Socket Classes – Low Level Networking Interface

Allen Goldstein, NIST

1. Introduction

The Socket Classes are a hierarchy of low-level network interfacing classes for TCP and UDP interfaces. The parent class for these is the Socket.Base class, then each socket type exists as a child class of Socket.Base. The parent class: "Socket" has 5 polymorphic methods: New, Connect, Transmit, Receive and Close. When placed into a block diagram and chained into either the TCP or UDP class control, the VI will automatically be overridden by the child class method. The parent Socket class method should not be used without the TCP or UDP class control input and will output override error 20000 if used alone. The below figure shows the parent class and child class methods as they appear before and after being overridden:

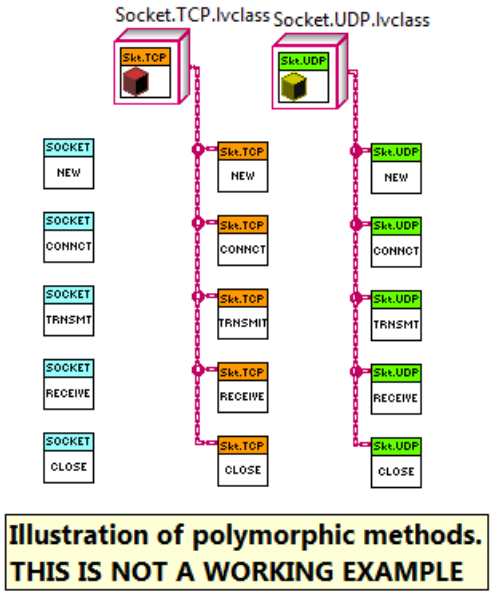

Illustration of polymorphic methods.
THIS IS NOT A WORKING EXAMPLE

Additionally, there is a set of TCP Listen methods in the TCP child class which support the creation and destruction of a TCP listener. Examples of TCP and UDP servers and clients are available in the LabVIEW example finder.

2. Socket.Base

The Base Sockets class. Only child classes (TCP, UDP) should be used. This base class should not be used directly; it must be overridden.

2.1 Properties

i32 **timeout**
The amount of time, in milliseconds, that a timed method (such as Receive or Transmit) will wait before moving on with a timeout error. -1 will cause the timeout to wait forever.

bool **connected**
Indicates that the socket is connected to another socket on a remote, or local host.

bool **initialized**

defaults to false and set to true by the *New* method.

2.1.1 Property Accessors

Properties should be accessed using property nodes. Property accessor vis can be used but are not recommended. Since accessor vis must exist in order to access property via property nodes, they are all found in the "Accessors" folders in the various classes. These can be ignored by the user.

2.2 Methods

socket.**New**(timeout)
Creates a new socket object. Sets the timeout and "initialized" property.

i32 **timeout** network timeout in milliseconds. Default is 10,000 ms (10 seconds).

socket.**Connect** (remote.address, remote_port, [host_address], [host_port])

(Must Be overridden) Establishes a remote connection. See the individual child classes for a description of how the input parameters are used.

socket **Close()**

(Must be overridden.) Closes an open connection.

socket.tcp.**Receive** (mode, bufsize, [clear_timeout_error])

(Must be overridden) Returns bytes received or aits for the timeout period in ms and produces a timeout error. See the individual child classes for the input parameter usage

socket.tcp.**Transmit** (data, packet?)

(Must be overridden) Transmits the data, returns the number of bytes transmitted. See the individual child classes for the usage of the input parameters.

3. Socket.TCP

3.1 Properties

string **host_address**
in the form: xxx.xxx.xxx.xxx (example 192.168.0.2)

u32 **host_port**

string **remote_address**
in the form: xxx.xxx.xxx.xxx (example 192.168.0.3)

u32 **remote_port**

refnum **tcp_refnum**

refnum **listener_refnum**

3.2 Methods

socket.tcp.**Connect** (remote.address, remote_port, [host_address], [host_port])

string **host_address** is the optional address of the host. Not used by *Connect* but saved in the class properties.

u16 **host_port** is the optional port to connect to on the host. It is recommended to leave this unconnected to allow the OS to choose the best local port to use. Otherwise, the system's TIME-WAIT state makes recently closed ports unavailable for a while. Allowing the OS to choose the port number allows for quick re-connect.

string **remote_address** the address to connect to on the remote target.

string **remote_port** is the port to connect to on the remote target.

socket.tcp.**Listen** (net_address, port, service name)

string **net_address** is the host address to listen on. If left blank, the listener will listen to all host addresses.

u16 **listener_port** is the port number to listen on.

string **service_name** is optional. If specified, labview will register the service name and port in the NI service locator

socket.tcp.**Listen_wait** ()

Waits for an accepted TCP network connection. Saves the *remote_address* and *remote_port* of the accepted connection when one arrives.

socket.tcp.**Listen_close** ()

Closes the listener.

socket.tcp.**Receive** (mode, bufsize, [clear_timeout_error])

Returns bytes received or waits for the timeout period in ms and sends a timeout error.

enum **mode**

Standard: Waits until all bytes in *bufsize* arrive or until timeout. If timeout occurs, returns the bytes read so far and reports timeout error.

Buffered: Waits until all bytes in *bufsize* arrive or until timeout. If timeout occurs, returns no bytes and reports timeout error

CRLF: Waits until all bytes in *bufsize* arrive or until a carriage return (CR) followed by a linefeed (LF) or until timeout. Returns all bytes including the CR and LF. If timeout occurs before CRLF, returns no bytes and a timeout error.

Immediate: Waits until any bytes in *bufsize* arrive. Only reports a timeout errror if no bytes arrive.

I32 **bufsize** is the amount of bytes the receiver will wait for as determined by *mode*. If *bufsize* = 0, the incoming bytes are assumed to be a TCP packet and the first two bytes will be used to determine the size of the following packet to be received.

bool **clear_timeout_error** optional input to clear the timeout error before exiting the method. This is used when looping receive to check for loop exit conditions.

socket.tcp.**Transmit** (data, packet?)

Transmits the data, returns the number of bytes transmitted

string **data**

bool **packet?**
Defaults to false. If true, will prepend the data with two bytes indicating the size of the data to follow (not including the size bytes)

socket.tcp.**Enquire** ()

Transmits unprintable ascii 05: "Enquiry". If used, be sure receiver is designed to ignore ignore this byte. Enquiry is used to be sure the receiver was not unexpectedly shut down without properly closing the connection. note: Enquire is not available in the vi palette but can be found in the user.lib folder. This is because it requires advanced understanding for proper use.

4. Socket.UDP

4.1 Properties

string **host_address**
in the form: xxx.xxx.xxx (example 192.168.0.2)

u32 **host_port**

string **remote_address**
in the form: xxx.xxx.xxx (example 192.168.0.3)

u32 **remote_port**

refnum **UDP_refnum**

enum **transfer_mode**

point_to_point
multicast
broadcast

u16 **time-to-live**

Constrains the number of network router "hops" over which the multicast or broadcast will travel. The TTL field inside a network packet is sometime called the "hop limit".

4.2 Methods

socket.udp.**Connect** (remote.address, remote_port, [host_address], [host_port])

string **host_address** is the optional address of the host. Usage varies depending on the *transfer_mode*:
point-to-point: host address becomes both the net address and the service name of the UDP connection. If left blank, the connection will use all Ethernet cards on the machine.
multicast: host address will be used as the net address for the connection.
broadcast: host address is not used but is stored in the properties for later use.

u16 **host_port** is the optional port to connect to on the host. Usage varies depending on the transfer_mode*:*

point-to-point: host_port is the connection port number.
multicast: if host_port is 0, then the connection is write-only (server),, if host_port is not 0, then the connection is read-only (client) and host_port is the port that the connection will listen on.
broadcast: if host port is 0, then the connection will be a server. If not 0, then the connection will be a client and host_port is the port that the client will listen on.

string **remote_address** the address to connect to the remote target. Usage varies depending on the transfer_mode*:*

point-to-point: not used by connection but stored in the properties for later use.
multicast: if *host_port* is 0 then the connection is write only (server) and remote_address is not used by the connect method but stored in the properties for later use. If host_port is not 0, then the connection is read only (client) and the remote port is the port that the client will listen on.

string **remote_port** is the port to connect to on the remote target. Not used by the *Connect* method but saved in the properties for future use.

socket.udp.**Receive** ([mode], [bufsize], [clear_timeout_error])

enum **mode** ignored for UDP.

I32 **bufsize** is the maximum number of bytes to read. Default is 548 and if you wire any value other than 548 to *bufsize*, windows mya return an error because the function cannot read fewer bytes than are in a packet

bool **clear_timeout_error** optional input to clear the timeout error before exiting the method. This is used when looping receive to check for loop exit conditions.

socket.tcp.**Transmit** (data, [packet?])

Transmits the data, returns the number of bytes transmitted

string **data**

bool **packet?:** Ignored for UDP.

5. Error Codes

20xxx	Comms Library Errors	
20000	Override Error. Abstract class called.	The parent method of the class was called. This parent method is required to be overridden. A valid child class must be wired into the class input terminal.
20001	Comms Class Unused Comms Type: %s	The communications class can handle only the following communication types: TCP, UDP, UDP-Multicast, UDP-Broadcast, USB, GPIB, BitBucket, TCPPacket.
20002	Cannot Create Listener for %s Connection Type	Only TCP listeners can be created.

6. Availablility

This software is freely available for download at https://github.com/usnistgov

7. License

8. Disclaimer