

## NAME

**netsck** – network utility tool

## DESCRIPTION

**netsck** is a network utility tool which enables to prototype or test network things. It provides a shell inside which runs a javascript engine. This manual will present the *netsck\_Javascript\_API* inside the shell.

Note that, shell supports multiline input with trailing escape '\ ' character.

## ENGINE

Uses *QJSEngine* class to evaluate javascript codes so anything which QJSEngine supports available to the user.

## METHODS

**help( topic : string = base-api )**

Opens the man page according to the topic. If topic isn't given then opens this man page.

**run( file\_path : string ) -> int**

Executes the lines inside the file. Returns non-zero if any error occurred, otherwise 0.

**stdout( obj : any )**

Prints the 'obj' to stdout without newline.

**dump( object : any )**

Prints the content of any object to the stdout.

**sleep( duration : int )**

Sleeps current thread for specified duration. Unit is milliseconds.

**wait\_key( timeout : int )**

Waits for user to input a key and returns the value. Key value is the value returned from **std::getchar()**. If timeout expires function returns -1. Unit of timeout is milliseconds. Note that if timeout is -1 it works like there is no timeout.

**read\_line() -> string**

Reads line from stdin and returns as string

**array( data : QByteArray ) -> Array**

Converts QByteArray to javascript array.

**flat( data : Array ) -> QByteArray**

Converts a javascript array to QByteArray. Array should contain either number or characters. Numbers should be between 0-255.

**beint16( num : short ) -> Array**

Converts a short to 2 bytes representation over javascript array. If host system is little endian, bytes are reversed.

**beint32( num : int ) -> Array**

Converts an integer to 4 bytes representation over javascript array. If host system is little endian, bytes are reversed.

**beint64( num : double ) -> Array**

Converts an double to quint64 and to 8 bytes representation over javascript array. If host system is little endian, bytes are reversed.

Note that, double represents maximum 53 bits resolution for integers, so if you have bigger number they probably be truncated.

**befloat( num : float ) -> Array**

Converts a float to 4 bytes representation over javascript array. If host system is little endian, bytes are reversed.

**bedouble( num : double ) -> Array**

Converts a double to 4 bytes representation over javascript array. If host system is little endian, bytes are reversed.

**leint16( num : short ) -> Array**

Converts a short to 2 bytes representation over javascript array. If host system is big endian, bytes are reversed.

**leint32( num : int ) -> Array**

Converts an integer to 4 bytes representation over javascript array. If host system is big endian, bytes are reversed.

**leint64( num : double ) -> Array**

Converts an double to quint64 and to 8 bytes representation over javascript array. If host system is big endian, bytes are reversed.

Note that, double represents maximum 53 bits resolution for integers, so if you have bigger number they probably be truncated.

**lefloat( num : float ) -> Array**

Converts a float to 4 bytes representation over javascript array. If host system is big endian, bytes are reversed.

**ledouble( num : double ) -> Array**

Converts a double to 4 bytes representation over javascript array. If host system is big endian, bytes are reversed.

## CLASSES

Detailed class documentations can be found through **help()** with their class names.

For example, `help( "udp_socket" )`.

- PascalCase naming means the class is **singleton**.
- snake\_case naming means the class is **instantiable**.

**socket**

Base class which provides an abstract base for socket classes.

**udp\_socket**

Socket class which enables to send or receive udp datagrams.

**Hex**

Singleton hexadecimal utility class which prints QByteArray as hexadecimal or creates a QByteArray from hexadecimal string.

**EXAMPLE**

```
// Create a 'send.js' and write some js code in it to make it worked
run( "send.js" );

var an_object = { \
    user : "Ozan", \
    repo : "netsck" \
};
dump( an_object )
```

**SEE ALSO**

**socket** (7) , **udp\_socket** (7) , **Hex** (7)

**SEE ALSO (JS Shell)**

**help("socket")** , **help("udp\_socket")** , **help("Hex")**

## NAME

**socket** : abstract class

## DESCRIPTION

**socket** is an **abstract class which udp\_socket** inherits. Common socket methods are contained in this class. It is binding of **QAbstractSocket** class. It is not instantiable.

## METHODS

**stdout\_enabled() -> bool**

Returns a value which indicates whether info messages are enabled.

**enable\_stdout( value : bool = true )**

Enables/Disables info messages according to the 'value' parameter.  
Default value is true.

**addr() -> string**

Returns host address of local socket. It is equivalent to **QAbstractSocket::localAddress**.

**port() -> int**

Returns the host port numberr of the local socket. It is equivalent to **QAbstractSocket::localPort**.

**close()** Closes the socket. It is equivalent to **QAbstractSocket::close**

**bind( addr : string , port : int = 0 , mode : enum ) -> bool**

Binds sockets according to the parameters. It is equivalent to **AbstractSocket::bind**.  
If port is '0' so the socket selects an arbitrary empty port.  
Returns true if an operation is successful, otherwise false.

**flush( timeout : int = -1 ) -> bool**

Flushes write buffer. It is equivalent to **QAbstractSocket::waitForBytesWritten**. Returns true if bytes have been written, otherwise false

**wait( duration : int = -1 ) -> bool**

Waits for the datagrams by duration. Unit of duration is milliseconds. It is equivalent to **QAbstractSocket::waitForReadyRead**.  
If duration is '-1' so it waits until some datagram is received.  
Returns true if new data has arrived, otherwise false.

**wait\_a\_bit( duration : int = 0 ) -> bool**

If some datagrams waits on the OS buffer, it just fetches so waits\_a\_bit. It is equivalent to **QAbstractSocket::waitForReadyRead**.  
Returns true if new data has arrived, otherwise false.

**error() -> enum**

Returns the last error. It is equivalent to **QAbstractSocket::error**.

## SEE ALSO

**udp\_socket (7)**

**SEE ALSO (JS Shell)****help("udp\_socket")**

**NAME**

**udp\_socket** : class

**DESCRIPTION**

**udp\_socket** is a concrete class which inherits **socket**. It is binding of QUdpSocket class. All methods of **socket** class is usable.

**EXAMPLE**

```
var echo_srv = new udp_socket()
var client  = new udp_socket()

echo_srv.bind( "127.0.0.1" , 12000 )
// Send 'echo' to echo_srv

client.send( "echo" , "127.0.0.1" , 12000 )

echo_srv.wait()

var dgram = echo_srv.read_datagram()
dump( dgram )
echo_srv.send( dgram.data , dgram.sender_addr , dgram.sender_port )

client.wait()
dump( client.read_datagram() )
```

**OBJECTS**

**datagram** : object

```
{
  sender_addr : string ,
  sender_port : int ,
  dest_addr  : string ,
  dest_port  : int ,
  data       : QByteArray ,
  data_utf8  : string ,
  hop_limit  : int ,
  iface_idx  : int
}
```

It is returned from **read\_datagram()** method.

**METHODS**

**has\_datagram()** -> bool

Returns true if has pending datagram, otherwise false. It is equivalent to **QUdpSocket::hasPendingDatagrams()**.

**read\_datagram()** -> datagram

Returns the pending datagram. If there is not datagram returns an 'undefined'. It is equivalent to **QUdpSocket::receiveDatagram**.

**clear()** Discards all pending datagrams.

**send( data : QByteArray , addr : string , port : int )** -> qint64

Sends 'data' to 'addr:port' as udp packet. Returns how many bytes have been written. It is equivalent to **QUdpSocket::writeDatagram**

**SIGNALS****datagram()**

Emitted when a new datagram has come. It is equivalent to **QUdpSocket::readyRead**.

**NAME**

**Hex** : singleton class

**DESCRIPTION**

**Hex** is a **singleton** class. Prints **QByteArray** as hexadecimal in table format. Also constructs a **QByteArray** from hexadecimal string.

**EXAMPLE**

```
Hex.print( Hex.from( "ab 01 23 11 14 78 64 77 34 24 12 09 08" ) )
Hex.print( Hex.from( "ab012311147864773424120908" ) )
Hex.print( "This is a test string." )
```

**METHODS**

**print( data : QByteArray )**

Prints the data as hexadecimal in table format.

**from( hex\_data : QByteArray ) -> QByteArray**

Constructs a **QByteArray** from hex string. It is equivalent to **QByteArray::fromHex**.



**NAME**

**Key** : singleton class

**DESCRIPTION**

**Key** is an **singleton** class which provides readable key names. It is not instantiable.

**EXAMPLE**

```
var c = 0;

while ( ( c = wait_key( 33 ) ) != Key.ESC )
{
    if ( c == Key.Space )
        print( "Space is pressed." );
}
```

**CONSTANTS**

**TAB**

**RETURN**

**ESC**

**Space**

**Exclam**

**D0**

**D1**

**D2**

**D3**

**D4**

**D5**

**D6**

**D7**

**D8**

**D9**

**Colon**

**SemiColon**

**Less**

**Equal**

**Greater**

**Question**

**At**

**A**

**B**

**C**

**D**

**E**  
**F**  
**G**  
**H**  
**J**  
**K**  
**L**  
**M**  
**N**  
**O**  
**P**  
**Q**  
**R**  
**S**  
**T**  
**U**  
**V**  
**W**  
**X**  
**Y**  
**Z**  
**Underscore**  
**a**  
**b**  
**c**  
**d**  
**e**  
**f**  
**g**  
**h**  
**j**  
**k**  
**l**  
**m**  
**n**  
**o**  
**p**  
**q**

**r****s****t****y****v****w****x****y****z****Tilda****Backspace**