.SUBNET = 255.255.252.0
ETH.GATEWAY = 10.64.80.1
ETH.DNS1 = 10.64.80.35
ETH.DNS2 = 10.64.128.227
```

Configuration Parameter Value Validation Checking

Each configuration parameter goes to validation checking, they are shown in [Table 63](#).

Table 63 CommEngine Configuration Key Name Validation

Configuration Key Name	Type	Validation		
		Min Length / Value	Max Length / Value	Format
authtype	Number	1	4	
auto_reconnect	Boolean	0	1	
devPort	String (alphanumeric)	6	15	
dhcp	Boolean	0	1	
dns1	IpAddr	N/A	N/A	xxx.xxx.xxx.xxx
dns2	IpAddr	N/A	N/A	xxx.xxx.xxx.xxx
gateway	IpAddr	N/A	N/A	xxx.xxx.xxx.xxx
ipaddr	IpAddr	N/A	N/A	xxx.xxx.xxx.xxx
password	String (alphanumeric)	0	50	
reconn_max	Number	0	10	
reconn_interval	Number	0	3600	
reconn_to	Number	0	18000	
reconn_exempt	String (alphanumeric)	0	500	
subnet	IpAddr	N/A	N/A	xxx.xxx.xxx.xxx

Table 63 CommEngine Configuration Key Name Validation (continued)

Configuration Key Name	Type	Validation		
		Min Length / Value	Max Length / Value	Format
startmode	String (alpha)	1	1	
username	String (alphanumeric)	0	50	

Table 64 lists additional validations are done for ipaddr, subnet, gateway, dns1 and dns2.

Table 64 List of CommEngine Configuration Key Name Additional Validation

Configuration Key Name	Required	Validation	
		CIDR Check	Notes
dns1	-	-	If dns2 was defined and dns1 was not, the values will be interchanged.
dns2	-	-	
gateway	✓	✓	
ipaddr	✓	-	
subnet	✓	-	

CommEngine Logging

By default, CommEngine logs are disabled, but applications can choose to enable CommEngine logs by defining configuration variable CELOG to C (CELOG=C). CommEngine logs, if enabled, by default only shows CommEngine major logs. CommEngine filter can be changed by defining configuration variable CEFIL to a value depending on how much details the application would like to see. Refer to the ReadMe.txt document packaged along with the software for the list of filter values.

Interface with Device Drivers

CommEngine starts and manages the device drivers. All device drivers work under the aegis of the Device Driver Manager. The Device Driver Manager implements the Device Driver Interface (DDI). CommEngine uses the DDI to manage it.

NOTE



*EOSNEWCIB config variable in GID 1 is used by CE to determine if re-scan of devices are needed. If value is set to 1, CE will scan the available devices in the terminal and re-generates the list of supported NWIF.

*EOSUSBDEV config variable in GID 1 can be used to enable/disable the NWIF for Ethernet dongle on Vx680 terminals. If value is set to 1, Ethernet dongle will be supported when the device scan process is conducted.

Interface with Verix eVo TCP/IP Library

Similar to the DDI, CommEngine uses the Verix eVo TCP/IP library to manage it. Management involves add /remove, configure and monitor network interfaces.

Application Interface

Application services are provided by CommEngine via `CEIF.LIB`. Applications link with `CEIF.LIB` which under the covers interfaces with CommEngine.

Application Events

CommEngine can forward application event and data from the driver. This is commonly used for notification purposes in case the application needs to do further processing when certain events occur. [Table 65](#) shows the events that the application can receive.

Table 65 Common Application Events

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
NO_BATTERY	200	No battery detected	Null-terminated String	When battery is removed from the terminal
NO_SIM_CARD	201	No SIM card	Null-terminated String	When there is no SIM card present
NOTIFY_SEND_TEXT_SMS	210	Data Format: "<mr>[, <scts>]" Where: <mr> - Message Reference <scts> - Service Centre Time Stamp (Optional)	Null-terminated String	When an IOCTL call with request ID <code>IOCTL_SEND_TEXT_SMS</code> is sent successfully.
NOTIFY_SEND_PDU_SMS	211	Data Format: "<mr>[, <ackpdu>]" Where: <mr> - Message Reference <ackpdu> - GSM 03.40 TPDU in hexadecimal format	Null-terminated String	When an IOCTL call with request ID <code>IOCTL_SEND_PDU_SMS</code> is sent successfully.
SIM_PIN_RETRY	202	Can be any of the following: SIM PIN: [attempts]	Null-terminated String	When SIM PIN or PUK verification fails

Table 65 Common Application Events (continued)

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
		<p>SIM PIN: UNKNOWN</p> <p>SIM PUK: [attempts]</p> <p>SIM PUK: UNKNOWN</p> <p>UNKNOWN</p> <p>Where: [attempts] – the number of remaining attempts</p> <p>Example: [attempts] – the number of remaining attempts</p> <p>Example: SIM PIN: 10 - There are 10 remaining SIM PIN attempts before the SIM card asks for SIM PUK.</p>		

Table 66 shows the GPRS application events.

Table 66 GPRS Application Events

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
GPRS_SIGNAL_QUALITY_STATUS	300	String that contains a any integer from 0 to 7, or 99.	Null- terminated String	When a signal quality URC is received from the modem
GPRS_ROAM_STATUS	301	<p>Can be any of the following:</p> <ul style="list-style-type: none"> • 0 - Registered to home network or not registered • 1 - Registered to other network 	Null- terminated String	When a roaming status URC is received from the modem
GPRS_MESSAGE_STATUS	302	<p>Can be any of the following:</p> <ul style="list-style-type: none"> • 0 - No unread SMS 	Null- terminated String	When a message URC is received from the modem

Table 66 **GPRS Application Events** (continued)

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
GPRS_CALL_STATUS	303	<ul style="list-style-type: none"> • 1 - There is an unread SMS Can be any of the following: <ul style="list-style-type: none"> • 0 - No active calls • 1 - There is an active call 	Null-terminated String	When a call status URC is received from the modem
GPRS_SMSFULL_STATUS	304	Can be any of the following: <ul style="list-style-type: none"> • 0 - SMS memory not full • 1 - SMS memory full 	Null-terminated String	When an SMS full memory URC is received from the modem
GPRS_SERVICE_STATUS	305	Can be any of the following: <ul style="list-style-type: none"> • 0 - Not registered to any network • 1 - Registered to any network 	Null-terminated String	When a network registration status URC is received from the modem
GPRS_PKT_SWITCHED_STATUS	306	Can be any of the following: <ul style="list-style-type: none"> • 0 - GPRS/EGPRS not available • 1 - GPRS available • 2 - GPRS attached • 3 - EGPRS available • 4 - EGPRS attached • 5 - WCDMA available • 6 - WCDMA attached 	Null-terminated String	When a packet switched URC is received from the modem.

Table 66 GPRS Application Events (continued)

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
		<ul style="list-style-type: none"> • 7 - HSDPA available • 8 - HSDPA attached • 9 - HSPA available • 10: HSPA attached • 11: HSPA+ attached 		

NOTE



These events are defined in `vos_ddi_app_evt.h`. Applications registering to CE should include this header file if it wants to process GPRS application events.

Table 67 shows the mobile phone (GSM) application events.

Table 67 Mobile Phone (GSM) Application Events

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
GSM_SIGNAL_QUALITY_STATUS	300	String that contains a any integer from 0 to 7, or 99.	Null-terminated String	When a signal quality URC is received from the modem
GSM_ROAM_STATUS	301	Can be any of the following: <ul style="list-style-type: none"> • 0 - Registered to home network or not registered • 1 - Registered to other network 	Null-terminated String	When a roaming status URC is received from the modem
GSM_MESSAGE_STATUS	302	Can be any of the following: <ul style="list-style-type: none"> • 0 - No unread SMS • 1 - There is an unread SMS 	Null-terminated String	When a message URC is received from the modem
GSM_CALL_STATUS	303	Can be any of the following: <ul style="list-style-type: none"> • 0 - No active calls • 1 - There is an active call 	Null-terminated String	When a call status URC is received from the modem

Table 67 Mobile Phone (GSM) Application Events (continued)

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
GSM_SMSFULL_STATUS	304	Can be any of the following: <ul style="list-style-type: none"> 0 - SMS memory not full 1 - SMS memory full 	Null-terminated String	When an SMS full memory URC is received from the modem
GSM_SERVICE_STATUS	305	Can be any of the following: <ul style="list-style-type: none"> 0 - Not registered to any network 1 - Registered to any network 	Null-terminated String	When a network registration status URC is received from the modem

NOTE



These events are defined in `vos_ddi_app_evt.h`. Applications registering to CE should include this header file if it wants to process mobile phone (GSM) application events.

Table 68 shows the CDMA application events.

Table 68 CDMA Application Events

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
CDMA_ROAM_STATUS	601	String that contains 0 or 1: <ul style="list-style-type: none"> 0 – Non roaming status 1 – Roaming status 	Null-terminated String	When a roaming status is received from the modem
CDMA_DORMANT_STATUS	602	String that contains 0 or 1: <ul style="list-style-type: none"> 0 – Non dormant status 1 – Dormant Status 	Null-terminated String	When a dormancy status is received from the modem
CDMA_CALL_STATUS	603	String that contains connection status	Null-terminated String	When connection status is received by the modem
CDMA_COVERAGE_STATUS	604	String that contains system mode: <ul style="list-style-type: none"> 0 – No Service 2 – CDMA mode 4 – HDR mode 8 – CDMA/HDR HYBRID mode 	Null-terminated String	When system mode status is received by the modem

Table 69 shows the Bluetooth events that the application can receive.

Table 69 Bluetooth Application Events

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
BT_SEARCH_DONE	800	Number of devices found	Null-terminated String	When search for Bluetooth devices is completed. Marks the start of retrieval of friendly names for each found device.
BT_QUERY_REMOTE_NAME_DONE	801	Number of devices found	Null-terminated String	When retrieval of friendly names for each found device is completed.
BT_FIRMWARE_UPGRADE_INIT	802	None	NA	Immediately after IOCTL_EX_BTBASE_FW_UP is called. Marks the start of sending the updated firmware to the Bluetooth base.
BT_FIRMWARE_DOWNLOAD_DONE	803	None	NA	When sending of updated firmware or factory restore firmware to the Bluetooth base is completed. The Bluetooth base starts the installation of the updated firmware or factory restore firmware and then restarts. All network interfaces (Ethernet/BT and PPP/BT) connected to the Bluetooth base are disconnected.
BT_FIRMWARE_UPGRADE_DONE	804	None	NA	When Bluetooth base has completed installation of the updated firmware and has restarted. All network interfaces that were disconnected are reconnected.
BT_FIRMWARE_UPGRADE_FAIL	805	Error message	Null-terminated String	When an error is encountered during the firmware upgrade. The Bluetooth base retains its current firmware version. Operation is aborted.

Table 69 Bluetooth Application Events (continued)

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
BT_SEARCH_INIT	806		NA	Immediately after IOCTL_SEARCH is called. Marks the start of search for Bluetooth devices.
BT_SEARCH_FAIL	807	Error message	Null-terminated String	When an error is encountered during the search for Bluetooth devices. Operation is aborted.
BT_QUERY_REMOTE_NAME_FOUND	808	Found device info	ddi_bt_device_query_info	When friendly name of a found Bluetooth device is retrieved. Event is sent for each found friendly name.
BT_RESTORE_FACTORY_INIT	812	None	NA	Immediately after IOCTL_EX_BTBASE_RESTORE_FACTORY is called. Marks the start of sending the factory restore firmware to the Bluetooth base.
BT_RESTORE_FACTORY_DONE	813	None	NA	When Bluetooth base has completed restore to factory and has restarted.
BT_RESTORE_FACTORY_FAIL	814	Error message	Null-terminated String	When an error is encountered during factory restore. The Bluetooth base retains its current settings.

Table 70 lists the ddi_bt_device_query_info parameters.

Table 70 ddi_bt_device_query_info

Parameter	Data Type	Description
bd_addr	Char[16]	String value of the Bluetooth address.
bt_class	Char[8]	String value of the Bluetooth class of device (CoD).
name	Char[64]	String value of the friendly name.

Table 71 shows the WiFi application events.

Table 71 **WiFi Application Events**

Param Name (Type:macro)	Event Value	Event Data	Event Data Type	When Sent
WIFI_SCAN_RESULTS	700	None	NA	When a scan operation triggered by IOCTL_SEARCH has been completed. Application can call IOCTL_GET_SEARCH_CNT and IOCTL_GET_SEARCH_RESULT.

CommEngine Auto-reconnection

For each network interface (NWIF), except for dial-only NWIFs like Dial and mobile phone (GSM), the AUTO_RECONNECT configuration, when set to true, allows the NWIF to maintain its connected state when the connection is suddenly lost. AUTO_RECONNECT can be used even if NWIF is not yet connected.

For scenarios wherein a NWIF is still attempting to connect and the attempt fails, the configuration of AUTO_RECONNECT is enhanced to support auto-reconnection of such failure. How and when CE will perform auto-reconnection is discussed in the following sections.

Two-tier Auto-Reconnection Scheme

CommEngine provides two levels of auto-reconnection procedures: back-to-back auto-reconnection and intervened auto-reconnection. NWIF configuration variables are available to enable/disable these procedures, and to customize the total number of consecutive auto-reconnections and how long each set of auto-reconnections can be intervened.

Tier One: Back-to-back Auto-Reconnection

Tier one is the basic recovery mechanism of auto-reconnection. When an NWIF fails to connect, CommEngine will initiate immediately a reconnection attempt. The total number of consecutive attempts to reconnect while there is failure is defined by the following NWIF configuration.

Variable Name (Type: Macro)	Type	Default Value	Range	Unit	Description
RECONN_MAX	Integer	0	0 - 10	N/A	Indicates the total number of reconnection attempts.

RECONN_MAX macro is defined in `ceifConst.h`, which can be found in the Include directory of Verix eVo (EOS) SDK. By default, the value of RECONN_MAX is set to zero (0) to indicate that the auto-reconnection scheme of CE is turned off by default. Corollary to this, RECONN_MAX is used to enable/disable the two-tier auto-reconnection scheme. The DDI CONN_RET for auto-reconnect attempts will be performed and exhausted for every iteration of RECONN_MAX.

Tier Two: Intervened Auto-Reconnection

In case tier one fails to connect the NWIF (e.g. RECONN_MAX attempts are exhausted and NWIF is still not connected), tier two can be used to continue the auto-reconnection attempt.

Aside from also performing consecutive reconnection attempts as indicated by RECONN_MAX, tier two introduces the feature to have intervals in between RECONN_MAX attempts. This feature is introduced to allow the terminal to go idle to enter power save mode.

The following configuration variable is used to set the interval between reconnection attempts:

Variable Name (Type: Macro)	Type	Default Value	Range	Unit	Description
RECONN_INTERVAL	Integer	600	0-3600	Seconds	Indicates the time interval in seconds in between RECONN_MAX reconnection attempts.

RECONN_INTERVAL macro is defined in `ceifConst.h`, which can be found in the Include directory of EOS SDK. If RECONN_INTERVAL is set to zero (0), CommEngine will attempt to reconnect immediately. In this case, the terminal is not expected to power save. Hence, it is advised to set RECONN_INTERVAL to a reasonably higher value to allow the terminal to go to power save mode.

Auto-Reconnection Time Period

The auto-reconnection scheme of CommEngine is time-bounded. To configure the total time that CE is allowed to perform the two-tiered auto-reconnection scheme, the following configuration variable is used:

Variable Name (Type: Macro)	Type	Default Value	Range	Unit	Description
RECONN_TO	Integer	3600	0-18000	Seconds	Indicates the total time that CE is allowed to perform the two-tiered auto-reconnection.

RECONN_TO macro is defined in `ceifConst.h`, which can be found in the `Include` directory of Verix eVo (EOS) SDK. If RECONN_TO is set to zero (0), CommEngine will perform auto-reconnection in an infinite time period. In this case, the terminal may enter power save mode during the reconnection interval (as indicated by RECONN_INTERVAL).

NOTE

The value of RECONN_TO should always be greater than RECONN_INTERVAL. Setting RECONN_TO with a lesser or equal value than RECONN_INTERVAL is an error. In the same way, setting RECONN_INTERVAL with a greater or equal value is an error.

There will always be at least one reconnection attempt executed for every set of RECONN_MAX attempts. This is to accommodate the scenario where in the value of RECONN_TO is considerably small that Verix eVo (EOS) may recognize the expiration of the auto-reconnection time period before even attempting a reconnection. In this manner, the auto-reconnection enhancement is not fully forfeited by the small value set to RECONN_TO.

The existing CONN_RET reconnection attempts performed and exhausted during a DDI connect request will then be performed and exhausted for every iteration of RECONN_MAX reconnection attempts. RECONN_TO is checked for every iteration of RECONN_MAX succeeding the first.

Auto-Reconnection Event Notification

If tier two has started, CE_EVT_RECONNECT_PENDING event is broadcasted to applications that the relevant network interface has a pending reconnection attempt to execute after RECONN_INTERVAL seconds. This event is broadcasted every time RECONN_MAX attempts are exhausted and RECONN_INTERVAL is set accordingly.

Exempting Connection Failure

Generally, CommEngine will perform auto-reconnection as long as there is an appropriate connection failure. However, there are some causes of failure that should not perform auto-reconnection. Below is a list of return errors that will not initiate auto-reconnection.

Error Code (Type: Macro)	Reason for Exclusion
E_DDI_LOW_BATT	Battery needs to be charged.
E_DDI_NO_DIAL_TONE	Landline cable may have been detached and should be attached back by user to the terminal.
E_DDI_NO_LINE	Landline cable may have been detached and should be attached back by user to the terminal.
E_DDI_NO_PRIMARY_NUMBER	Dial primary number should be set by user/ application.

E_DDI_NO_BATT	Battery needs to be attached back by user.
E_DDI_NO_SIM	SIM card is missing and should be inserted by user.
E_DDI_SIM_PIN_FAIL	SIM PIN is incorrect and should be set properly by user/application.
E_DDI_NO_NW_PROFILE	Channel should be set properly by user/application.
ECE_NOTDEVOWNER	Device returned by application to CE is not its own.

The above error code macros are defined in `ceifConst.h` and `vos_ddi_ext_err.h`, which are both found in the Include directory of the Verix eVo (EOS) SDK. When any one of the above errors is causing the connection failure, CE will not attempt an auto-reconnection. The error code is sent to the application through the broadcast event `CE_EVT_NET_FAILED`. The error code is part of the event data of type `stceNWEvt` and is represented by `neParam1`.

Aside from the mandatory error exemption list above, the application can also add specific error codes that they do not want CE to perform auto-reconnection. The following configuration variable can be used for this purpose:

Variable Name (Type: Macro)	Type	Default Value	Range	Unit	Description
RECONN_EXEMPT	String	Empty String	0 - 500	Characters	A string list of numeric error codes delimited by comma that will not initiate auto-reconnection.

RECONN_EXEMPT macro is defined in `ceifConst.h`, which can be found in the Include directory of EOS SDK. The format of RECONN_EXEMPT string should be as follows:

“error_code1, error_code2, error_code3,...”

For example, if `ECE_PPP_AUTH_FAILURE (-1038)` and `ECE_PPP_TIMEOUT_FAILURE (-1046)` connection failure should not auto-reconnect, then the NWIFs RECONN_EXEMPT should be:

“-1038, -1046”

Configuring Auto-Reconnection Parameters

In summary, [Table 72](#) shows the auto-reconnection parameters than can be set per network interface.

Table 72 Auto-reconnection Parameters per NWIF

Variable Name (Type: macro)	Type	Default Value	Range/ Length	Unit	Description
RECONN_MAX	Integer	0	0 - 10	N/A	Indicates the total number of reconnection attempts.
RECONN_INTERVAL	Integer	600	0 - 3600	Seconds	Indicates the time interval in seconds in between RECONN_MAX reconnection attempts.
RECONN_TO	Integer	3600	0 - 18000	Seconds	Indicates the total time that CE is allowed to perform the two-tiered auto-reconnection.
RECONN_EXEMPT	String	<i>Empty String</i>	0 - 500	Characters	A string list of numeric error codes delimited by comma that will not initiate auto-reconnection

These parameters can be set through the following methods:

- Through `ceSetNWParamValue()`. For more information, see [ceSetNWParamValue\(\)](#).
- Through `VXCEHW.INI` Delta File. For more information, see [Network Configuration via Configuration Delta File](#).
- Through `I:1/Config.sys` Variable. For more information, see [Network Configuration via GID 1 CONFIG.SYS parameters](#).

Other Connection Recovery Mechanisms

Aside from the AUTO_RECONNECT and CONN_RET configuration of CommEngine and DDI components, there are several recovery mechanism internally employed by EOS particularly in CE.

The following table lists the connection failure and the respective recovery mechanism used by CE.

Failure	Recovery
PPP Authentication Failure	CE will try to establish the PPP connection at most 3 times. If after all retry attempts are exhausted, CE will broadcast CE_EVT_NET_FAILED with ECE_PPP_AUTH_FAILURE.
Link Layer Open Failure	CE with retry once to establish the PPP connection. If retry connection still fails, CE will broadcast CE_EVT_NET_FAILED with ECE_TCPIP_STACK_FAILURE.

Network Control Panel (NCP)

Verix eVo is designed to be self contained, including user interface for administration, monitoring, diagnostics, configuration and setup tasks. This approach is implemented providing Network Control Panel as the default user interface for users to interface with different Verix eVo components.

The Network Control Panel consists of multiple functional modules collectively known as the Services Layer. This layer interacts directly with the components in Verix eVo. The user interface modules that will be present in the Verix eVo are:

- Configuration, Status & Management
- Software Downloads
- Network Diagnostics and Logging
- Device Drivers Configuration
- Bluetooth Configuration
- WiFi Configuration

The following diagram represents how Network Control Panel integrates Verix eVo components to implement the functionality listed above.

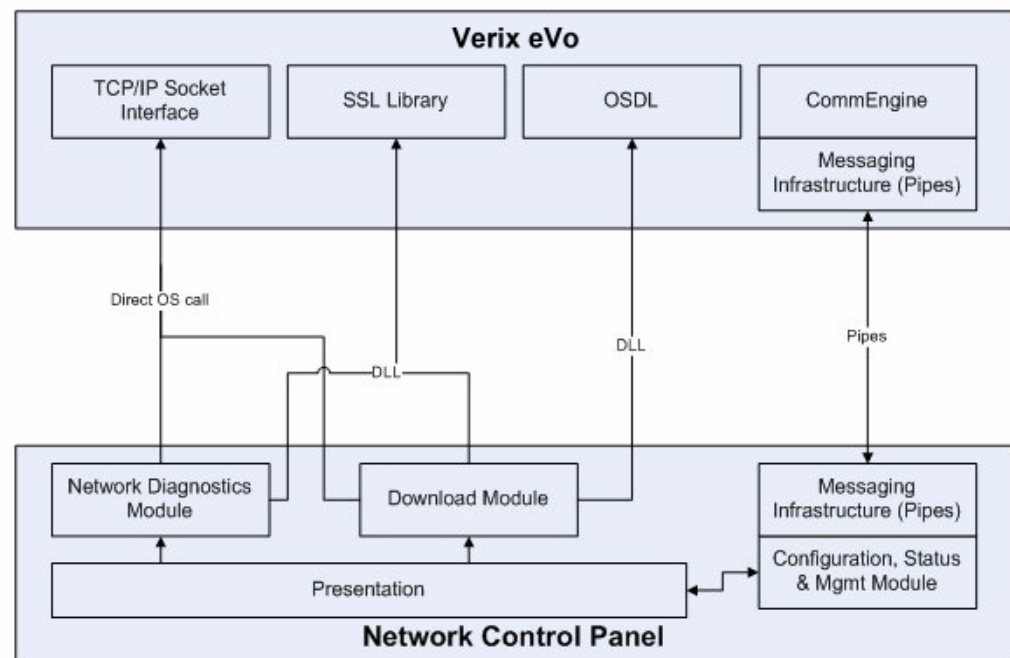


Figure 7 Network Control Panel

Startup Operation

By default Verix eVo will start NCP executable. Specific Customization may be implemented overriding the default behavior of NCP or completely replacing it with a new implementation.

Because default NCP components are bundled within Verix eVo, clearing the whole application space and CONFIG.sys variables only erases the custom NCP; upon restart Verix eVo will revert back to its default NCP solution. Full details will be covered in the next sections.

NCP is provided as default UI for all Verix eVo components. Alternative solutions may require implementing different features and functionality for specific markets; with this in mind Verix eVo NCP execution may be turned off.

Following sections summarize how Verix eVo starts NCP and how NCP will be available under different scenarios.

Starting NCP Executable

On startup, OS starts Verix eVo master application. This application is responsible for starting all other Verix eVo components, including NCP.

For those implementations where NCP will be replaced or turned off, CONFIG.sys variable *NO.VXNCP on GID 1 should be used.

On startup, VxEOS.out will look for configuration variable *NO.VXNCP=1 on GID 1. If *NO.VXNCP is set to 1 (one), NCP will not run. If *NO.VXNCP is not present or set to a different value, NCP will run normally.

On Startup, NCP will create all communication channels to operate with other eVo components.

- Registers with CommEngine via ceAPI.
- Creates named PIPE “VXNCP” which purpose is detailed below in [Invoking and Exiting NCP](#)
- Registration with VMAC is outside NCP responsibility and will be handled by VMACIF, as described in [Running under VMAC Environment](#).

Once NCP has initialized and completed registration/connection with other components, it waits until it gets activated.

Interoperability

The following sections describe how NCP communicates and interacts with other components on the system.

CommEngine

On Startup, NCP will:

- Register with CommEngine via ceAPI.
- Create named PIPE “VXNCP” which purpose is discussed under [Invoking and Exiting NCP](#).

In case registration with CommEngine fails, NCP will limit its functionality to those operations not requiring communication with CommEngine or the Device Drivers.

Device Ownership

Console Ownership

As described in [Invoking and Exiting NCP](#), it is important for NCP to return the CONSOLE to the same application that triggered its activation. For this purpose, NCP will keep ownership of the CONSOLE device until the user explicitly selects `Exit` from the UI.

To prevent user to abruptly switch application, NCP will disable hotkey on activation using OS API `disable_hot_key()`. When user selects `Exit` on IDLE, hotkey functionality will be restored by calling OS API `enable_hot_key()`.

Printer Ownership

Normal NCP operations only require the CONSOLE while some menu options, the user has the option to print the same information shown on the display. This directly implies getting control over the PRINTER.

NCP will not prevent its activation until printer becomes available, instead it will only attempt to open the PRINTER once the user selects the `Print` option. If open fails user will be properly notified and NCP will continue its regular operations.

Application invoking NCP is responsible for making PRINTER device available for NCP. Under no circumstances will NCP retry obtaining the printer nor retry the print operation. Users will have to manually retry the `Print` option from the UI.

For VMAC environments, PRINTER will be mandatory requirement for NCP activation and this will be handled by an external application. PRINTER will be listed on the ACTIVATE event of VMACIF application, who will be responsible for activating NCP.

Invoking and Exiting NCP

Invoking NCP

The invoking application only needs to call `ceAPI – ceActivateNCP()` to activate NCP. Upon return from this API, caller application does not own the CONSOLE.

- Identifying NCPs Task ID to assign CONSOLE ownership.
- Communicating Task ID of the Invoking Application; required to return CONSOLE to the same originator.
- Transporting caller Task ID to NCP via named PIPE “VXNCP”.

Returning to Invoking Application

When the user presses `Cancel` or selects `Exit` on the Idle Screen, NCP will return CONSOLE ownership to the original invoking application. Task ID of the originator is received during activation process via named PIPE.

Control Flow between invoking application and NCP

Figure 8 depicts the flow between the Invoking Application and NCP in both directions such as when NCP acquires control and when it relinquishes control. It follows this sequence of steps:

- 1 Invoking Application calls `ceActivateNCP()`.
- 2 API `ceActivate()`
 - a Disables hot key by calling OS API `disable_hot_key()`.
 - b Obtains task Id of named pipe "VXNCP" via OS API `get_owner()`
 - c Sends message to name pipe "VXNCP". This notifies NCP the task ID of the invoking application.
 - d Calls OS API `activate_task()`.
 - i. OS posts event `EVT_DEACTIVATE` to invoking application.
 - ii. OS posts event `EVT_ACTIVATE` to NCP
 - e API `ceActivate()` returns to calling application.
- 3 User is ready to exit NCP.
- 4 NCP enables hot key by calling OS API `enable_hot_key()`.
- 5 NCP calls OS API `activate_task()`.
 - a OS posts event `EVT_DEACTIVATE` to NCP.
 - b OS posts event `EVT_ACTIVATE` to invoking application

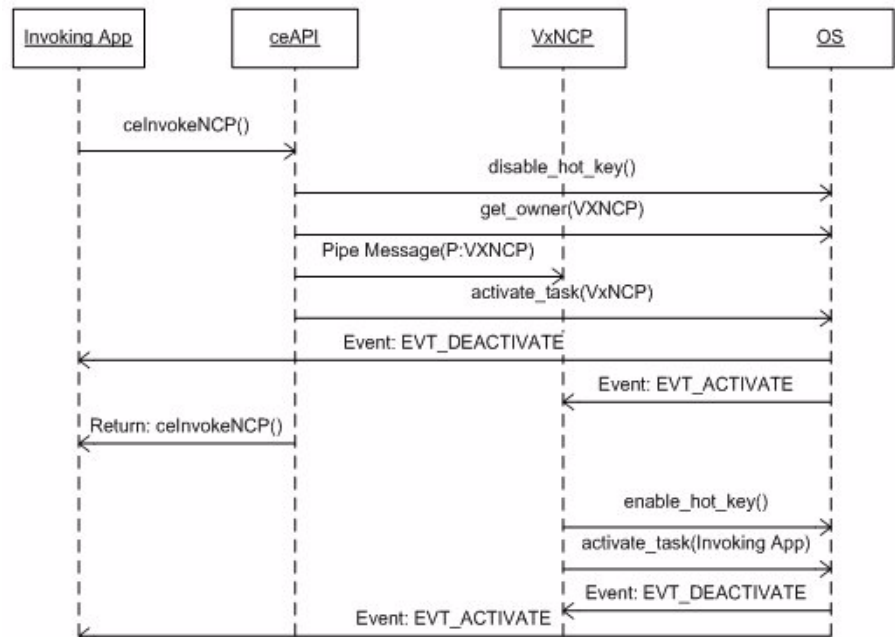


Figure 8 Control Flow – Invoking and Exiting VxNCP

IP Downloads from System Mode

An empty terminal needs the ability to download applications. OS will continue to handle downloads via dial, while Verix eVo becomes the engine responsible to handle IP downloads.

When user selects TCP/IP downloads in System Mode, OS will run `VxEOS.out` indicating the special startup sequence and will also run the application configured via `*ZTCP` on `GID 1`. When NCP starts, it will verify if `*ZTCP` is not configured to continue operation; if it is configured to any value NCP exits without doing anything.

OS communicates two parameters when invoking the application to handle IP downloads; Verix eVo will communicate same parameters to NCP

- `arg1` – download type: either “F” for full, or “P” for partial.
- `arg2` – download group: for example, “01” for group 1. The group will always be two digits to simplify parsing the user application.

NCP passes arguments to run in download mode. Similar to [Download](#), NCP will display the screens to confirm/enter the download parameters required by `SVC_ZONTALK_NET *ZA, *ZT, *ZN` and `*ZSSL`.

The following diagram depicts flow to handle IP downloads from System Mode.

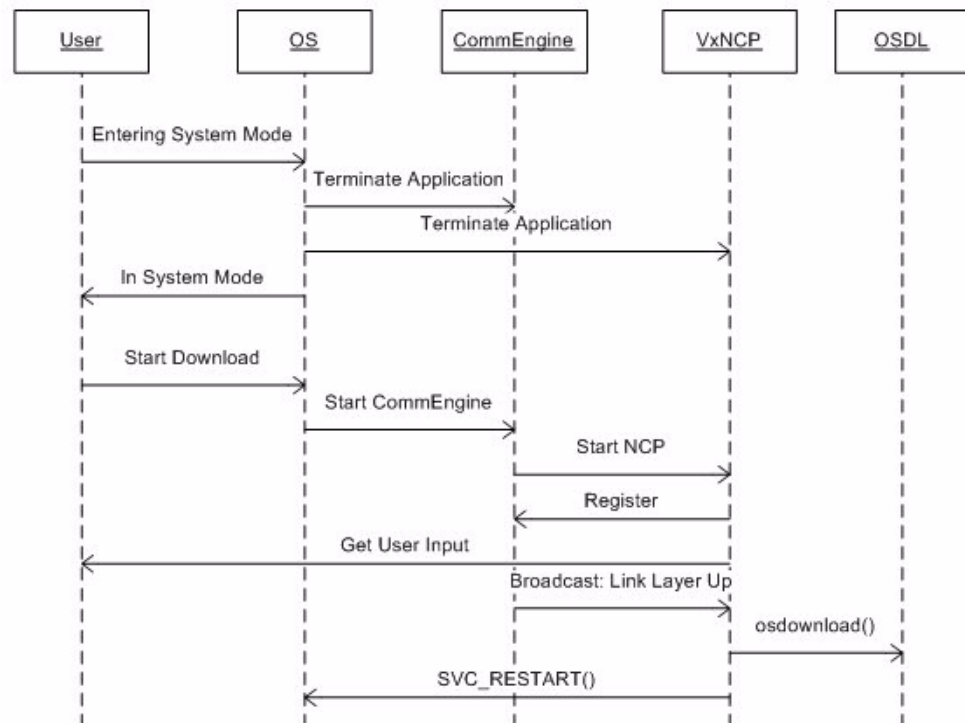


Figure 9 IP downloads from System Mode

Running under Single-Application Mode

No specific action is required. By default, NCP will run in the background waiting to be activated as described above.

Running under VMAC Environment

Under this environment, NCP is invoked through the VMAC menu. Since NCP runs before VMAC, NCP cannot register with VMAC/IMM. This implies that NCP is not aware of VMAC.

The VMAC hot key is enabled in this environment. When selected, it allows the user to `Exit` from Idle Screen. A surrogate application that is VMAC compliant will be created to interface with other applications on NCPs behalf. Selecting this application from VMAC menu activates NCP immediately.

Since VMAC is an optional component for Verix eVo solutions, the surrogate application will not be installed as part of the default Verix eVo bundle. It will be installed during deployment and customization for specific solutions.

Running in VxGUI Mode

When VxGUI is installed and is running on the terminal, NCP can be configured to use VxGUI for its UI display. Configuration is via the `GID 1 config.sys` parameter `*EOSUSEVXGUI`. When parameter is set to 1, NCP uses VxGUI for its UI display. When parameter is not 1, NCP uses its legacy UI display.

Configuration Files

NCP will be released with *factory* default settings via `.INI` file on `GID 46`. Customers will be able to overwrite some or all settings via `.INI` file on `GID 1`. File locations will be as follow:

Table 73 NCP Configuration Files

Location	Filename	Description
N:46	VXNCP.INI	Default configuration file. This file contains the name and value of configurable NCP parameters.
I:1	VXNCP.INI	User's NCP configuration file. This file contains NCPs settings defined during deployment. Values on this file will overwrite those configured on N:46/VXNCP.INI. Changes via NCPs UI will be reflected on this file.

NCPs configuration file will follow the standard `.INI` file format. Following is the current *factory* file.

```
[ Diagnostic-PING ]
    URL = Google.com

[ Diagnostic-DNS ]
    URL = VeriFone.com

[ Diagnostic-TCP ]
    URL = time-a.nist.gov
    PORT = 13

    send_data = 0;           //boolean value
    receive_data = 1;        //boolean value
    echo_host = 0;           //boolean value
```

```

send_size = 100;          //packet size to send
delay_after_connect = 0;  //delays in milliseconds, calls to SVC_WAIT

delay_before_close = 0;   //delays in milliseconds, calls to SVC_WAIT

send_timeout = 0;         //timeout in milliseconds
receive_timeout = 0;      // timeout in milliseconds
delay_between_sockets = 0; //delays in milliseconds, calls to SVC_WAIT
[ UI ]
  Prompts  = N:prompts.dat

  // Use absolute paths for FONT files, meaning GID must be explicitly noted
  // UI navigation may require switching GID and fonts must be accessible
  Font6x8  = default;
  Font8x16 = N:46/asc4x16.vft;

  // enable/disable arrows to navigate menus
  arrows4menus = 1 ;

  // Timeout values in milliseconds
  timeout_screens = 90000
  timeout_menus = 45000
  timeout_input = 30000
  timeout_confirm = 15000

  keyscan_up = 97;        //Default menu browsing UP key
  keyscan_down = 98;      //Default menu browsing DOWN key

  keyscan_up_custom = 50;  //Custom menu browsing UP key
  keyscan_down_custom = 56; //Custom menu browsing DOWN key

  Vx700_key_up  = 109;    //Different keyscan codes for Vx700
  Vx700_key_down = 108;    //No function/ATM keys available

  // DATE format on the display
  // Example for Jan 11th 2009
  // date separator on the 1st row
  // date format  on the 1st column
  //
  //          /          -          .
  //  MMDDYYYY  11/01/2009  11-01-2009  11.01.2009

```

```
// DDMMYYYY 01/11/2009 01-11-2009 01.11.2009
// MMDDYYYY Jan/11/2009 Jan-11-2009 Jan.1.2009
// DDMMYYYY 11/Jan/2009 11-Jan-2009 11.Jan.2009
date_format      = mmmddyyyy
date_separator = /
// TIME format on the display
; 12              01:00 PM
; 24              13:00
time_format = 12

[ DefaultNWIF ]
// Priority list to set Default NWIF on IDLE.
// First NWIF available becomes the default
// Key values will be ignored.
    GPRS =
    CDMA =
    Ethernet =
    PPP/Dial =
[ DownloadOptions ]
// Read-only section. Cannot be modified via GID 1 options = FPprR

[DownloadPrompts ]
upperF = Full
upperP = Partial - Restart
lowerP = Partial - Resume
upperR = Erase - Download
lowerR = Delete Files
upperR = Delete and Defrag
```

User Interface

The default language for all UI prompts will be English. To facilitate portability and use on different markets, the following considerations will be taken during development:

- Same font file will be used across all prompts and menus.
- All prompts will be external to the binary executable. Replacement prompts may be configured.
- To reduce the number of prompts, when possible icons will be used to represent the operation to perform. Icons will not be replaceable.

NOTE



Prompts will be limited to ASCII set supported by default OS fonts. No Unicode support.

Prompts file is generated using TXOFILE standard utility.

Customization

Verix eVo installation will include its own fonts to display the default prompts in English.

Prompts

```
[ UI ]
Prompts = F:1/MyPrompts.dat
```

Section	UI
Key Name	Prompts
Description	Custom prompts file

Display Font

```
[ UI ]
Font6x8 = default
Font8x16 = N:46/asc4x16.vft
```

Section	UI
Key Name	Font6x8
Description	Custom font file for small (6 x 8) strings

Section	UI
Key Name	Font8x16
Description	Custom font file for medium (8 x 16) strings

Time Format

Time will be configurable where the only valid strings are noted below:

Valid options	Examples
"12"	01:00 p.m.
"24"	13:00

```
[ UI ]
time format=12
```

Section	UI
Key Name	Time format
Description	Default time format for display and printed information

Date Format

Date will be configurable. `Date_format` specifies order to show Day-Month-Year while `date_separator` is used as separator character.

Valid values for `date_separator` are '/' (forward slash), '.' (dot) or '-' (dash). If key is not defined or its value does not match any of the valid strings the default separator '/' will be used.

Valid values for `Date_format` are MMDDYYYY, DDMMYYYY, MMMDDYYYY, DDMMMYYYY; where month in the format MM represents the numeric value while MMM represents the first three letters of the month's name. If variable is not defined or its value does not match any of the valid strings the default MMDDYYYY format will be used.

Table 74 represents the possible combinations based on the acceptable formats noted above.

Table 74 Possible Date_format combinations

Date_format Valid options	Example for Jan 11 2009 based on date_separator values		
	/	-	.
MMDDYYYY	11.01.2009	11.01.2009	11.01.2009
DDMMYYYY	01.11.2009	01.11.2009	01.11.2009
MMMDDYYYY	Jan/11/2009	Jan-11-2009	Jan.1.2009
DDMMMYYYY	11.Jan.09	11.Jan.09	11.Jan.09

The configuration variable `Month_names` allows defining the abbreviated string name to use as 'month' for the formats MMMDDYYYY & DDMMMYYYY. This variable provides twelve (12) strings, each one up to three (3) characters long, separated by comma.

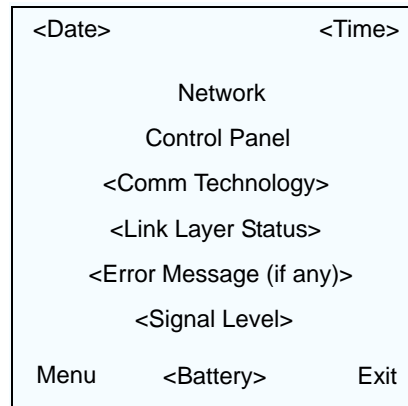
`Month_names` allows replacing the month names without having to download the whole prompts file. The following sample shows how to use this variable to download month names in Spanish

```
Month_names=Ene, Feb, Mar, Abr, May, Jun, Jul, Ago, Sep, Oct, Nov, Dic
```

`Month_names` variable is only verified if `Date_format` is configured to MMMDDYYYY or DDMMMYYYY. If `Month_names` variable is not defined or it does not contain exactly 12 strings, NCP will revert to its default English names.

Idle Screen

The following information is displayed as result of activating NCP. Menu options are organized following latest OS menu distribution.



- Date & Time
- NW Control Panel as product name
- Primary Communication Technology
- Link Layer Connection Status
- Error message if any
- Signal level on wireless devices
- Battery status on portable devices
- Menu to bring up UI options
- Exit to return to the application that activated NCP

Once any menu option is selected, all sub-menus will be text-only; browsing menu options is available via programmable function (PF) keys (purple) 1 and 2. Alternatively keys 2 and 8 may be used (similar to cell phone).

NOTE



For devices without function keys (i.e. Vx 700) custom keys are used and factory default will redefine each on a case to case basis.

Menu: Tools

This menu groups all menu options requiring operations, either locally to the network interface(s) available or externally over network interface(s) services.

Diagnostics

For all following menu options, IP addresses for those operations are previously configured.

1 All

When selected, it runs all tests listed below automatically.

- Ping Servers will use the Default NWIF
- Ping IP address will do a single attempt

After completion, if printer is available, user will also have the option to print the results as shown on the display.

2 Ping Servers

When selected, it prompts the user to select the NWIF. If selected `All` it will run the PING Servers for all servers for all NWIF.

Ping Gateway

Input	Gateway IP address from Primary NWIF
Action	PING Gateway's IP address
Result	If PING operation succeeds, UI will display total RTT (Round Trip Time) in milliseconds. If PING operation fails, UI will show a failure message.

Ping Primary DNS Server

Input	Primary DNS Server's IP address from NWIF selected
Action	PING Primary DNS Server's IP address
Result	If PING operation succeeds, UI will display total RTT (Round Trip Time) in milliseconds. If PING operation fails, UI will show a failure message.

Ping Secondary DNS Server

Input	Secondary DNS Server's IP address from NWIF selected
Action	PING Secondary DNS Server's IP address
Result	If PING operation succeeds, UI will display total RTT (Round Trip Time) in milliseconds. If PING operation fails, UI will show a failure message.

3 Ping IP Address / URL (Single or Continuous)

When menu option is manually selected, choices to select `Single` vs `Continuous` will be displayed.

Single

Input	Preconfigured IP address / URL
Action	PING IP address
Result	If PING operation succeeds UI will display total RTT (Round Trip Time) in milliseconds. If PING operation fails UI will show a failure message.

Note: Server must support PING feature

Continuous

Input	Preconfigured IP address / URL
Action	Continuously PING IP address until user press "Cancel" Key
Result	Summary screen will show accumulated results Successful PINGs / Attempts Average RTT (Round Trip Time) in milliseconds Accumulated RTT (Round Trip Time) in milliseconds Note: Server must support PING feature

4 DNS Lookup

Input	Preconfigured URL
Action	Convert URL to dotted IP address using DNS service
Result	IP address (or addresses) returned by DNS server Total resolution time in milliseconds If operation fails, UI will show a failure message.

5 TCP Socket

Input	Preconfigured IP address / URL and PORT
Action	Perform TCP socket connect and TCP socket disconnect to the IP:PORT address specified
Result	Total time in milliseconds. If operation fails, UI will show a failure message.

Network Maintenance

NCP will detect all Communication Technologies available on the device and will list all names for user to select the Network Interface (NWIF from CommEngine).

Once user selects the network interface to operate, the following options will be listed:

1 Network Restart

Input	Network Interface selected by user.
Action	Use ceAPI to <i>Stop</i> and <i>Start</i> the specific Network Interface.
Result	Total time in milliseconds. If operation fails, UI will show a failure message.

2 Network Start

Input	Network Interface selected by user.
Action	Use ceAPI to <code>Start</code> specific Network Interface.
Result	Total time in milliseconds. If operation fails, UI will show a failure message.

3 Network Stop

Input	Network Interface selected by user.
Action	Use ceAPI to <code>Stop</code> specific Network Interface.
Result	Total time in milliseconds. If operation fails, UI will show a failure message.

4 Switch SIM

Input	Network Interface selected by user.
Action	Use ceAPI to perform manual switching of SIM.
Result	Total time in milliseconds. If operation fails, UI will show a failure message.

Download

Currently VxDL is the default application for Software Downloads for most of the latest products. Moving forward to Verix eVo, NCP becomes the default application for Software Downloads from System Mode.

This menu option will be available to download applications into the device and setup the following timeouts:

- send timeout
- receive timeout

To perform the download, the following information is required:

- Target GID
- Selecting `Full`, `Partial` and `Restart`, `Partial` and `Resume`, `Delete Files`, or `Delete` and `Defrag` download
- How to reach VeriCentre Server
 - IP address
 - PORT number
- Block size
- Heap size
- Terminal ID

- Application ID
- Selecting TCP vs SSL

Summary screen will summarize all information for confirmation or editing before starting the download.

Once user confirms data entered, similar to the process described in [IP Downloads from System Mode](#) download starts.



NCP mimics VxDL features for secured (SSL) downloads. No SSL Server or Client certificates are supported.

FTP

NCP will detect if `VxNCP_ftp.ini` is available on the device to use FTP functionalities. The file is located as follows:

Table 75 NCP-FTP Configuration Files

Location	Filename	Description
Any GID between 1 and 15 on I:	VXNCP_ftp.INI	NCPs FTP configuration file. This file contains the name and value of configurable NCP-FTP profiles.

NCPs FTP configuration file will follow the standard `.INI` file format. The following is an example of the `VXNCP_ftp.ini` file:

```
[ FTPGET.FTPGET1 ]
ftpHost    = 10.64.31.58
port       = 21
userID     = user1
password   = 12345
localFile  = I:1/test1.txt
remoteFile = test2.txt

[ FTPPUT.FTPPUT1 ]
ftpHost    = 10.64.31.58
port       = 21
userID     = user1
password   = 12345
localFile  = I:1/test2.txt
remoteFile = test2.txt
```

The user will be allowed to select from the following options:

- GET FILE
- PUT FILE
- Edit INI
- Advanced Setup
 - Recv Buf size (edit FTP configuration *FTPRCVBUF)

USB Drive

This menu option groups all menu items for operation on a USB drive. Each menu item is displayed only when certain conditions for the operations are met.

3G Firmware Update

This menu item is displayed when NCP detects the presence of the PHS8 3G radio module. This menu item handles the 3G radio firmware update operation using firmware files from a USB flash drive. When this option is selected, the following conditions are checked:

- Terminal is connected to the power supply.
- Firmware files in the USB flash drive are complete including signature files.

NOTE



The firmware files are provided as a separate package from EOS. The readme file in the package contains the instructions for signing and copying the files to a USB flash drive.

When all conditions are met, 3G firmware update operation is initiated. The following screens are then displayed:

- Confirm Firmware Update
 - Number of files
 - Version number of firmware in file.
 - Version number of firmware in radio.
 - User prompt to either proceed or cancel firmware update
- Firmware Update Status
 - Current file being processed.
 - Number of bytes sent to radio.
 - Warning message to not remove the power supply.
 - Result of firmware update.

NOTE



EOS will not be able to handle failed firmware update when the power supply is removed and when the battery is drained before firmware update is completed.

In the case of failed firmware update, it is possible that the radio will become inoperative except to perform firmware update again. In this case, EOS can be used to re-initiate the firmware update. Ensure that all requirements are met and that the power supply is/remains connected while the firmware update is in progress.

**Menu: Terminal
Info (Terminal
Information)**

Selecting this menu option automatically presents several screens with current information from the device, drivers and connections.

The following sections will be presented in consecutive pages. User will have control when to scroll pages. As described in the [Printer Ownership](#) section, if PRINTER device is available, on any page, the user will also have the option to print all pages.

IP Status (IP Addresses Status)

This menu option will be listed separately from the menu option [Comm Technology](#), but in reality IP address information is directly related to the Network Interface. This menu is available to allow user direct access to the IP address information.

NCP will detect all Communication Technologies available on the device and will list all names for user to select the Network Interface (NWIF from CommEngine).

- If Static IP
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address
- If applicable
 - DHCP Lease Start Time
 - DHCP Lease End Time

When terminal has dual SIM support, IP information for the currently used SIM is displayed.

Comm Technology (Communication Technology)

CE allows every technology to be automatically or manually started on power up. NCP will retrieve the current setting via CEIF and will allow the user to change its value.

Based on the Communication Technologies available on the device, the following screens will be generated

NOTE



The lists below represent the minimum data to be provided by the DDI driver for display. The DDI driver may provide additional information not listed on this document which may be displayed by NCP.

Ethernet Status

- Device Name
- Device Driver Name
- Connection Status
- Link Speed
- MAC Address
- Driver Build Date
- Driver Version

Ethernet/BT Status

- Device Name
- Device Driver Name
- Connection Status
- Driver Version
- Driver Build Date
- MAC Address
- Link Speed
- Current State
- Target State

PPP/BT Status

- Device Name
- Device Driver Name
- Connection Status
- Driver Version
- Driver Build Date
- Primary Phone Number
- Secondary Phone Number
- SDLC
- Fast Connect
- PPP Authentication Type

WiFi Status

- Device Name
- Device Driver Name
- Connection Status
- Driver Version
- Driver Build Date
- MAC Address
- Link Speed
- Current State
- Target State

CDMA Status

- Device Name
- Device Handler
- Phone Number
- Username
- Password
- Connection Status
- RSSI
- Signal Quality
- Model
- Firmware Version
- PRL (Preferred Roaming List) Version
- ESN (Electronic Security Number)
- SID (System Identification Number)
- MDN
- MIN

GPRS Status

- Device Name
- Device Handler
- APN
- Phone Number
- Username
- Password
- Connection Status

- RSSI
- Signal Quality
- Model
- Firmware Version
- ICC ID
- IMSI
- IMEI
- Driver Version
- Driver Build Date
- Link Status
- Min Battery

PPP over Mobile Phone (GSM) Status

- Device Name
- Device Handler
- Phone Number
- Username
- Password
- Connection Status

PPP over Dial Status

- Device Name
- Device Driver Name
- Connection Status
- Driver Version
- Driver Build Date
- Primary Phone Number
- Secondary Phone Number
- SDLC
- Fast Connect
- PPP Authentication Type

Device Drivers

- WiFi
- Bluetooth
- Ethernet/BT

- PPP/BT
- Dial/BT
- PPP/Dial
- Dial
- Based on the .MTD setup and values via ceAPI, user will be able to change setup or restore to factory default
- Updated values will be saved on the specific .INI file

Versions

- VxNCP version and timestamp
- OS version and HW information
- SDK version (static library)
- eVo package version
- CommEngine and CEIF versions
- Device Driver(s) Software (DDI) version
- VxBT version
- VxWiFi version
- TCP/IP Library version, size, and timestamp
- SSL Library version, size, and timestamp
- eVo LOG version (static library)
- Heap and Stack

System

- Battery Threshold
- Navigation Keys
 - Up
 - Down
 - Enter
 - Cancel
 - Print (if not touchscreen)

NOTE



Battery threshold will only be available if the terminal is portable.

The **ENTER** key is also recognized as a **down scroll** key only when navigating multiple screens (ex. Terminal Info/Versions).

Menu: Setup

This menu option is password protected. User must type the System Password for GID 1.

NCP will detect all Communication Technologies available on the device and will list all names for user to select the Network Interface (NWIF from CommEngine).

Communication Technology

Changes to any Network Interface require manually restarting the corresponding interface via menu option Network Maintenance.

Based on the technologies available, following settings will be configurable via NCP.

Ethernet

- IP Setup
 - Startup Mode (Auto/Manual)
 - PPP AUTH Type (None/PAP/CHAP/MSCHAP)
 - Username
 - Password
 - IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address
- Auto Reconnect

- AUTO_RECONNECT
- RECONN_MAX
- RECONN_INTERVAL
- RECONN_TO

Ethernet/BT

- IP Setup
 - Startup Mode (Auto/Manual)
 - PPP AUTH Type (None/PAP/CHAP/MSCHAP)
 - Username
 - Password
 - IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address
- Auto Reconnect
 - AUTO_RECONNECT
 - RECONN_MAX
 - RECONN_INTERVAL
 - RECONN_TO

PPP/BT

- IP Setup
 - Startup Mode (Auto/Manual)
 - PPP AUTH Type (None/PAP/CHAP/MSCHAP)
 - Username
 - Password
 - IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address

- Secondary DNS Server IP address
- Auto Reconnect
 - AUTO_RECONNECT
 - RECONN_MAX
 - RECONN_INTERVAL
 - RECONN_TO

WiFi

- IP Setup
 - Startup Mode (Auto/Manual)
 - PPP AUTH Type (None/PAP/CHAP/MSCHAP)
 - Username
 - Password
 - IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address
- Auto Reconnect
 - AUTO_RECONNECT
 - RECONN_MAX
 - RECONN_INTERVAL
 - RECONN_TO

CDMA

- Username
- Password
- IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address

GPRS

- IP Setup
 - Startup Mode (Auto/Manual)
 - PPP AUTH Type (None/PAP/CHAP/MSCHAP)
 - Username
 - Password
 - IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address
- Auto Reconnect
 - AUTO_RECONNECT
 - RECONN_MAX
 - RECONN_INTERVAL
 - RECONN_TO
- Dual SIM
 - IP Setup
 - PPP AUTH Type (None/PAP/CHAP/MSCHAP)
 - Username
 - Password
 - IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address
 - A/T Switch Mode
 - A/T Switch Mode
 - Net Res Interval
 - Revert Interval

- Revert Timeout

PPP/Mobile Phone (GSM)

- IP Setup
 - Startup Mode (Auto/Manual)
 - PPP AUTH Type (None/PAP/CHAP/MSCHAP)
 - Username
 - Password
 - IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address
- Auto Reconnect
 - AUTO_RECONNECT
 - RECONN_MAX
 - RECONN_INTERVAL
 - RECONN_TO
- Dual SIM
 - IP Setup
 - PPP AUTH Type (None/PAP/CHAP/MSCHAP)
 - Username
 - Password
 - IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address
 - A/T Switch Mode
 - A/T Switch Mode
 - Net Res Interval

- Revert Interval
- Revert Timeout

PPP/Dial

- IP Setup
 - Startup Mode (Auto/Manual)
 - PPP AUTH Type (None/PAP/CHAP/MSCHAP)
 - Username
 - Password
 - IP Setup (DHCP/Static IP)
 - If Static IP:
 - IP address
 - Subnet Mask
 - Gateway IP address
 - Primary DNS Server IP address
 - Secondary DNS Server IP address
- Auto Reconnect
 - AUTO_RECONNECT
 - RECONN_MAX
 - RECONN_INTERVAL
 - RECONN_TO

NOTE



No Network Interface for Dial without PPP and mobile phone (GSM) without PPP are supported by VxNCP.

Default settings may be defined via configuration files. Run-time edition of those files will be available via NCP.

Diagnostics

- All
- Ping IP Address/URL
- DNS Lookup
- TCP Socket

Device Drivers

NCP as default UI for all Verix eVo components serves as UI Engine to dynamically modify settings for all Device Drivers available on the device. NCP will query CommEngine via ceAPI for all Network Interfaces and the corresponding Device Driver loaded. All Device Driver names will be listed for user to select which one to setup. Once user selects a specific Device Driver, NCP will use the Device Driver Name returned by ceAPI to load the corresponding .INI and .MTD configuration files.

The following steps demonstrate how NCP will dynamically build the UI based on the specific Device Driver configurable parameters.

- Device Driver name retrieved via ceAPI
- Configurable data is defined on the specific .MTD file
- Current values will be dynamically queried via ceAPI
- Based on the .MTD setup and values via ceAPI, user will be able to change setup or restore to factory default
- Updated values will be saved on the specific .INI file

Communication Technology on Idle

Generally, NCPs menu options detect all Network Interfaces from CommEngine via ceAPI. Under specific circumstances, defining a 'Default Network Interface' simplifies and improves the response time. This setting applies for the following options:

- WiFi
- Ethernet/BT
- PPP/BT
- PPP/Dial

User Interface

The NCP user interface defaults to settings defined via configuration file on startup. The run-time edition of the following settings will be configurable via NCP:

- Time Format (not available when displayed via VxGUI)
- Date Format
- Date Separator
- Timeout Screens
- Timeout Menus
- Timeout Inputs
- Timeout Confirm
- Inactivity Timeout

- Use VxGUI (see [Running in VxGUI Mode](#) for more information)

System

This screen displays changes to system settings. Based on the terminal available, the following settings will be configurable via NCP:

- Battery Threshold

NOTE



The System screen will only be available if the terminal is portable.

Menu: WiFi

This menu option is available when terminal is a VX 680 WiFi/Bluetooth. Selecting this menu option activate the WiFi Control Panel.

For more information, see [WiFi Control Panel \(WCP\)](#).

Menu: Bluetooth

This menu option is available when terminal is a VX 680 WiFi/Bluetooth. Selecting this menu option activates the Bluetooth Control Panel.

For more information, see [Bluetooth Control Panel \(BCP\)](#).

LogSpooler

At runtime, COM1 port is mainly used for logging purposes, but if an external device is connected to it, or if the terminal lacks a COM1 port, this could mean an inability for developers to capture logs. This can be remedied by using the OS logging feature (`*DEBUG / dbprintf()`) which allows for the accumulation of data on a circular buffer stored in the file system.

There are three advantages to using this feature, and these are:

- The log includes a timestamp using system clock;
- Writing to the file is faster than writing to the serial port; and,
- There is a capability to obtain an integrated log with OS information (`*DEBUG`) + Verix eVo (EOS) + app.

The disadvantage of using this feature is the user cannot get real time logs while debugging applications. Manual intervention is needed to enter into system/VTM mode. This is done by pressing F2+F4 and then pressing fourth programmable function (purple) key to dump log data.

To eliminate the manual intervention requirement for dumping logs via VTM mode, NCP has been enhanced to implement a thread called LogSpooler. LogSpooler runs in the background and periodically reads the system log and dump the data to a specific serial device.

NOTE



Earlier Verix eVo (EOS) releases may not yet have a LogSpooler feature. Please check Verix eVo (EOS) release notes for its availability.

Enabling Logs via LogSpooler

LogSpooler aims to enhance the logging capabilities of Verix eVo (EOS) by adding an option for the user to use other COM ports aside from COM1. COM port devices that can be used aside from COM1 are the COM2, USB-UART (COM6) and the USBD port.

NOTE



Currently, LogSpooler will keep waking up and will not allow a device to enter sleep mode. LogSpooler must be disabled for the terminal to do power save mode.

LogSpooler Settings

LogSpooler settings can be configured using system variables or via the NCP Menu.

LogSpooler Configuration Variable

To enable LogSpooler, NCP has provided the `*LOGSPOOLER` configuration variable. Valid values for `*LOGSPOOLER` are "COM1", "COM2", "COM6" and "USBD".

`*LOGSPOOLER` and `*LOG` are required to run LogSpooler on power up. `*DEBUG` must be 0 or not defined. `*LOGP` must not be defined.

LogSpooler will automatically be disabled if the value of `*LOGSPOOLER` is invalid or a port is not present on the device.

Other configuration variables relating to debug logs are described in *Verix eVo Volume I: Operating System Programmers Manual* (VPN DOC00301).

NCP s LogSpooler Menu

The NCP menu has optional menus for LogSpooler. Additional LogSpooler menus will only be visible if `*LOGSPOOLER` and `*LOG` environment variables are present in GID 1.

The following optional LogSpooler menus that are added are as follows:

Menu > Setup > Tools

- **Start** – Starts LogSpooler. Error message is displayed if LogSpooler is already running.
- **Stop** – Stops LogSpooler. Error message is displayed if LogSpooler is not running.

Menu > Setup > System

- **Log Frequency Time** – Specifies the time interval for LogSpooler to check and dumped log data.

- **Log Port** – Displays a list of valid COM ports the user can select. This is optional and will only appear if there are 2 or more supported port available for logging.



Updating COM port setting is not allowed when LogSpooler is running. Error message will be displayed if LogSpooler is running.



WiFi Control Panel (WCP)

WiFi Control Panel (WCP) is the default user interface for performing WiFi operations. On activation of WCP, the following operations can be performed:

- Discovery of WiFi access points
- Setup of network profiles for connecting to WiFi access points

Startup Operation

This section describes starting, invoking and exiting WCP.

Starting WCP Executable

Similar to NCP, WCP is started by the Verix eVo master application. For those implementations where WCP will be replaced or turned off, the `CONFIG.SYS` variable `*NO.VXWIFI` on GID 1 should be used.

On startup, VxEOS.out will look for configuration variable `*NO.VXWIFI=1` on GID 1. If `*NO.VXWIFI` is set to 1 (one), WCP will not run. If `*NO.VXWIFI` is not present or set to a different value, WCP will run normally.

Invoking and Exiting WCP

WCP is invoked from NCP via selecting the WiFi menu item. NCP then transfers console ownership to WCP. Exiting from WCP is performed by pressing the back button. WCP then transfers console ownership back to NCP.

Configuration Files

WCP will be released with factory default settings via `.INI` file on GID 46. The UI settings will be retrieved from NCPs `NCP.INI` which will provide a seamless look between NCP and WCP. The WiFi operation settings will be retrieved from `VXWIFI.INI` and is editable from WCPs UI.

User Interface

NCP, BCP and WCP share one prompt file. Please refer to the [User Interface](#) section of Network Control Panel for discussions on language for prompts.

Menu: Information

Selecting this menu option presents several screens with current information for WiFi.

The following sections will be presented in consecutive pages. The user will have control when to scroll pages. As described in the [Printer Ownership](#) section, if PRINTER device is available, on any page, the user will also have the option to print all pages.

Connection Status

This menu option displays BSS information for the currently associated network:

- SSID
- BSSID
- Signal Strength
- Network Authentication
- EAP Info
- Data Encryption
- Channel

Network Profile

This menu option lists down the SSID of all network profiles and displays BSS information of each network profile.

List of Network Profiles (SSID):

- Network ID
- SSID
- Network Type
- Channel
- Network Authentication
- EAP Info
- Data Encryption

Menu: Setup

This menu option is password protected. The user must type the System Password for GID 1.

Network Profile

This menu option lists down the SSID of all network profiles. Under this menu option, network profile parameters can be configured.

List of Network Profiles (SSID):

- Edit
 - SSID
 - Network Type (Infrastructure, Ad-hoc)
 - Network Authentication (Open, Shared, WPA/WPA2-PSK, WPA/WPA2-EAP, WPA/WPA2-CCKM)
 - Data Encryption (WEP, TKIP, AES)
 - WEP/WPA Info
 - Channel

- Set As Default
- Remove

Scan Networks

This menu option performs a scan of available networks at all channels. Networks found can be added as network profiles.

List of Network Profiles (SSID):

- Info
- Add Profile

New Profile

This menu option displays an Edit profile menu with blank values.



Bluetooth Control Panel (BCP)

Bluetooth Control Panel (BCP) is the default user interface for performing Bluetooth operations. On activation of BCP, the following operations can be performed:

- Discovery of Bluetooth devices
- Pairing with found Bluetooth devices
- Configuration of Bluetooth device ports

Startup Operation

This section describes starting, invoking, and exiting BCP.

Starting BCP executable

Similar to NCP, BCP is started by the Verix eVo master application. For implementations where BCP will be replaced or turned off, the `CONFIG.SYS` variable `*NO.VXBT` on GID 1 should be used.

On startup, `VxEOS.out` will look for the configuration variable `*NO.VXBT=1` on GID 1. If `*NO.VXBT` is set to 1, BCP will not run. If `*NO.VXBT` is not present or set to a different value, BCP will run normally.

Invoking and Exiting BCP

BCP is invoked from NCP via selecting the Bluetooth menu item. NCP then transfers console ownership to BCP. Exiting from BCP is performed by pressing the back button. BCP then transfers console ownership back to NCP.

Configuration Files

BCP is released with factory default settings via `.INI` file on GID 46. The UI settings are retrieved from NCPs `NCP.INI` that provides a seamless look between NCP and BCP. The Bluetooth operation settings will be retrieved from `VXBT.INI` and is editable from BCPs UI.

NOTE



Configurations in BCP does not reflect configurations in NCP Setup and information for Bluetooth Device Driver, and vice-versa.

User Interface

NCP, BCP and WCP share one prompt file. Please refer to [User Interface](#) section of Network Control Panel for discussions on language for prompts.

Menu: Information

Selecting this menu option presents several screens with current information for Bluetooth.

The following sections will be presented in consecutive pages. User will have control when to scroll pages. As described in the [Printer Ownership](#) section, if PRINTER device is available, on any page, the user will also have the option to print all pages.

Paired Devices

This menu option lists down the friendly names of all paired Bluetooth devices. When the friendly name is not available for a paired device, the BD address is displayed instead.

List of Paired Devices (Friendly name):

- General Info
 - Name
 - BD Address
 - Class
 - BT Device Port
- Vx680 Base Info
 - Firmware Version
 - PAN Connect Type
 - Ethernet Cable
 - Router IP Address
 - Serial Number

Configuration

This menu option displays the current Bluetooth operation settings:

- Discovery VFI Base Only
- Discovery Max Count
- Discovery Timeout
- VFI Base Prefix
- VFI Base Class
- Pair Max Count
- Secure Simple Pairing

Driver Info

This menu option displays current Bluetooth driver information:

- Version
- MAC Address

Menu: Setup

This menu option is password protected. The user must type the System Password for GID 1.

Discovery

This menu option performs a search for Bluetooth devices and presents a list of found Bluetooth devices. When the friendly name is not available for a found device, the BD address is displayed instead. Pairing can be performed on Bluetooth devices found.

Paired Devices

This menu option lists down the friendly names of all paired Bluetooth devices. When the friendly name is not available for a paired device, the BD address is displayed instead. Under this menu option, configuration of Bluetooth device port or delete of paired Bluetooth device can be performed.

List of Paired Devices (Friendly name):

- BT Device Port
 - Not Set
 - All Ports
 - /DEV/BT_PAN
 - /DEV/BT_MODEM
 - /DEV/BT_SERIAL
- Firmware Upgrade
 - GID
 - File
- Restore Factory
- Remove
- BD Address

Configuration

This menu option configures the Bluetooth operation settings:

- Discovery VFI Base Only
- Discovery Max Count
- Discovery Timeout

- VFI Base Prefix
- VFI Base Class
- Pair Max Count
- Pin Code
- Secure Simple Pairing



Verix eVo Log Library

The ELog (Verix eVo Log) library provides applications with a standard mechanism to log trace information that can be used for debugging and support.

Prior to Verix eVo, the Verix V platform used the LogSys library that was distributed as part of the VMAC libraries in the Verix V (ARM 9) DTK. With the introduction of Verix eVo, ELog is a separate shared library and distributed as part of the Verix eVo package. In addition to this, the underlying logging implementation has been changed to utilize the Verix V system logging mechanism. This change allows one log to capture information from applications, libraries, Verix eVo (EOS) components, and the operating system.

At the source level, the ELog library is backwards compatible to LogSys. ELog also adds some additional functionality and some differences in behavior may be noticed. The differences between ELog and VMACs LogSys are summarized in the [Backwards Compatibility Considerations Summary](#) section.

NOTE



The TCP/IP stack does not use the same logging mechanism. For capturing trace information from the TCP/IP stack, see the corresponding specification.

Library Files

The ELog library consists of the following files which are available in the Verix eVo package:

<code>eoslog.h</code>	Header file defines the logging macros
<code>elog.o</code>	Shared library interface file
<code>elog.lib</code>	Shared library file. Downloaded in drive N GID 15

NOTE



Verix eVo applications must link ELog interface file. `vrxhr.exe` must be used to specify the library location (drive N: GID 15).

Ex: `vrxhr app.out -L elog.lib=N:/elog.lib`

LOGSYS_FLAG

To access the ELog functionality, applications use a set of macros that calls library functions to format the information, including application name and source file information. These macros are conditionally compiled based on the macro LOGSYS_FLAG. If this macro is defined, the log macro statements are expanded in the application code, otherwise, the log macro statements are null.

LOGSYS_INIT

Initializes the ELog library for an application. The initialization consists of the following steps:

- Store the application name. This information will be added to any logged data.
- Determine if the log is enabled or disabled. The value provided by the `logType` input parameter may be overridden by the current GID `config.sys` variable `<APPNAME>LOG`. See [Config.Sys Variables](#).
- Determine the filter setting. The value provided by the `filter` input parameter may be overridden by the current GID `config.sys` variable `<APPNAME>FIL`. See [Config.Sys Variables](#).

Prototype

```
void LOG_INIT(char * appName, short logType, unsigned long filter);
```

Parameters

Parameter	Description
<code>char* appName</code>	Pointer to null terminated string containing application name. Maximum name length is 9 characters (not including terminating null).
<code>short logType</code>	Log Method. Valid values (defined in <code>eoslog.h</code>) are: <ul style="list-style-type: none"> • <code>LOGSYS_OS</code>: operating system logging • <code>LOGSYS_NONE</code>: no logging • <code>LOGSYS_COMM</code>, <code>LOGSYS_PIPE</code>, <code>LOGSYS_COMM_FILE</code>, <code>LOGSYS_FILE</code>: for backwards compatibility. These values are accepted but are converted to <code>LOGSYS_OS</code>.
<code>unsigned long filter</code>	Bit map to be used for filtering log statements. For <code>LOG_PRINTF</code> requests, the information is only logged if the corresponding bits are set in both filters.

`LOGSYS_INIT()` may be called multiple times by an application. Each call will re-execute the initialization steps.

The `logType` parameter (and `<APPNAME>LOG` variable) are interpreted differently by ELog and the older VMAc LogSys. In VMAc LogSys, this parameter (and config variable) allows an application to direct it's log output to a device, file, pipe or to disable logging. In ELog, this parameter (and config variable) simply allows the application to enable/disable the logging.

LOG_PRINTF, LOG_PRINTF_MOD

LOG_PRINTF and LOG_PRINTF_MOD provide a general mechanism for logging trace messages. Both APIs follow the same logic but LOG_PRINTF_MOD will include the module name in the formatted message. LOG_PRINTF_MOD is useful for logging from libraries (allowing the library information to be easily identified).

Prototype


```
void LOG_PRINTF(const char* fmt, ...);  
void LOG_PRINTF_MOD(const char* module, const char* fmt, ...);
```

Parameters

Parameter	Description
const char* module	Module name to be included in the final formatted log message. Module name is a null terminated string. If longer than 10 characters, the library truncates the name.
const char* fmt	Standard C printf format string. Based on the format string, additional arguments will be processed. The application should ensure that the formatted data does not exceed 100 bytes.

If the log method is not set to LOGSYS_NONE , LOG_PRINTF/LOG_PRINTF_MOD formats and logs the provided data. The ELog library adds the application name, source file name and line number to the application’s information. For LOG_PRINTF_MOD, the module name is also added.

NOTE



LOG_PRINTF/LOG_PRINTF_MOD are not affected by the filter setting. Information is logged regardless of the application’s filter value.

Applications should ensure that the formatted log message does not exceed 100 bytes. If this size is exceeded, the application memory may be corrupted. For messages that exceed 100 bytes, the ELog library attempts to truncate the information then process the data.

LOG_PRINTF, LOG_PRINTF_MOD

LOG_PRINTF and LOG_PRINTF_MOD provide filtered logging.

LOG_PRINTF_MOD messages will include the provided module name which can be used to identify logging from libraries or specific modules.

Prototype

```
void LOG_PRINTF(unsigned long filter, const char* fmt, ...)
void LOG_PRINTF_MOD(unsigned long filter, const char* fmt, ...)
```

Parameters

Parameter	Description
const char* module	Module name to be included in the final formatted log message. Module name is a null terminated string. If longer than 10 characters, the library truncates the name.
unsigned long filter	Filter bit map. If the corresponding bits are not set in the application's filter (see LOG_INIT), the message is not logged.
const char* fmt	Standard C printf format string. Based on the format string, additional arguments will be processed. The application should ensure that the formatted data does not exceed 100 bytes.

If the log method is not set to LOGSYS_NONE, LOG_PRINTF() / LOG_PRINTF_MOD will test the provided filter value. If the filter test is passed, the information is formatted and logged. The ELog library will add the application name, source file name and line number to the application's information. For LOG_PRINTF_MOD, the module name is also added.

Applications should ensure that the formatted log message does not exceed 100 bytes. If this size is exceeded, the application memory may be corrupted. For messages that exceed 100 bytes, the ELog library will attempt to truncate the information then process the data.

LOG_INVALID_POINTER, LOG_INVALID_POINTER_MOD

LOG_INVALID_POINTER and LOG_INVALID_POINTER_MOD can be used to log a statement if memory is not accessible at an address. LOG_INVALID_POINTER only verifies that the memory at the pointer is accessible and will not result in a system crash. The access may still be erroneous. Remember that:

- Pointers on the stack and the heap is valid as long as the size remains in the overall stack/heap. So for example, if a char array test[20] is declared on the stack, and then check for access for 25 bytes, it will likely be ok. The validation IS NOT based on the array variable size.
- Similarly, addresses of r/w global data is valid as long as the size is within the overall r/w global data region. The size of this region is the size required rounded up to the next page boundary.
- If the address points to const data, it will always be invalid for writing. For reading, it will be valid as long as the size remains in the overall const data area.

The LOG_INVALID_POINTER feature requires operating support that is available in Vx OS with System Library Version 1.01 or greater, and in eVO OS with System Library 3.00 or greater. If used on an older operating system, the macro has no affect (it just returns without attempting to validate the address). To determine the system library version for an OS, see the `syslib_version()` API.

If an application uses these macros but is running on a terminal with an older VxEOS package that does not support these APIs, the application may crash. Libraries earlier than 1.0.0.8 will crash the system. In library versions 1.0.0.8 – 1.0.1.1, calling the function will just log an “unsupported function” message. To verify the EOSLog library version, use the `GetEOSLogVersion()` function and check for version is 1.0.1.2 or later.

Prototype

```
void LOG_INVALID_POINTER(void* ptr, char access, int size)
void LOG_INVALID_POINTER_MOD(const char* module, void* ptr, char access,
int size)
```

Parameters

<code>ptr</code>	The address to validate.
<code>access</code>	Use 'r' if the address points to memory that will only be read. Any other value is assumed to be read/write access.
<code>size</code>	The number of bytes that will be accessed from the pointer.
<code>Mod</code>	The module name to be included in any logged statement.

The logged message will have the following format:

```
<app name>:<file name>:<line number>: Invalid Address <ptr hex value>  
Access <rw> Size <size>
```

```
<app name>:<module name>:<file name>:<line number>: Invalid Address <ptr  
hex value> Access <rw> Size <size>
```

Example

```
TESTAPP:F:FOO.C: ||L:00202|Invalid Address 0x99ABCDEF Access r Size=1000
```

Macros for Reporting Errors

The following macros check a value (as described by the macro name) and if the test succeeds, log a message.

Prototype

```
void LOG_PRINTFF_MOD(unsigned long filter, const char* fmt, ...)
void LOG_NONZERO_ERROR(int valueToCheck, unsigned long notUsed)
void LOG_NEGATIVE_ERROR(int valueToCheck, unsigned long notUsed)
void LOG_ZERO_ERROR(int valueToCheck, unsigned long notUsed)
void LOG_NONZERO_ERROR_CRIT(int valueToCheck, unsigned long notUsed)
void LOG_NEGATIVE_ERROR_CRIT (int valueToCheck, unsigned long notUsed)
void LOG_ZERO_ERROR_CRIT (int valueToCheck, unsigned long notUsed)
void LOG_ZERO_NULL_POINTER(int valueToCheck, unsigned long notUsed)
void LOG_NONZERO_ERROR_MOD(const char* module, int value,
unsigned long notUsed)
void LOG_NEGATIVE_ERROR_MOD (const char* module,int valueToCheck, unsigned
long notUsed)
void LOG_ZERO_ERROR_MOD (const char* module,int valueToCheck, unsigned
long notUsed)
void LOG_NONZERO_ERROR_CRIT_MOD (const char* module,int valueToCheck,
unsigned long notUsed)
void LOG_NEGATIVE_ERROR_CRIT_MOD (const char* module,int valueToCheck,
unsigned long notUsed)
void LOG_ZERO_ERROR_CRIT_MOD (const char* module,int valueToCheck,
unsigned long notUsed)
void LOG_ZERO_NULL_POINTER_MOD (const char* module, int
valueToCheck,unsigned long notUsed)
```

Parameters

Parameter	Description
const char* module	Module name to be included in the final formatted log message. Module name is a null terminated string. If longer than 10 characters, the library truncates the name.
int valueToCheck	Value to check, as described by the macro name. For example, LOG_ZERO_ERROR will log a message if the value provided equals 0.

If the log method is not set to LOGSYS_NONE, these macros will test the described condition and, if true, log a message. With the exception of the additional `errno` information, these message formats are similar to earlier LogSys libraries. The message formats are as follows.

LOG_NONZERO_ERROR	<AppName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> NCT
LOG_NEGATIVE_ERROR	<AppName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> NCT

LOG_ZERO_ERROR	<AppName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> NCT
LOG_NONZERO_ERROR_CRIT	<AppName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> CRT
LOG_NEGATIVE_ERROR_CRIT	<AppName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> CRT
LOG_ZERO_ERROR_CRIT	<AppName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> CRT
LOG_ZERO_NULL_POINTER	<AppName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> NULLP
LOG_NONZERO_ERROR_MOD	<AppName>:<ModuleName>:ER <valueToCheck> <errno> F:<filename> L:<line number> NCT
LOG_NEGATIVE_ERROR_MOD	<AppName>:<ModuleName>:ER <valueToCheck> <errno> F:<filename> L:<line number> NCT
LOG_ZERO_ERROR_MOD	<AppName>:<ModuleName>:ER <valueToCheck> <errno> F:<filename> L:<line number> NCT
LOG_NONZERO_ERROR_CRIT_MOD	<AppName>:<ModuleName>:ER <valueToCheck> <errno> F:<filename> L:<line number> CRT
LOG_NEGATIVE_ERROR_CRIT_MOD	<AppName>:<ModuleName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> CRT
LOG_ZERO_ERROR_CRIT_MOD	<AppName>:<ModuleName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> CRT
LOG_ZERO_NULL_POINTER_MOD	<AppName>:<ModuleName>:ER <valueToCheck> <errno> F:<filename> L:<linenumber> NULLP

The error reporting macros are not affected by the filter setting. Regardless of the application's filter value, these macros will log messages if the corresponding condition is true.

NOTE



Some macros might not be listed, please refer to `eoslog.h` for complete list of macros.

Re-assigning Log Settings

Re-assigns log settings.

Prototype

```
int LOG_SET_TYPE(short logType)
int LOG_SET_FILTER(unsigned long filter)
```

Parameters

Parameter	Description
short logType	Log Method. Valid values (defined in eoslog.h) are: LOGSYS_OS – Operating system logging. LOGSYS_NONE – No logging LOGSYS_COMM, LOGSYS_PIPE, LOGSYS_COMM_FILE, LOGSYS_FILE – For backwards compatibility, these values are accepted but are reassigned to LOGSYS_OS.
unsigned long filter	Bit map to be used for filtering log statements. For LOG_PRINTF requests, the information is only logged if the corresponding bits are set in both filters.

Return Values

0	Success
LOGSYS_NOT_INIT	No action taken. Application did not call LOG_INIT.

After initializing logging, it is sometimes useful to enable/disable or reset the filter. Two APIs are defined for this purpose.

LOG_SET_TYPE() allows the log type to be changed. The input parameter is handled the same as in LOG_INIT except the application's corresponding config.sys variable will not be checked.

LOG_SET_FILTER() allows the filter to be changed. The input parameter is handled the same as in LOG_INIT except the application's corresponding config.sys variable will not be checked.

For both APIs, the operation fails if the application has not called LOG_INIT.

Config.Sys Variables

Table 76 summarizes the configuration variables related to logging. The OS variables are checked at system start time and remain in effect until the next system restart. For full information on the OS variable settings, see Verix eVo Volume I: Operating System Programmers Manual, VPN - DOC00301.

Table 76 Config.sys Variables

Variable Name	GID	Description
<APPNAME>LOG	Current	Application specific configuration variable to override the LOG_INIT log type. Valid values are: <ul style="list-style-type: none"> • O (letter O): Operating System Logging • N: No logging • C, P, F, B: allowed for backwards compatibility. Translated to O. • Any other value: No logging.
<APPNAME>FIL	Current	Application specific configuration variable to override the LOG_INIT filter value.
*DEBUG	1	OS variable. Specifies the device to send OS logged data. Valid values are 1 (COM1), 2 (COM2), 0 (USB). This device will be reserved for the OS and not available for any other purposes.
*LOG	1	OS variable. If set, this specifies the size (KB) of a circular RAM buffer that will be used to store any data logged through the OS.
*DEBUGO		OS variable. Debug Options. Logging is enabled in various OS components using this value.
*LOGP	1	System Mode configuration variable. If *LOG is used, *LOGP specifies the device to send the log data. Valid values are 1 (COM1) and 2 (COM2). To use this mechanism, press the 4 th purple key on the System mode password screen or the System mode menu 1 screen.
#LOGPORT	15	No longer used

Samples

Snippets of sample application code are shown in the following figure:

```
// application has defined FILTER_SHOW_ALL_MESSAGES as
0xFFFFFFFF

LOG_INIT("TestApp", LOGSYS_OS, FILTER_SHOW_ALL_MESSAGES);
LOG_PRINTF(("Start"));

...

// application has defined FILTER_UI_LOGIC as 0x00100000.
LOG_PRINTF((FILTER_UI_LOGIC,"User Canceled Operation %d", rc));

...

// Application support library with error handling logging
hdl = open(REQUIRED_FILE, O_RDWR);
LOG_NEGATIVE_ERROR_CRIT_MOD("UtilLib", hdl, 0);
```

The corresponding log output, directly to a comm port, is shown in the following figure. In this case, an `open()` error occurred (`errno 2`) and is logged (`hdl` value is negative). If OS logging is enabled, the OS log information will be mixed with the application trace messages.

```
TESTAPP:F:main.c|L:00049|Start
TESTAPP:F:trans.cpp|L:00166|User Canceled Operation 4
TESTAPP:UtilLib:ER -1:ER -1 2|F:utilfile.c|L:00073|CRT
```

If the OS logging is configured to use the RAM buffer, then the System Mode user interface (fourth purple button on the System Mode password prompt or Menu 1 screen), can be used to transfer the data to a PC. A sample is shown below:

```
19 10:50:00.045 TESTAPP:F:main.c|L:00049|Start
19 10:50:01.030 TESTAPP:F:trans.cpp|L:00166|User Canceled
Operation 4
19 10:50:01.102 TESTAPP:UtilLib:ER -1:ER -1
2|F:utilfile.c|L:00073|CRT
```


Log Tracing Best Practices

Application logging provides a valuable tool for trouble-shooting individual applications as well as system issues. As previously described, the EOS Log Library supports mechanisms to enable, or disable, logging at run time on a task by task basis. In addition, the logging can be selectively filtered at a task level. The following guidelines provide information on how best to use the logging capabilities.

- Though specific application requirements will vary, the following approach should be considered for logging in a task:
 - Though the OS `SVC.H dbprintf()` method can be used to log information, EOS Log is the preferred method for released code. This is principally because EOS Log allows the logging to be controlled at a task level. In a multiple task environment, this can be critical to eliminate unrelated information from a system log.
 - In your task, consider initializing the EOS Log to disabled, example: `LOG_INIT(myAppName, LOGSYS_NONE, 0)`. Then use the EOS Log config variables to turn on and filter the logging for your task.
 - When logging for your task is enabled, the `LOG_PRINTF` statements will always be logged. So `LOG_PRINTF` is ideal for minimal information of general importance such as the version information and identifying special error conditions or logic flow. For detailed information on logic flow or features, use `LOG_PRINTF` and define filters for different functional areas or levels of detail.
- In the OS `GID1 config.sys`, variable `*DEBUG` specifies the system log device and variable `*LOG` enables the memory based logging. When using `*LOG`, variable `*LOGP` specifies the device where the log will be sent on demand. See the eVo OS Programmers Manual Volume I for a complete description. When setting these variables, keep the following in mind.
 - If logging directly to a comm port (example `*DEBUG=1, *LOG not set`), your task execution will be completely blocked while the log information is sent to the low level driver and out the comm port. In fact, no other tasks will be scheduled while this log statement is being transmitted. Though this is valuable in that it guarantees that the information is logged, it can greatly impact the performance and behavior of a system. If setting `*DEBUG` to a comm port in a complex environment (multiple tasks, using comm devices, handling time critical operations like contactless card transactions, etc.), keep in mind the potential impact on task scheduling.

- Using *LOG allows information to be logged with minimal impact to performance. Your task execution will still be blocked during the log operation but because writing to RAM is so much faster than to a serial device, the impact is minimal. To dump the *LOG information, the following approaches can be used:
 - In System Mode, the dual keypress of Backspace 4 will dump the log. This keypress can be used on either the System Mode Password prompt screen or Menu page 1. Note that while the System Mode Password prompt is displayed, applications are still executing. So one technique to view the log and not restart the system is to press Enter 7 to display the System Mode Password prompt screen. Then press Backspace 4 to dump the log. After the log is dumped, press cancel to re-activate the application (i.e. Do not enter System Mode because that will restart the system).
 - The EOS VxNCP has a feature to spool the memory based log to a comm port. Like other tasks, VxNCP will be scheduled and in the background, it will read the log and write to a comm port. Using this feature, you can view the log in “almost real time” without any user interaction (i.e. no need to press Backspace 4) while still minimizing the overhead of logging. The VxNCP variable *LOGSPOOLER enables this feature.

Backwards Compatibility Considerations Summary

Source Level

At the source level, the Verix eVo ELog library is compatible with earlier LogSys library versions with the following exceptions:

- LOG_PRINTF_BIN() is not supported.
- LOG_INIT parameter for log type: Values for pipe, comm., file and file/comm. are accepted but converted to “OS logging”.

Configurations

At runtime, ELog checks config.sys for configuration information. The Verix eVo ELog library configurations differs from earlier LogSys library versions as follows:

- GID 15 #LOGPORT is no longer used.
- GID 1 OS config.sys variables are used to configure the logging mechanism. These variables include *LOG, *LOGP, and *DEBUG.
- For application level configuration, the LogSys library still checks for the following variables in the current GID: <appname>FIL and <appname>LOG. With ELog, the valid values for <appname>LOG, are:
 - o (letter o): Operating System Logging
 - C, P, F, B: allowed for backwards compatibility. Translated to o.
- All other values are interpreted as no logging.

Behaviors

- Maximum message size is now 150, including the application name, file name, and line number.
- Only `LOG_PRINTF` macros use the filter settings. The `LOG_PRINTF` and error reporting macro output is not filtered. As long as logging is not disabled, these messages are always logged.
- For error checking macros, the formatted messages now also include `errno` information.



GLOSSARY

BCP Bluetooth Control Panel. The default user interface for performing Bluetooth operations.

CONFIG.SYS A compressed ASCII format file maintained as a keyed file.

CommEngine Verix eVo Communication Engine or CommEngine. The core component of the communication infrastructure. It determines the right drivers to load, identifies the communication devices present on the terminal, and then loads the right device driver. CommEngine also provides application services.

customer configuration file Allows customers to provide specific configuration values. Following filename, extension and format defined by Verix eVo, this file is provided during installation on GID 1. Verix eVo reads its default settings and overwrites them with the customer's settings.

data files All files used by the application, including CONFIG.SYS, files for batches, negative files, reports, and so on.

device driver A device driver is a software component that manages a device. It is distinct and different from the OS device driver. A device driver supports a published interface referred as device driver interface or DDI. It is not uncommon to refer to this device driver as DDI driver in common usage.

DTK Developer's toolkit. A package available from VeriFone containing tools to support application development in the Verix environment.

ELog Verix eVo Log library. It provides applications with a standard mechanism to log trace information that can be used for debugging and support.

file handle The operating system uses the file handle internally when accessing the file.

ICC Integrated circuit card; smart card.

IPP internal PIN pad. An internal device that allows a customer to enter a PIN.

LogSpooler Aims to enhance the logging capabilities of Verix eVo (EOS) by adding an option for the user to use other COM ports aside from COM1.

PIN Personal identification number. A security feature usually used in conjunction with ATM card transactions.

pipes A temporary software connection between two programs or commands.

SDLC Synchronous data link control is a data transmission protocol used by networks and conforming to IBMs Systems Network Architecture (SNA).

TLV Tag, Length, and Value. It consists of a fixed size Tag name size field followed by a variable length Tag name field. This pair of fields is followed by a fixed size Value length field and terminated by the variable length Value field.

WCP WiFi Control Panel. The default user interface for performing WiFi operations.



Symbols

*UGPHS8FW 242

Numerics

- 1 - Proceed 242
- 2 - Cancel 242
- 3 - Abort 242
- 3G Firmware Update 308
- 3G firmware update operation 308
- 3G Radio Firmware Update 241
 - Firmware Update Operation 241
 - Firmware Update Result 242
 - Firmware Update Status 242
 - User Prompt 241
 - Requirements 241
 - Firmware Files 241

A

- acronym list 12
- APIs
 - IP Address conversion
 - inet_addr() 77
 - inet_aton() 78
 - inet_ntoa() 79
- Application Programming Interfaces 9
- auto-reconnection 288

B

- Back-to-back Auto-Reconnection 287
- Backwards Compatibility 346
- BCP 329
- Bluetooth Control Panel 329
- Bluetooth events 284

C

- CDMA application events 283
- CE configuration section names 274
- CEIF 274
- CEs configuration key names 275
- Comm Technology 309
- CommEngine logs 278
- CommEngine Pipe 274
- Config.Sys Variables 343
- conventions
 - measurement 11
- Customization 301

D

- Data Structures 269
- Date Format 301
- ddi_bt_device_query_info parameters 286
- Development Toolkit 9
- download group 297
- download type 297
- DTK 9
- dual SIM 309

E

- ELog 333
- ELog library 333

F

- failed firmware update 242
- FTP 307

G

- Geo-Fence Downloadable File 268
- GPS 243
- GPS API 244
- GPS operations 269
- GPS port 243
- GPSD 243
- GPSD Application 243

I

- Idle Screen 303
- inet_addr() 77
- inet_aton() 78
- inet_ntoa() 79
- Interoperability 294
- Intervened Auto-Reconnection 287
- IP Status 309

L

- LOG_INVALID_POINTER,
LOG_INVALID_POINTER_MOD 338
- LOG_PRINTF, LOG_PRINTF_MOD 336
- LOG_PRINTFF, LOG_PRINTFF_MOD 337
- LogSpooler 322
- LOGSYS_FLAG 334
- LOGSYS_INIT 335

M

Macros for Reporting Errors 340

measurement conventions 11

Menu

- Bluetooth 321

- Tools 303

- WiFi 321

mobile phone (GSM) application events 282

N

Network Control Panel 293

Network Maintenance 305

NMEA sentences 243

P

PCI DSS 14

PHS8 3G 241

PHS8 3G radio module 241

port 2947 243

R

Re-assigning Log Settings 342

recovery mechanism 291

S

SDP 14

Site Data Protection 14

Startup Operation 294

successful firmware update 242

T

TCP/IP downloads 297

Tier one 287

Tier Two 287

Time Period 288

Two-tier 286

U

USB Drive 308

User Interface 300

V

validation 277

validation checking 277

Verix eVo Log 333

VMAC 298

W

WCP 325

WiFi Control Panel 325



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

Verix eVo Volume II: Operating System and Communication

Programmers Guide

