It's no trick...
it's a vision system

Vision
Components

The Smart Camera People

VCRT 5.0 TCP/ IP Functions

TCP/ IP Operation System Functions

Revision 5.01 Jan 2010
Document name: VCRT5_TCP_IP.pdf

Vision Components GmbH
Ottostraße 2 • D-76275 Ettlingen
Tel. +49(7243)2167-0
Fax +49(7243)2167-11
sales@vision-components.de

Foreword and Disclaimer

This documentation has been prepared with most possible care. However Vision Components GmbH does not take any liability for possible errors. In the interest of progress, Vision Components GmbH reserves the right to perform technical changes without further notice.
Please notify support@vision-components.com if you become aware of any errors in this manual or if a certain topic requires more detailed documentation.
This manual is intended for information of Vision Component's customers only. Any publication of this document or parts thereof requires written permission by Vision Components GmbH.

Please also consult the following resources for further reference:

Description	Titel on VC website	Download from Area
Installation Manual for VC20XX cameras	InstallVC20XX_VC40XX.pdf (1.36 MB)	Service & Support > Download-Center
Programming Tutorial Basic for VC20XX and VC40XX Cameras	Prog_Tut.pdf	Service & Support > Download-Center
Demonstration Source code	All Demo Programs	Service & Support > Download-Center
VCLIB 3.0 Image Processing Library Manual	VCLIB 3.0 Software Manual (539.67 Kbytes) VCLIB_300.pdf	Service & Support > Download-Center
VCRT 5 VCRT Functions	VCRT 5.0 Software Manual VCRT5.pdf	Service & Support > Download-Center

Note:

- **This document is valid for VC Smart Cameras with Texas Instrument DSP only!**
- **The remaining VCRT OS Functions are described in a separate document:** VCRT5.pdf (see list of references above).

The Light bulb highlights hints and ideas that may be helpful for a development.

This warning sign alerts of possible pitfalls to avoid. Please pay careful attention to sections marked with this sign.

Table of Contents

1 Definitions

Socket definition A socket is an abstraction that identifies an endpoint and includes:

type of socket; one of:
datagram (uses UDP)
stream (uses TCP)

socket address identified by:
port number
IP address
It may have a **remote endpoint**.

Socket options Each socket has socket options, which define characteristics of the socket, such as:
checksum calculations
Ethernet-frame characteristics
IGMP membership
non-blocking (nowait options)
push operations
sizes of send and receive buffers
timeouts

2 Datagram Sockets

Connectionless A datagram socket is connectionless in that an application uses a socket without first establishing a connection. Therefore, an application specifies the destination address and destination port number for each data transfer. An application can prespecify a remote endpoint for a datagram socket if desired.

Unreliable transfer A datagram socket is used for datagram-based data transfer, which does not acknowledge the transfer. Because delivery is not guaranteed, a higher layer is responsible for ensuring that the data is acknowledged when necessary.

Block oriented A datagram socket is block oriented. This means that when an application sends a block of data, the bytes of data remain together. If an application writes a block of data of, say, 100 bytes, VC/RT sends the data to the destination in a single packet, and the destination receives 100 bytes of data.

3 Stream Sockets

Connection based A stream-socket connection is uniquely defined by an address-port number pair for each of the two endpoints in the connection. For example, a connection to a Telnet server uses the local IP address with a local port number, and the server's IP address with port number 23.

Reliable transfer A stream socket provides reliable, end-to-end data transfer. To use stream sockets, a client establishes a connection to a peer, transfers data, and then closes the connection. Barring physical disconnection, VC/RT guarantees that all sent data is received in sequence.

Character oriented A stream socket is character oriented. This means that VC/RT may split or merge bytes of data as it sends the data from one protocol stack to another. An application on a stream socket may perform, for example, two successive write operations of 100 bytes each, and VC/RT may send the data to the destination in a single packet. The destination may then receive the data using, for example, four successive read operations of 50 bytes each.

4 Comparison of Datagram and Stream Sockets

	Datagram socket	**Stream socket**
Protocol	UDP	TCP
Connection based	No	Yes
Reliable transfer	No	Yes
Transfer mode	Block	Character

5 Creating and using Sockets

An application follows the following general steps to create and use sockets. The steps are summarized in the following diagrams and described in subsequent sections.

1. **Create a new socket** by calling socket(), indicating whether the socket is a datagram socket or a stream socket.
2. **Bind the socket** to a local address by calling bind().
3. If the socket is a stream socket, **assign a remote IP address** by doing one of the following:

3a. calling connect()

3b. calling listen() followed by accept()

4. **Send data** by calling sendto() for a datagram socket or send() for a stream socket.
5. **Receive data** by calling recvfrom() for a datagram socket or recv() for a stream socket.
6. When data transfer is finished, optionally **destroy the socket** by calling shutdown().

5.1 Diagram: Creating and Using Datagram Sockets (UDP)

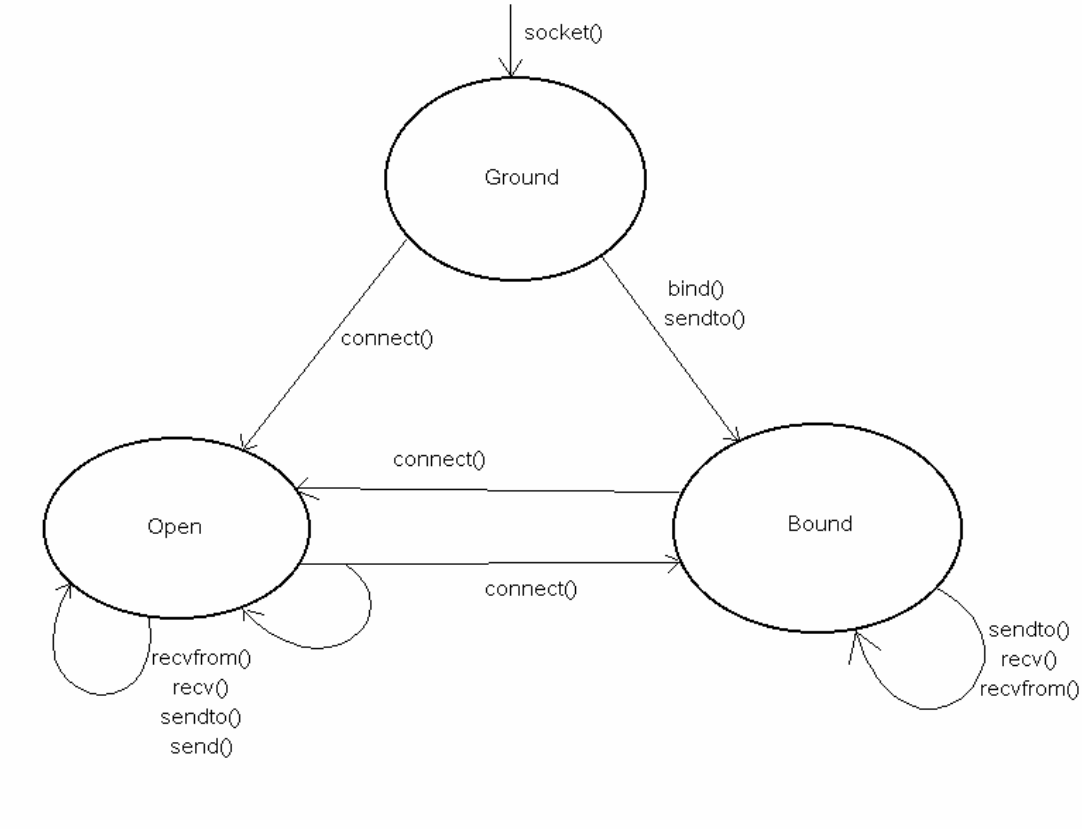

Diagram Creating and using datagram sockets (UDP)

5.2 Diagram: Creating and Using Stream Sockets (TCP)

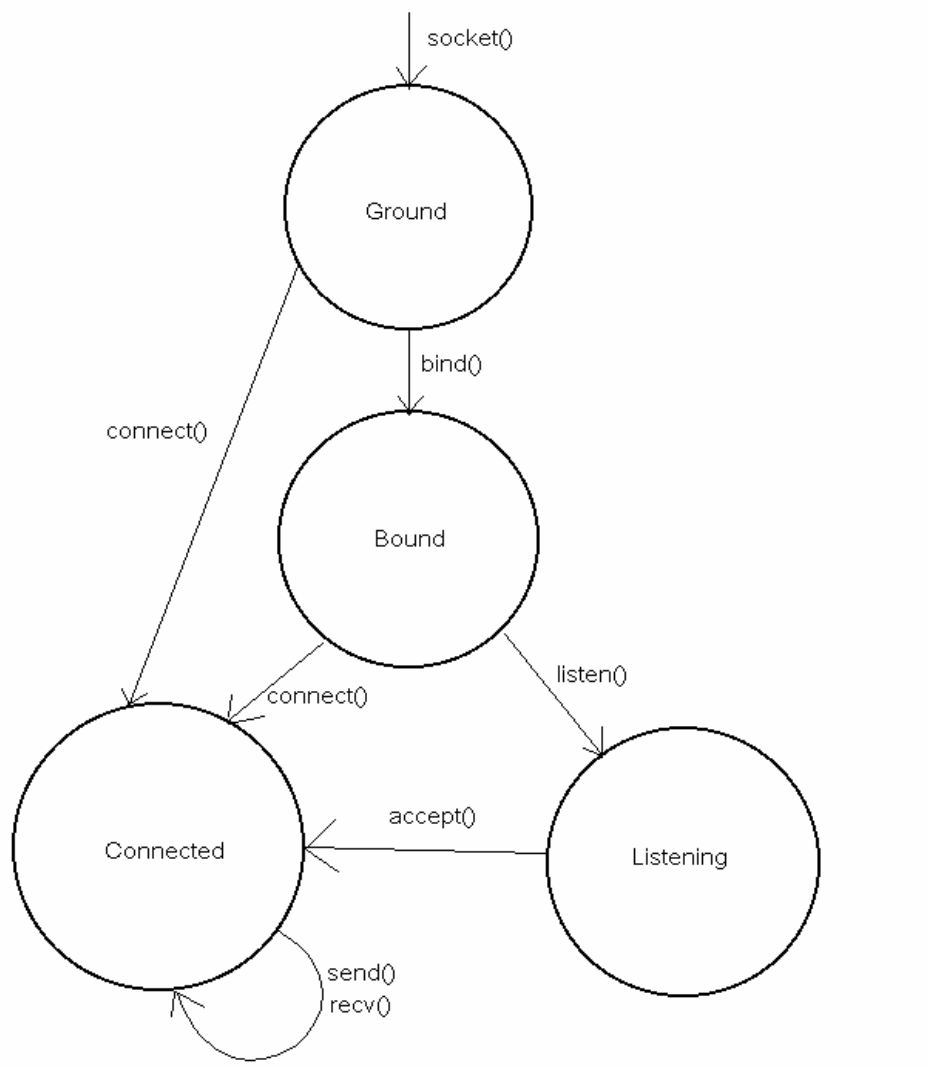

Diagram 2 Creating and using stream sockets (TCP)

6 Creating Sockets

To create a socket, an application calls socket() and specifies whether the socket is a datagram socket or a stream socket. The function returns a socket handle, which the application subsequently uses to access the socket.

7 Changing Socket Options

When VC/RT creates a socket, it sets all the socket options to **default values**.
To change the value of certain options, an application must do so **before it binds** the socket.

An application can change other options anytime.

All socket options and their default values are described in the following setsockopt().

8 Binding Sockets

After an application creates a socket and optionally changes or sets socket options, it must bind the socket to a local port number by calling bind(). The function defines the endpoint of the local socket by the local IP address and port number.

You can specify the local port number as any number, but if you specify zero, VC/RT chooses an unused port number. To determine the port number that VC/RT chose, call getsockname().
After the application binds the socket, how it uses the socket depends on whether the socket is a datagram socket or a stream socket.

9 Using Datagram Sockets

9.1 Setting Datagram Socket Options

By default, VC/RT uses IGMP, and, by default, a socket is not in any group. The application can change the following socket options for the socket:

- IGMP add membership
- IGMP drop membership
- send nowait
- checksum bypass

9.2 Transferring Datagram Data

An application transfers data by making calls to sendto() or send() and recvfrom() or recv().
With each call, VC/RT either sends or receives one UDP datagram,
which contains up to 65,507 bytes of data.

If an application specifies more data, the functions return an error.

The functions send() and sendto() return when the data is passed to the Ethernet interface.

The functions recv() and recvfrom() return when the socket port receives the packet or immediately if a queued packet is already at the port. The receive buffer should be at least as large as the largest datagram that the application expects to receive. If a packet overruns the receive buffer, VC/RT truncates the packet and discards the truncated data.

9.3 Buffering

By default, send() and sendto() do not buffer outgoing data.

This behavior can be changed by using either the OPT_SEND_NOWAIT socket option, or the VCRT_MSG_NONBLOCK send flag.

For incoming data, VC/RT matches the data, packet by packet, to recv() or recvfrom() calls that the application makes. If a packet arrives and a recv() or recvfrom() call is not waiting for data, VC/RT queues the packet.

9.4 Prescpecifying a peer

An application can optionally prespecify a peer by calling connect().
Prespecification has the following effect:

- send() can be used to send a datagram to the peer that is specified in the call to connect(). Calls to send() fail if connect() has not been called previously.
- the behavior of sendto() is unchanged. It is not restricted to the specified peer.
- the function recv() or recvfrom() returns datagrams that have been sent by the specified peer only

9.5 Shutting Down Datagram Sockets

An application can shut down a datagram socket by calling shutdown(). Before the function returns:

- outstanding calls to recvfrom() return immediately
- VC/RT discards received packets that are queued for the socket and frees their buffers

When shutdown() returns, the socket handle is invalid, and the application can no longer use the socket.

10 Using Stream Sockets

10.1 Changing Stream Socket Options

An application can change the value of certain stream-socket options anytime.
For details, see under setsockopt().

10.2 Establishing Stream Socket Connections

An application can establish a connection to a stream socket in one of these ways:

passively	by listening for incoming connection requests (by calling listen() followed by accept())
actively	by generating a connection request (by calling connect())

10.2.1 Passive Establishing

By calling listen(), an application can passively put an unconnected socket in a listening state, after which the local socket endpoint responds to a single incoming connection request.
After it calls listen(), the application calls accept(), which returns a new socket handle and lets the application accept the incoming connection request.

Usually,the application calls accept() immediately after it calls listen(). The application uses the new socket handle for all communication with the specified remote endpoint until one or both endpoints close the connection. The original socket remains in the listening state and continues to be referenced by the initial socket handle that socket() returned.

The new socket that the listen-accept mechanism creates inherits the socket options of the parent socket.

10.2.2 ActiveEstablishing

By calling connect(), an application can actively establish a stream-socket connection to the remote endpoint that the function specifies. If the remote endpoint is not in the listening state, connect() fails. Depending on the state of the remote endpoint, connect() fails immediately or after the time that the connect-timeout socket option specifies.
If the remote endpoint accepts the connection, the application uses the original socket handle for all its communication with that remote endpoint, and VC/RT maintains the connection until either or both endpoints close the connection.

10.3 Getting Stream Socket Names

After an application establishes a stream-socket connection, it can get the identifiers for the local endpoint (by calling getsockname()) and for the remote endpoint (by calling getpeername()).

10.4 Sending Stream Data

An application sends data on a stream socket by calling send(). When the function returns depends on the values of the send nowait (OPT_SEND_NOWAIT) socket option. An application can change the value by calling setsockopt().

10.4.1 Send nowait (nonblocking I/O)

send() returns **FALSE** (default) when TCP has buffered all data but not necessarily sent it
send() returns **TRUE** Immediately (the result is a filled or partially filled buffer)

10.5 Receiving Stream Data

An application receives data on a stream socket by calling recv(). The application passes the function a buffer, into which VC/RT places the incoming data. When the function returns depends on the values of the receive-nowait (OPT_RECEIVE_NOWAIT) and receive-push (OPT_RECEIVE_PUSH) socket options. The application can change the values by calling setsockopt().

Receive nowait (non-blocking I/O)	**Receive push (delay transmission)**	***recv*() returns when:**
FALSE (default)	TRUE (default)	One of: • a push flag in the data is received • supplied buffer is completely filled with incoming data • receive timeout expires (the default receive timeout is an unlimited time)
FALSE (default)	FALSE	Either: • supplied buffer is completely filled with incoming data • receive timeout expires
TRUE	(ignored)	Immediately after it polls TCP for any data in the internal receive buffer

10.6 Buffering Data

The size of the VC/RT per-socket send buffer is determined by the socket option that controls the size of the send buffer. VC/RT copies data into its send buffer from the buffer that the application supplies. As the peer acknowledges the data, VC/RT releases space in its buffer. If the buffer is full, calls to send() with the send-push (OPT_SEND_PUSH) socket option FALSE block until the remote endpoint acknowledges some or all the data.
The size of the VC/RT per-socket receive buffer is determined by the socket option that controls the size of the receive buffer. VC/RT uses the buffer to hold incoming data when there are no outstanding calls to recv(). When the application calls recv(), VC/RT copies data from its buffer to the buffer that the application supplies, and, consequently, the remote endpoint can send more data.

10.7 Improving the Throughput of Stream Data

- Include the push flag in sent data only where the flag is needed; that is, at the end of a stream of data.
- Specify the largest possible send and receive buffers to reduce the amount of work that the application and VC/RT do.
- When you call recv(), call it again immediately to reduce the amount of data that VC/RT must copy into its receive buffer.
- Specify the size of the send and receive buffers to be multiples of the maximum packet size.
- Call send() with an amount of data that is a multiple of the maximum packet size.

10.8 Shutting Down Stream Sockets

10.8.1 Shutting Down Gracefully

If the socket is to be shut down gracefully, VC/RT tries to deliver all the data that is in its send buffer for the socket. As specified by the TCP specification, VC/RT maintains the socket connection for four minutes after the remote endpoint disconnects.

10.8.2 Shutting Down with an abort operation

If the socket is to be shut down with an abort operation:
- VC/RT immediately discards the socket and the socket's internal send and receive buffers.
- The remote endpoint frees its socket immediately after it sends all the data that is in its send buffer.

11 Summary of Socket Functions

accept	Accept the next incoming stream connection and clone the socket to create a new socket, which services the connection
bind	Identify the local application endpoint by providing a port Number
connect	Establish a stream connection with an application endpoint or set a remote endpoint for a datagram socket
getpeername	Determine the peer address-port number endpoint of a connected socket
getsockname	Determine the local address-port number endpoint of a bound socket
getsockopt	Get the value of a socket option
listen	Allow incoming stream connections to be received on the port number that is identified by a socket
recv	Receive data on a stream or datagram socket
recvfrom	Receive data on a datagram socket
VCRT_attachsock	Get access to a socket that is owned by another task
VCRT_detachsock	Relinquish ownership of a socket
VCRT_geterror	Get the reason why an VC/RT function returned an error for the socket
VCRT_selectall	Wait for activity on any socket that a caller owns
VCRT_selectset	Wait for activity on any socket in a set of sockets
send	Send data on a stream socket or on a datagram socket for which a remote endpoint has been specified
sendto	Send data on a datagram socket
setsockopt	Set the value of a socket option
shutdown	Shutdown a connection and discard the socket
Socket_stream	Create a stream socket
Socket_dgram	Create a datagram socket

11.1 Accept

accept **create a new stream socket to accept incoming connections**

synopsis uint_32 accept(uint_32 socket, sockaddr_in *peeraddr, uint_16 *addrlen)

parameters

socket [IN]	Handle for the parent stream socket
peeraddr [OUT]	Pointer to where to place the remote endpoint identifier
addrlen [IN/OUT]	[IN] Pointer to the length, in bytes,of what *peeraddr* points to [OUT] Full size, in bytes, of the remote-endpoint identifier

returns Handle for a new stream socket
VCRT_SOCKET_ERROR

traits Blocks until an incoming connection is available

see also bind(), connect(), listen(), socket()

description The function accepts incoming connections by creating a new stream socket for the connections. The parent socket (socket) must be in the listening state; it remains in the listening state after each new socket is created from it. The new socket has the same local endpoint and socket options as the parent; the remote endpoint is the originator of the connection.

example

```
uint_32 handle;
uint_32 child_handle;
sockaddr_in remote_sin;
uint_16 remote_addrlen;
uint_32 status;
...
status = listen(handle, 0);
if (status != VCRT_OK)
{
    printf("\nError, listen() failed with error code %lx", status);
}
else
{
    remote_addrlen = sizeof(remote_sin);
    child_handle = accept(handle, &remote_sin, &remote_addrlen);
```

```
        if (child_handle != VCRT_SOCKET_ERROR)
        {
          printf("\nConnection accepted from %lx, port %d",
          remote_sin.sin_addr, remote_sin.sin_port);
      }
      else
      {
        status = VCRT_geterror(handle);
        if (status == VCRT_OK)
        {
          printf("\nConnection reset by peer");
        }
        else
        {
          printf("Error, accept() failed with error code %lx",
          status);
        }
      }
    }
```

11.2 bind

bind **bind the local address to the socket**

synopsis uint_32 bind(uint_32 socket, sockaddr_in *localaddr, uint_16 addrlen)

parameters

socket [IN]	Socket handle for the socket to bind
localaddr [IN]	Pointer to the local endpoint identifier to which to bind socket (see description)
addrlen [IN]	Length in bytes of what localaddr points to

returns VCRT _OK
Specific error code

traits Blocks, but VC/RT immediately services the command and is replied to by the socket layer

see also socket()

description

Field in sockaddr_in:	Must have this input value:
sin_family	AF_INET
sin_port	One of: •local port number for the socket •0 (To determine the port number that VC/RT chooses, call

	getsockname())
sin_addr	One of: •IP address that was previously bound •INADDR_ANY

Usually, TCP/IP servers bind to INADDR_ANY, so that one instance of the server can service all IP addresses.

example: Bind a socket to port number 2010.

```
uint_32 sock;
sockaddr_in local_sin;
uint_32 result;
...
sock = socket(AF_INET, SOCK_DGRAM, 0);
if (sock == VCRT_SOCKET_ERROR)
{
        printf("\nError, socket create failed");
        return;
}
memset((char *) &local_sin, 0, sizeof(local_sin));
local_sin.sin_family = AF_INET;
local_sin.sin_port = 2010;
local_sin.sin_addr.s_addr = INADDR_ANY;
result = bind(sock, &local_sin, sizeof (sockaddr_in));
if (status != VCRT_OK)
        printf("\nError, bind() failed with error code %lx",
result);
```

11.3 connect

connect	**Connect the stream socket to the remote endpoint**
synopsis	uint_32 connect(uint_32 socket, sockaddr_in *destaddr, uint_16 addrlen)

parameters	
socket [IN]	Handle for the stream socket to connect
destaddr [IN]	Pointer to the remote endpoint identifier
addrlen [IN]	Length in bytes of what destaddr points to

returns	VCRT _OK (success) Specific error code (failure)
traits	Blocks until the connection is accepted or until the connection-timeout socket option expires

see also accept(), bind(), getsockopt(), listen(), setsockopt(), socket_dgram(), socket_stream()

description

Stream socket:
The function fails if the remote endpoint rejects the connection request, which it may do immediately is unreachable, which causes the connection timeout to expire
If the function is successful, the application can use the socket to transfer data.

Datagram socket:
The function connect() has the following effects on a datagram socket:
send() can be used instead of sendto() to send a datagram to destaddr the behavior of sendto() is unchanged: it can still be used to send a datagramto any peer the socket receives datagrams from destaddr only
connect() may be used multiple times. Whenever **connect**() is called, the current endpoint is replaced by the new one.
A connection can be dissolved by calling connect() and specifying an address family of AF_UNSPEC. This dissolves the association, places the socket in the bound state, and returns the error code VCRTERR_SOCK_INVALID_AF.
Should connect() fail, the socket will be in a bound state (no remote endpoint).

example: Stream socket

```
uint_32 sock;
uint_32 child_handle;
sockaddr_in remote_sin;
uint_16 remote_addrlen = sizeof(sockaddr_in);
uint_32 result;
...
/* Connect to 192.203.0.83, port 2011: */
memset((char *) &remote_sin, 0, sizeof(sockaddr_in));
remote_sin.sin_family = AF_INET;
remote_sin.sin_port = 2011;
remote_sin.sin_addr.s_addr = 0xC0A80001; /* 192.168.0.1 */
result = connect(sock, &remote_sin, remote_addrlen);
if (result != VCRT_OK)
{
printf("\nError--connect() failed with error code %lx.",
result);
} else {
printf("\nConnected to %lx, port %d.",
remote_sin.sin_addr.s_addr, remote_sin.sin_port);
}
```


11.4 ENET_get_stats

ENET_get_stats	**get a pointer to the Ethernet statistics that VCRT collects**
synopsis	ENET_STATS *ENET_get_stats(enet_handle *handle)
parameters	handle [IN] Pointer to the Ethernet handle
returns	Pointer to an ENET_STATS structure
traits	
see also	ICMP_stats(), IP_stats(), IPIF_stats(), TCP_stats(), UDP_stats(),ENET_STATS
description	

example

```
ENET_STATS *enet;
_enet_handle ehandle;
...
enet = ENET_get_stats();
printf("\n%d Ethernet packets received", enet->ST_RX_TOTAL);
```

11.5 getpeername

getpeername	**get the remote-endpoint identifier of a socket**
synopsis	uint_32 getpeername(uint_32 socket, sockaddr_in *name, uint_16 *namelen)
Parameters	Handle for the stream socket
socket [IN]	
name [OUT]	Pointer to a placeholder for the remote-endpoint identifier of the socket
namelen [IN/OUT]	[IN] Pointer to the length, in bytes, of what name points to [OUT] Full size, in bytes, of the remote-endpoint identifier
returns	VCRT_OK (success) Specific error code (failure)
traits	Blocks, but the command is immediately serviced and replied to
see also	accept(), connect(), getsockname(), socket()

description The function returns the remote endpoint for the socket as was determined by connect() or accept().

example

```
uint_32 handle;
sockaddr_in remote_sin;
uint_32 status;
uint_16 namelen;
...
namelen = sizeof (sockaddr_in);
status = getpeername(handle, &remote_sin, &namelen); if (status != VCRT_OK)
{
printf("\nError, getpeername() failed with error code %lx",
status);
} else {
printf("\nRemote address family is %x", remote_sin.sin_family);
printf("\nRemote port is %d", remote_sin.sin_port);
printf("\nRemote IP address is %lx",
remote_sin.sin_addr.s_addr);
}
```

11.6 getsockname

getsockname **Get the local-endpoint identifier of the socket**

synopsis uint_32 getsockname(uint_32 socket,sockaddr_in *name, uint_16 *namelen)

parameters

socket [IN]	Socket handle
name [OUT]	Pointer to a placeholder for the local-endpoint identifier of the socket
namelen [IN/OUT]	[IN] Pointer to the length, in bytes, of what name points to [OUT] Full size, in bytes, of the remote-endpoint identifier
returns	VCRT_OK (success) Specific error code (failure)

traits Blocks, but the command is immediately serviced and replied to

see also bind(), getpeername(), socket()

description The function returns the local endpoint for the socket as was defined by bind().

example

```
uint_32 handle;
sockaddr_in local_sin;
uint_32 status;
uint_16 namelen;
...
namelen = sizeof (sockaddr_in);
status = getsockname(handle, &local_sin, &namelen);
if (status != VCRT_OK)
{
printf("\nError, getsockname() failed with error code %lx",
status);
} else {
printf("\nLocal address family is %x", local_sin.sin_family);
printf("\nLocal port is %d", local_sin.sin_port);
printf("\nLocal IP address is %lx", local_sin.sin_addr.s_addr);
}
```

11.7 getsockopt

getsockopt	**Get the value of the socket option**
synopsis	uint_32 getsockopt(uint_32 socket, int_32 level, uint_32 optname, pointer optval, uint_32 *optlen)
parameters	
socket [IN]	Socket handle
level [IN]	Protocol level at which the option resides
optname [IN]	Option name (see description)
optval [IN/OUT]	Pointer to the option value
optlen [IN/OUT]	[IN] Size of optval in bytes [OUT] Full size, in bytes, of the option value
returns	VCRT_OK (success) Specific error code (failure)
traits	Blocks, but the command is immediately serviced and replied to
see also	setsockopt()
description	An application can get all socket options for all protocol levels. For a complete description of socket options and protocol levels, see setsockopt().

11.8 listen

listen	**put the stream socket into the listening state**
synopsis	uint_32 listen(uint_32 socket, uint_16 backlog)
parameters	
socket [IN]	Socket handle
backlog [IN]	Ignored
returns	VCRT_OK (success) Specific error code (failure)
traits	Blocks, but the command is immediately serviced and replied to
see also	accept(), bind(), socket()
description	After the application calls listen(), it should call accept() to attach new sockets to the incoming requests.
example	See accept().

11.9 VCRT_ping

VCRT_ping	**send an ICMP echo-request packet to the IP address and wait for a reply**
synopsis	uint_32 VCRT_ping(ip_address address, uint_32 *timeout, uint_16 id)
parameters	
address [IN]	IP address to which to send the packet
timeout [IN/OUT]	[IN] One of: Pointer to the maximum time to wait for a reply 0 (wait indefinitely) [OUT] Pointer to the round-trip time
id [IN]	User ID for the echo request
returns	VCRT_OK (success) Error code (failure)

11.10 recv

recv	**provide VCRT with the buffer in which to place incoming stream data**
synopsis	int_32 recv(uint_32 socket, char *buffer, uint_32 buflen, uint_32 flags)
parameters	Handle for the connected stream socket
socket [IN]	
buffer [OUT]	Pointer to the buffer in which to place received data
buflen [IN]	Size of buffer in bytes
flags [IN]	Flags to underlying protocols. One of: VCRT_MSG_PEEK - For a UDP socket, receive a datagram but don't consume it (ignored for stream sockets) 0 - Ignore
returns	Number of bytes received (success) VCRT_ERROR (failure)
traits	May block, but the command is immediately serviced If non-blocking I/O is **disabled** on the socket, the function blocks until data satisfying the receive-push socket option is received If non-blocking I/O is **enabled** on the socket, the command is immediately replied to, returning whatever incoming data is buffered internally
see also	accept(), bind(), getsockopt(), listen(), VCRT_geterror(), send(), setsockopt(), shutdown(), socket()
description	When the flags parameter is VCRT_MSG_PEEK, the same datagram is received the next time recv() or recvfrom() is called. If the function returns VCRT_ERROR, the application can call VCRT_geterror() to determine the reason for the error. **Stream socket** If the receive-nowait socket option is TRUE, VCRT immediately copies internally buffered data (up to buflen bytes) into the buffer (at buffer), and recv() returns. If the

receive-wait socket option is TRUE, recv() blocks until the buffer is full or the receive-push socket option is satisfied. If the receive-push socket option is TRUE, a received TCP push flag causes recv() to return with whatever data has been received. If the receive-push socket option is FALSE, VCRT ignores incoming TCP push flags, and recv() returns when enough data has been received to fill the buffer.

Datagram socket
The recv() function on a datagram socket is identical to recvfrom() with NULL fromaddr and fromlen pointers. The recv() function is normally used on a connected socket.

example: Stream socket

```
uint_32 handle;
char buffer[20000];
uint_32 count;
...
count = recv(handle, buffer, 20000, 0);
if (count == VCRT_ERROR)
{
printf("\nError, recv() failed with error code %lx",
VCRT_geterror(handle));
} else {
printf("\nReceived %ld bytes of data.", count);
}
```

11.11 Recvfrom

recvfrom	**provide VC/RT with the buffer in which to place incoming datagram socket data**
synopsis	int_32 recvfrom(uint_32 socket, char *buffer, uint_32 buflen, uint_32 flags, sockaddr_in *fromaddr, uint_16 *fromlen)

parameters

socket [IN]	Handle for the datagram socket
buffer [OUT]	Pointer to buffer in which to place received data
buflen [IN]	Number of bytes in the buffer
flags [IN]	Flags to underlying protocols. One of: VCRT_MSG_PEEK - Receive a datagram but don't consume it 0 - Ignore
fromaddr [OUT]	Source socket address of the message
fromlen [IN/OUT]	[IN] Size of the fromaddr buffer

	[OUT] Size of the socket-address stored in the fromaddr buffer, or if the provided buffer was too small (socket-address was truncated), the length before truncation
returns	Number of bytes received (success) VCRT_ERROR (failure)
traits	Blocks until data is available or an error occurs
see also	bind(), VCRT_geterror(), sendto(), socket()
description	If a remote endpoint has been specified with connect(), only datagrams from that source will be received. When the flags parameter is VCRT_MSG_PEEK, the same datagram is received the next time recv() or recvfrom() is called. If fromlen is NULL, the socket address is not written to fromaddr. If fromaddr is NULL and the value of fromlen is not NULL, the result is unspecified. If the function returns VCRT_ERROR, the application can call VCRT_geterror() to determine the reason for the error.

example: **Receive up to 500 bytes of data.**

```
uint_32 handle;
sockaddr_in remote_sin;
uint_32 count;
char my_buffer[500];
uint_16 remote_len = sizeof(remote_sin);
...
count = recvfrom(handle, my_buffer, 500, 0,
&remote_sin, &remote_len);
if (count == VCRT_ERROR)
{
printf("\nrecvfrom() failed with error %lx",
VCRT_geterror(handle));
} else {
printf("\nReceived %ld bytes of data.", count);
}
```

11.12 VCRT_attachsock

VCRT_attachsock	**take ownership of the socket**
synopsis	uint_32 VCRT_attachsock(uint_32 socket)
parameters	socket [IN] Socket handle
returns	new socket handle (success) VCRT_SOCKET_ERROR (failure)
traits	Blocks, although the command is serviced and responded to immediately
see also	accept(), VCRT_detachsock()
description	The function adds the calling task to the socket's list of owners.
example	A main task loops to accept connections. When it accepts a connection, it creates a child task to manage the connection: it relinquishes control of the socket by calling VCRT_detachsock() and then creates the child with the accepted socket handle as the initial parameter.

```
while (TRUE)
{
 /* Issue ACCEPT: */
 TELNET_accept_skt =
 accept(TELNET_listen_skt, &peer_addr, &addr_len);
 if (TELNET_accept_skt != VCRT_SOCKET_ERROR)
{
   /* Transfer the socket and create the child task to look
after the    socket: */
   if (VCRT_detachsock(TELNET_accept_skt) ==
VCRT_OK)
 {
        child_task = (_task_create(LOCAL_ID,
CHILD),TELNET_accept_skt);
   }
 else
 {
        printf("\naccept() failed, error
        0x%lx", VCRT_geterror(TELNET_accept_skt));
   }
 }
}
```

11.13 VCRT_detachsock

VCRT_detachsock	**relinquish ownership of the socket**
synopsis	uint_32 VCRT_detachsock(uint_32 socket)
parameters	socket [IN] Socket handle from socket(), accept(), or VCRT_attachsock()
returns	VCRT_OK (success) Specific error code (failure)
traits	Blocks, although the command is serviced and responded to immediately
see also	accept(), VCRT_attachsock(), socket()
description	The function removes the calling task from the socket™s list of owners.
example	See VCRT_attachsock().

11.14 VCRT_geterror

VCRT_geterror	**Get the reason why an VC/RT function returned an error for the socket**
synopsis	uint_32 VCRT_geterror(uint_32 socket)
parameters	socket [IN] Socket handle
returns	VCRT_OK (no socket error) Last error code for the socket
traits	Does not block
see also	accept(), recv(), recvfrom(), send(), sendto()
description	Use the function if accept() returns VCRT_SOCKET_ERROR or any of the following functions returns VCRT_ERROR: recv() recvfrom() send() sendto()
example	See accept(), recv(), recvfrom(), send(), and sendto().

11.15 VCRT_selectall

VCRT_selectall **wait for activity on any socket that the caller owns**

synopsis uint_32 VCRT_selectall(uint_32 timeout)

parameters

timeout [IN] One of:
Maximum number of milliseconds to wait for activity
0 (wait indefinitely)
-1 (do not block)

returns Socket handle (activity was detected; see description)
0 (timeout expired)
VCRT_SOCKET_ERROR (error)

traits If timeout is not -1, blocks until activity is detected on any socket that the calling task owns

see also VCRT_selectset()

description Activity consists of any of the following.

This type of socket:	**Receives:**
Unbound datagram	Datagrams
Listening stream	Connection requests
Connected stream	Data or Shutdown requests that are initiated by the remote Endpoint

Example **Echo data on TCP port number 7.**

```
int_32 servsock;
int_32 connsock;
int_32 status;
SOCKET_ADDRESS_STRUCT addrpeer;
uint_16 addrlen;
char buf[500];
int_32 count;
uint_32 error
/* create a stream socket and bind it to port 7: */
error = listen(servsock, 0);
if (error != VCRT_OK) {
printf("\nlisten() failed, status = %d", error);
return;
}
for (;;) {
connsock = VCRT_selectall(0);
if (connsock == VCRT_SOCKET_ERROR) {
printf("\nVCRT_selectall() failed!");
} else if (connsock == servsock) {
```

```
status = accept(servsock, &addrpeer, &addrlen);
if (status == VCRT_SOCKET_ERROR)
printf("\naccept() failed!");
} else {
count = recv(connsock, buf, 500, 0);
if (count <= 0)
shutdown(connsock, FLAG_CLOSE_TX);
else
send(connsock, buf, count, 0);
}
}
```

11.16 VCRT_selectset

VCRT_selectset		**wait for activity on any socket in the set of sockets**
synopsis		uint_32 VCRT_selectset(pointer sockset, uint_32 count, uint_32 timeout)
parameters	sockset [IN]	Pointer to an array of sockets
	count [IN]	Number of sockets in the array
	timeout [IN]	One of: Maximum number of milliseconds to wait for activity 0 (wait indefinitely) -1 (do not block)
returns		Socket handle (activity was detected) 0 (timeout expired) VCRT_SOCKET_ERROR (error)
traits		If timeout is not -1, blocks until activity is detected on at least one of the sockets in the set
see also		VCRT_selectall()
description		For the description of what constitutes activity, see VCRT_selectall().
Example		**Echo UDP data that is received on ports 2010, 2011, and 2012.**

```
int_32 socklist[3];
sockaddr_in local_sin;
uint_32 result;
...
memset((char *) &local_sin, 0, sizeof(local_sin));
local_sin.sin_family = AF_INET;
local_sin.sin_addr.s_addr = INADDR_ANY;
```

```
local_sin.sin_port = 2010;
socklist[0] = socket(AF_INET, SOCK_DGRAM, 0);
result = bind(socklist[0], &local_sin, sizeof (sockaddr_in));
local_sin.sin_port = 2011;
socklist[1] = socket(AF_INET, SOCK_DGRAM, 0);
result = bind(socklist[1], &local_sin, sizeof (sockaddr_in));
local_sin.sin_port = 2012;
socklist[2] = socket(AF_INET, SOCK_DGRAM, 0);
result = bind(socklist[2], &local_sin, sizeof (sockaddr_in));
while (TRUE) {
sock = VCRT_selectset(socklist, 3, 0);
rlen = sizeof(raddr);
length = recvfrom(sock, buffer, BUFFER_SIZE, 0, &raddr, &rlen);
sendto(sock, buffer, length, 0, &raddr, rlen);
}
```

11.17 send

send	**Send data on the stream socket, or on a datagram socket for which a remote endpoint has been specified.**	
synopsis	int_32 send(uint_32 socket, char *buffer, uint_32 buflen, uint_32 flags)	
parameters		
socket [IN]	Handle for the socket on which to send data	
buffer [IN]	Pointer to the buffer of data to send	
buflen [IN]	Number of bytes in the buffer (no restriction)	
flags [IN]	Flags to underlying protocols, selected from three independent groups. Perform a bitwise OR of one flag only from one or more of the following groups: **Group 1:**	
	VCRT_MSG_BLOCK	Override the OPT_SEND_NOWAIT datagram socket option; make it behave as though it is FALSE
	VCRT_MSG_NONBLOCK	Override the OPT_SEND_NOWAIT datagram socket option; make it behave as though it is TRUE
	Group 2:	
	VCRT_MSG_CHKSUM	Override the OPT_CHECKSUM_BYPASS checksum bypass option; make it behave as though it is FALSE
	VCRT_MSG_NOCHKSUM	Override the

		OPT_CHECKSUM_BYPASS checksum bypass option; make it behave as though it is TRUE
	Group 3:	
	VCRT_MSG_NOLOOP	Do not send the datagram to the loopback interface
	0	Ignore

returns

Number of bytes sent (success)
VCRT_ERROR (failure)

traits

May block until data is placed in the socket's send buffer, whose size is set by setsockopt()

see also

accept(), bind(), getsockopt(), listen(), recv(), VCRT_geterror(), setsockopt(), shutdown(), socket()

description

If the function returns VCRT_ERROR, the application can call VCRT_geterror() to determine the cause of the error.

Stream socket
VC/RT packetizes the data (at buffer) into TCP packets and delivers the packets reliably and sequentially to the connected remote endpoint.

If the send-nowait socket option is TRUE, VC/RT immediately copies the data into the internal send buffer for the socket, to a maximum of buflen. The function then returns.

If the send-push socket option is TRUE, VC/RT appends a push flag to the last packet that it uses to send the buffer; all data is sent immediately, taking into account the capabilities of the remote endpoint buffer.

Datagram socket If a remote endpoint has been specified using connect(),

send() is identical to sendto() using the specified remote endpoint. If a remote endpoint has not been specified, send() returns VCRT_ERROR.

The flags parameter can be used for datagram sockets only. The override is temporary and lasts for the current call to send() only.

Setting flags to VCRT_MSG_NOLOOP is useful when broadcasting or multicasting a datagram to several destinations. When flags is set to VCRT_MSG_NOLOOP, the datagram is not duplicated for the local host interface.

example: Stream socket

```
uint_32 handle;
char buffer[20000];
uint_32 count;
...
count = send(handle, buffer, 20000, 0);
if (count == VCRT_ERROR)
  printf("\nError, send() failed with error code %lx", VCRT_geterror(handle));
```

11.18 Sendto

sendto **send data on the datagram socket**

synopsis int_32 sendto(uint_32 socket, char *buffer, uint_32 buflen, uint_16 flags, sockaddr_in *destaddr, uint_16 addrlen)

parameters

socket [IN]	Handle for the socket on which to send data	
buffer [IN]	Pointer to the buffer of data to send	
buflen [IN]	Number of bytes in the buffer	
flags [IN]	Flags to underlying protocols, selected from three independent groups. Perform a bitwise OR of one flag only from one or more of the ollowing groups: **Group 1:**	
	VCRT_MSG_BLOCK	Override the OPT_SEND_NOWAIT datagram socket option; make it behave as though it is FALSE
	VCRT_MSG_NONBLOCK	Override the OPT_SEND_NOWAIT datagram socket option; make it behave as though it is TRUE
	Group 2:	
	VCRT_MSG_CHKSUM	Override the OPT_CHECKSUM_BYPASS checksum bypass option; make it behave as though it is FALSE
	VCRT_MSG_NOCHKSUM	Override the OPT_CHECKSUM_BYPASS checksum bypass option; make it behave as though it is TRUE
	Group 3:	
	VCRT_MSG_NOLOOP	Do not send the datagram to the loopback interface
	0	Ignore
destaddr [IN]	Remote endpoint identifier to which to send the data	

addrlen [IN] Number of bytes pointed to by destaddr

returns Number of bytes sent (success)
VCRT_ERROR (failure)

traits Blocks, but the command is immediately serviced and replied to

see also setsockopt(), bind(), recvfrom(), VCRT_geterror(), socket()

description The function sends the data (at buffer) as a UDP datagram to the remote endpoint (at destaddr).

This function can also be used when a remote endpoint has been prespecified through connect(). The datagram is sent to destaddr even if it is different than the prespecified remote endpoint.
If the socket address has been prespecified, you can call sendto() with destaddr set to NULL and addrlen equal to zero: this combination sends to the prespecified address. Calling sendto() with destaddr set to NULL and addrlen equal to zero without first having prespecified the destination will result in an error.
The flags parameter can be used for datagram sockets only. The override is temporary and lasts for the current call to sendto() only. Setting flags to VCRT_MSG_NOLOOP is useful when broadcasting or multicasting a datagram to several destinations. When flags is set to VCRT_MSG_NOLOOP, the datagram is not duplicated for the local host interface.
If the function returns VCRT_ERROR, the application can call VCRT_geterror() to determine the cause of the error.

Example **Send 500 bytes of data to IP address 192.203.0.54, port number 678.**

```
uint_32 handle;
sockaddr_in remote_sin;
uint_32 count;
char my_buffer[500];
...
for (i=0; i < 500; i++) my_buffer[i]= (i & 0xff);
memset((char *) &remote_sin, 0, sizeof(sockaddr_in));
remote_sin.sin_family = AF_INET;
remote_sin.sin_port = 678;
remote_sin.sin_addr.s_addr = 0xC0CB0036;
count = sendto(handle, my_buffer, 500, 0, &remote_sin,
sizeof(sockaddr_in));
if (count != 500)
printf("\nsendto() failed with count %ld and error %lx",
count, VCRT_geterror(handle));
```

11.19 setsockopt

setsockopt	**set the value of the socket option**
synopsis	uint_32 setsockopt(uint_32 socket, uint_32 level, uint_32 optname, pointer optval, uint_32 optlen)
parameters	
socket [IN]	One of: If level is anything but SOL_NAT, handle for the socket whose option is to be changed If level is SOL_NAT, socket is ignored
level [IN]	Protocol level at which the option resides; one of: SOL_IGMP SOL_LINK SOL_NAT (not available) SOL_SOCKET SOL_TCP SOL_UDP
optname [IN]	Option name; see description
optval [IN]	Pointer to the option value
optlen [IN]	Number of bytes that optval points to
returns	VCRT_OK (success) Specific error code (failure)
traits	Blocks, but the command is immediately serviced and replied to
see also	bind(), getsockopt(),ip_mreq, nat_ports, nat_timeouts
Description	You can set most socket options by calling setsockopt(). However, the following options cannot be set; you can use them only with getsockopt(): IGMP get membership receive Ethernet 802.1Q priority tags receive Ethernet 802.3 frames socket error socket type Settable options have default values. If you want to change the value of some settable options, you must do so before you bind the socket. For other settable options, you can change the value anytime after the socket is created.
NOTE	Some options can be temporarily overridden for datagram sockets. For more information, see send() and sendto().

11.19.1 Description of Socket Options

- **OPT_CHECKSUM_BYPASS**

Checksum bypass
Option name OPT_CHECKSUM_BYPASS (can be overridden)
Protocol level SOL_UDP
Values TRUE
VC/RT sets to 0 the checksum field of sent datagram packets, and the generation of checksums is bypassed

FALSE
VC/RT generates checksums for sent datagram packets
Default value FALSE
Change Before bound
Socket type Datagram
Comments

- **OPT_CONNECT_TIMEOUT**

Connect timeout
Option name OPT_CONNECT_TIMEOUT
Protocol level SOL_TCP
Values >= 180,000 VC/RT maintains the connection for this number of milliseconds
Default value 480,000 (8 min)
Change Before bound
Socket type Stream
Comments Connect timeout corresponds to R2 (as defined in RFC 793) and is sometimes called the hard timeout. It indicates how much time VC/RT spends attempting to establish a connection before it gives up. If the remote endpoint does not acknowledge a sent segment within the connect timeout (as would happen if a cable breaks, for example), VC/RT shuts down the socket connection, and all function calls that use the connection return.

- **VCRT_SO_IGMP_ADD_MEMBERSHIP**

IGMP add membership
Option name VCRT_SO_IGMP_ADD_MEMBERSHIP
Protocol level SOL_IGMP
Values
Default value Not in a group
Change Anytime
Socket type Datagram
Comments IGMP must be in the VC/RT protocol table.

Example
To join a multicast group:
```
uint_32 sock;
```

```
struct ip_mreq group;
group.imr_multiaddr = multicast_ip_address;
group.imr_interface = local_ip_address;
error = setsockopt(sock, SOL_IGMP, VCRT_SO_IGMP_ADD_MEMBERSHIP,
&group,sizeof(group));
```

- **VCRT_SO_IGMP_DROP_MEMBERSHIP**

IGMP drop membership
Option name VCRT_SO_IGMP_DROP_MEMBERSHIP
Protocol level SOL_IGMP
Values
Default value Not in a group
Change After the socket is created
Socket type Datagram
Comments IGMP must be in the VC/RT protocol table.

Example
To leave a multicast group:

```
uint_32 sock;
struct ip_mreq group;
group.imr_multiaddr = multicast_ip_address;
group.imr_interface = local_ip_address;
error = setsockopt(sock,SOL_IGMP,VCRT_SO_IGMP_DROP_MEMBERSHIP,
&group,sizeof(group));
```

- **VCRT_SO_IGMP_GET_MEMBERSHIP**

IGMP get membership
Option name VCRT_SO_IGMP_GET_MEMBERSHIP
Protocol level SOL_IGMP
Values
Default value Not in a group
Change (use with getsockopt() only; returns value in optval)
Socket type Datagram
Comments

- **OPT_RETRANSMISSION_TIMEOUT**

Initial retransmission timeout
Option name OPT_RETRANSMISSION_TIMEOUT
Protocol level SOL_TCP
Values >= 15 ms (See comments)
Default value 3000 (3 seconds)
Change Before bound
Socket type Stream
Comments Value is a first, best guess of the round-trip time for a stream socket packet. VC/RT attempts to resend the packet if it does not receive an acknowledgment in this time. After a connection is established, VC/RT determines the retransmission timeout, starting from this initial value.

If the initial retransmission timeout is not longer than the end-to-end acknowledgment time expected on the socket, the connect timeout will expire prematurely.

- **OPT_KEEPALIVE**

Keep-alive timeout
Option name OPT_KEEPALIVE
Protocol level SOL_TCP
Values 0

VC/RT does not probe the remote endpoint

non-zero

If the connection is idle, VC/RT periodically probes the remote endpoint, an action that detects whether the remote endpoint is still present
Default value 0 minutes
Change Before bound
Socket type Stream
Comments The option is not a standard feature of the TCP/IP specificationand generates unnecessary periodic network traffic

- **OPT_MAXRTO**

Maximum retransmission timeout
Option name OPT_MAXRTO
Protocol level SOL_TCP
Values non-zero

Maximum value for the retransmission timers exponential backoff

0

VC/RT uses the default value, which is 2 times the maximum segment lifetime (MSL).
Since the MSL is 2 minutes, the MTO is 4 minutes.
Default value 0 milliseconds
Change Before bound
Socket type Stream
Comments The retransmission timer is used for multiple retransmissions of a segment.

- **OPT_NO_NAGLE_ALGORITHM**

No Nagle algorithm
Option name OPT_NO_NAGLE_ALGORITHM
Protocol level SOL_TCP
Values TRUE
VC/RT does not use the Nagle algorithm to coalesce short segments

FALSE
To reduce network congestion, VC/RT uses the Nagle algorithm (defined in RFC 896) to coalesce short segments
Default value FALSE
Change Before bound
Socket type Stream
Comments If an application intentionally sends short segments, it can improve efficiency by setting the option to TRUE

- **OPT_RBSIZE**

Receive-buffer size
Option name OPT_RBSIZE
Protocol level SOL_TCP
Values Recommended to be a multiple of the maximum segment size, where the multiple is at least three
Default value 4380 bytes
Change Before bound
Socket type Stream
Comments When the socket is bound, VC/RT allocates a receive buffer of the specified number of bytes, which controls how much received data VC/RT can buffer for the socket

- **VCRT_SO_LINK_RX_8021Q_PRIO**

Receive Ethernet 802.1Q priority tags
Option name VCRT_SO_LINK_RX_8021Q_PRIO
Protocol level SOL_LINK
Values -1
The last received frame did not have an Ethernet 802.1Q priority tag

0..7
The last received frame had an Ethernet 802.1Q priority tag with the specified priority
Default value
Change (use with getsockopt() only; returns value in optval)
Socket type Stream (Ethernet)
Comments Returned information is for the last frame that the socket received

- **VCRT_SO_LINK_RX_8023**

Receive Ethernet 802.3 frames
Option name VCRT_SO_LINK_RX_8023
Protocol level SOL_LINK
Values TRUE
The last received frame was an 802.3 frame

FALSE
The last received frame was an Ethernet II frame
Default value
Change (use with getsockopt() only; returns value in optval)
Socket type Stream (Ethernet)
Comments Returned information is for the last frame that the socket received

- **OPT_RECEIVE_NOWAIT**

Receive nowait
Option name OPT_RECEIVE_NOWAIT
Protocol level SOL_TCP
Values TRUE
recv() returns immediately, regardless of whether there is data to be received
FALSE
recv() waits until there is data to be received
Default value FALSE
Change Anytime
Socket type Stream
Comments

- **OPT_RECEIVE_PUSH**

Receive push
Option name OPT_RECEIVE_PUSH
Protocol level SOL_TCP
Values TRUE
recv() returns immediately if it receives a push flag from the remote endpoint, even if the specified receive buffer is not full

FALSE
recv() ignores push flags and returns only when its buffer is full or if the receive timeout expires
Default value TRUE
Change Anytime
Socket type Stream
Comments

- **OPT_RECEIVE_TIMEOUT**

Receive timeout
Option name OPT_RECEIVE_TIMEOUT
Protocol level SOL_TCP
Values 0
VC/RT waits indefinitely for incoming data during a call to recv()

non-zero
VC/RT waits for this number of milliseconds for incoming data during a call to recv()
Default value 0 milliseconds
Change Anytime
Socket type Stream
Comments When the timeout expires, recv() returns with whatever data that has been received

- **OPT_TBSIZE**

Send-buffer size
Option name OPT_TBSIZE
Protocol level SOL_TCP
Values Recommended to be a multiple of the maximum segment size, where the multiple is at least three
Default value 4380 bytes
Change Before bound
Socket type Stream
Comments When the socket is bound, VC/RT allocates a send buffer of the specified number of bytes, which controls how much sent data VC/RT can buffer for the socket

- **VCRT_SO_LINK_TX_8021Q_PRIO**

Send Ethernet 802.1Q priority tags
Option name VCRT_SO_LINK_TX_8021Q_PRIO
Protocol level SOL_LINK
Values -1
VC/RT does not include Ethernet 802.1Q priority tags

0-7
VC/RT includes Ethernet 802.1Q priority tags with the specified priority
Default value -1
Change Anytime
Socket type Stream (Ethernet)
Comments

- **VCRT_SO_LINK_TX_8023**

Send Ethernet 802.3 frames
Option name VCRT_SO_LINK_TX_8023
Protocol level SOL_LINK

Values TRUE
VC/RT sends 802.3 frames
FALSE
VC/RT sends Ethernet II frames
Default value FALSE
Change Anytime
Socket type Stream (Ethernet)
Comments Returns information for the last frame that the socket received

- **OPT_SEND_NOWAIT (StreamSocket)**

Send nowait (stream socket)
Option name OPT_SEND_NOWAIT
Protocol level SOL_TCP
Values TRUE
Task that calls send() does not wait if data is waiting to be sent; VC/RT buffers the outgoing data, and send() returns immediately

FALSE
Task that calls send() waits if data is waiting to be sent
Default value FALSE
Change Anytime
Socket type Stream
Comments

- **OPT_SEND_NOWAIT (Datagram Socket)**

Send nowait (datagram socket)
Option name OPT_SEND_NOWAIT (can be overridden)
Protocol level SOL_UDP
Values TRUE
VC/RT buffers every datagram and send() or sendto() returns immediately

FALSE
Task that calls send() or sendto() blocks until the datagram has been transmitted. Datagrams are not copied.
Default value FALSE
Change Anytime
Socket type Datagram
Comments

- **OPT_SEND_PUSH**

Send push
Option name OPT_SEND_PUSH
Protocol level SOL_TCP
Values TRUE

If possible, VC/RT appends a send-push flag to the last packet in the segment of the data that is associated with send() and immediately sends the data. A call to send() may block until another task calls send() for that socket.

FALSE

Before it sends a packet, VC/RT waits until it has received from the host enough data is completely fill the packet

Default value TRUE
Change Anytime
Socket type Stream
Comments

- **OPT_SOCKET_ERROR**

Socket error
Option name OPT_SOCKET_ERROR
Protocol level SOL_SOCKET
Values
Default value
Change (use with getsockopt() only; returns value in optval)
Socket type Datagram or stream
Comments Returns the last error for the socket

- **OPT_SOCKET_TYPE**

Socket type
Option name OPT_SOCKET_TYPE
Protocol level SOL_SOCKET
Values
Default value
Change (use with getsockopt() only; returns value in optval)
Socket type Datagram or stream
Comments Returns the type of socket (SOCK_DGRAM or SOCK_STREAM)

- **OPT_TIMEWAIT_TIMEOUT**

Timewait timeout
Option name OPT_TIMEWAIT_TIMEOUT
Protocol level SOL_TCP
Values > 0 ms
Default value 2 times the maximum segment lifetime (which is a constant)
Change Before bound
Socket type Stream
Comments Timewait timeout is the number of milliseconds that TCP waits in the timewait state

11.19.2 Example: Change send-push option to FALSE

```
uint_32 handle;
uint_32 opt_length = sizeof(uint_32);
uint_32 opt_value = FALSE;
uint_32 status;
...
status = setsockopt(handle, 0, OPT_SEND_PUSH,
&opt_value, opt_length);
if (status != VCRT_OK)
printf("\nsetsockopt() failed with error %lx", status);
status = getsockopt(handle, 0, OPT_SEND_PUSH,
&opt_value, &opt_length);
if (status != VCRT_OK)
printf("\ngetsockopt() failed with error %lx", status);
```

11.19.3 Example: Change receive nowait option to TRUE

```
uint_32 handle;
uint_32 opt_length = sizeof(uint_32);
uint_32 opt_value = TRUE;
uint_32 status;
...
status = setsockopt(handle, 0, OPT_RECEIVE_NOWAIT,
&opt_value, opt_length);
if (status != VCRT_OK)
printf("\nError, setsockopt() failed with error %lx", status);
```

11.19.4 Example: Change Cecksum Bypass option to TRUE

```
uint_32 handle;
uint_32 opt_length = sizeof(uint_32);
uint_32 opt_value = TRUE;
uint_32 status;
...
status = setsockopt(handle, SOL_UDP, OPT_CHECKSUM_BYPASS,
&opt_value, opt_length);
if (status != VCRT_OK)
printf("\nError, setsockopt() failed with error %lx", status);
```


11.20 shutdown

shutdown **shut down the socket**

synopsis uint_32 shutdown(uint_32 socket, uint_16 how)

parameters

socket [IN]	Handle of the socket to shut down
how [IN]	One of (see description): FLAG_CLOSE_TX FLAG_ABORT_CONNECTION

returns
VCRT_OK
Specific error code

traits Blocks, but the command is processed and returned to immediately
The application can no longer use socket

see also socket_dgram, socket_stream

description

Type of socket	**Value of *how***	***shutdown*n() does the following:**
Datagram	(ignored)	Shuts down *socket* immediately •Calls to *recvfro*m() return immediately •Discards queued incoming packets
Unconnected stream	(ignored)	Shuts down *socket* immediately
Connected stream	FLAG_CLOSE_ TX	Gracefully shuts down *socke*t, ensuring that all sent data is acknowledged Calls to *sen*d() and *rec*v() return immediately
	FLAG_ABORT_ CONNECTION	If VC/RT is originating the disconnection, it maintains the internal socket context for 4 min. (twice the maximum TCP segment lifetime) after the remote endpoint closes the connection Immediately discards the internal socket context Sends a TCP reset packet to the remote endpoint Calls to *sen*d() and *rec*v() return immediately

example

```
uint_32 handle;
uint_32 status;
...
status = shutdown(handle, 0);
if (status != VCRT_OK)
printf("\nError, shutdown() failed with error code %lx",
status);
```

11.21 socket_stream

socket_stream	**create a stream socket**
synopsis	uint_32 socket_stream(void)
parameters	none
returns	Socket handle (success) VCRT_SOCKET_ERROR (failure)
traits	Blocks, although the command is serviced and responded to immediately
see also	bind()
description	The application uses the socket handle to subsequently use the socket.
example	See bind().

11.22 socket_dgram

socket_dgram	**create a datagram socket**
synopsis	uint_32 socket_dgram(void)
parameters	none
returns	Socket handle (success) VCRT_SOCKET_ERROR (failure)
traits	Blocks, although the command is serviced and responded to immediately
see also	bind()
description	The application uses the socket handle to subsequently use the socket.
example	See bind

Index

Visit the Vision Components site **www.vision-components.com** for further information and documentation and hardware or software downloads.

Vision Components GmbH
Ottostraße 2 • D-76275 Ettlingen
Tel. +49(7243)2167-0
Fax +49(7243)2167-11
sales@vision-components.de