# 1 Introduction

Plugin Operations.

Please note
When sending variables to the plugin, they use the standard Neobook convention that encloses them in square brackets [ ].
When the plugin needs to set a variable in Neobook, the variable name must NOT be enclosed in square brackets.

DEMO Notes
The DEMO version will display a message box indicating it is a demo version when the plugin is unloaded.

New Command added 20th September 2007
gtCommSetHandshake - Enable or disable hardware handshaking.

Updated command 22nd May 2008
gtCommSendASCII - Now more than one ASCII code can be sent at one time. Simply enter more codes separated by a comma.

2011-07-22
Removed;
gtCommSetHandshake
Added;
gtCommCTSflow
gtCommDSRflow
gtCommDTRflow
gtCommRTSflow
gtCommFlowReset

Corrected some typo's in the PDF document.

For information please visit www.gjturner.net or send me an email at glendon@gjturner.net

***Plugin & document revision information.***

| | |
|---|---|
| Plugin version: | 1.8.0.2 |
| Plugin release date: | 21st July 2011 |
| PDF updated date: | 21st July 2011 |

# 2 RS232 Commands

## 2.1 Open-Close

### 2.1.1 gtCommOpen

**`gtCommOpen`** "port,baud,bits,parity,stop,txbuffer,rxbuffer"

Opens the communications 'port' at the specified 'baud' rate.
This command expects **seven** comma separated values.

Most simple serial devices operate at 9600 baud, with 8 data bits, no parity, and 1 stop bit. Please note

© 2011 ... G Turner

that your product may not use these settings, please refer to your devices configuration settings). The TX and RX buffers allocate the specified amount of RAM for both the transmit (TX) and receive (RX) buffers. If your need to send your device lots of data then increase the TX buffer. If your device sends a lot of data to your computer and you might think that it could possibly send more data than the buffer can hold before you receive it into NeoBook then increase the size of th RX buffer.

**PORT**
Port is the communications port number and must be in the range of 1 to 16.

**BAUD**
Baud is the data (baud) rate the port will communicate at, and must be one of the following, or 9600 will be set by default.
1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200.

**BITS**
Bits (number of bits per byte) can be one of the following (4,5,6,7,8)

**PARITY**
Parity must be one of the following:
0 = None
1 = Odd
2 = Even
3 = Mark
4 = Space
Entering an invalid parity value will set the default to NO Parity.

**STOP**
Stop (number of stop bits) can be one of the following:
0 = 1 stop bit
1 = 1.5 stop bits
2 = 2 stop bits

**TXBUFFER**
txbuffer is the number of bytes to allocate for the transmit buffer. Typical values are 1024, 2048, 4096.

**RXBUFFER**
rxbuffer is the number of bytes the allocate for the receive buffer. Typical values are 1024, 2048, 4096 etc. If you are not checking the serial port often but are expecting a reasonable amount of data then increase this value. (Max 32768).

**Notes**
When a port is opened, no values are set automatically.

**Returns**
Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Open comm port 1 at 9600 baud*
```
gtCommOpen "1,9600,8,0,0,0,1024,1024"
```

**Example #2**
*.Open comm port 1 at 9600 baud*
```
SetVar "[cport]" "1"
SetVar "[cbaud]" "9600"
gtCommOpen "[cport],[cbaud],8,0,0,0,1024,1024"
```

© 2011 ... G Turner

### 2.1.2 gtCommOpenStd

**gtCommOpenStd** "port,baud"

Opens the communications 'port' at the specified 'baud' rate.

Port must be in the range of 1 to 16.

Baud must be one of the following, or 9600 will be set by default.
1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200.

All other values are set to the following by default
Bits (number of bits per byte) = 8
Parity = None
Stop (number of stop bits) = 1 stop bit
Transmit and Receive buffers are set to 4096 bytes

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Open comm port 1 at 9600 baud*
**gtCommOpenStd** "1,9600"

**Example #2**
*.Open comm port 1 at 9600 baud*
**SetVar** "**[cport]**" "1"
**SetVar** "**[cbaud]**" "9600"
**gtCommOpen** "**[cport]**,**[cbaud]**"

### 2.1.3 gtCommClose

**gtCommClose** "port"

Closes an  open communications 'port'
Port must be in the range of 1 to 16
Port must already be open or an error will be returned in gtCommErr

This command expects one value.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Close open comm port 1*
**gtCommClose** "1"

**Example #2**
*.Close open comm port 1*
**SetVar** "**[cport]**" "1"
**gtCommClose** "**[cport]**"

© 2011 ... G Turner

## 2.2 Port Settings

### 2.2.1 gtCommSetBaud

**gtCommSetBaud** "port,baud"

Change the 'baud' rate of an already open communications 'port'
Port must already be open.
Port must be in the range of 1 to 16.
Baud must be one of the following, or 9600 will be set by default.
1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200.

This command expects two comma separated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Set baud rate of open comm port 2 to 19200 baud*
**gtCommSetBaud** "2,19200"

**Example #2**
*.Set baud rate of open comm port 2 to 19200 baud*
**SetVar** "**[cport]**" "2"
**SetVar** "**[cbaud]**" "19200"
**gtCommSetBaud** "**[cport]**,**[cbaud]**"

### 2.2.2 gtCommSetTxBuffer

**gtCommSetTxBuffer** "port,buffersize"

Change the transmit buffer size on the open communications 'port'
Port must already be open.
Port must be in the range of 1 to 16.
'buffersize' must be an interger in the range 0 to 8192. If a value greater than 8192 is specified, then 8192 will be set.

This command expects two comma separated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Set the transmit buffer size of open comm port 1 to 512 bytes*
**gtCommSetTxBuffer** "1,512"

**Example #2**
*.Set the transmit buffer size of open comm port 1 to 512 bytes*
**SetVar** "**[cport]**" "1"
**SetVar** "**[ctxbuff]**" "512"
**gtCommSetTxBuffer** "**[cport]**,**[ctxbuff]**"

### 2.2.3 gtCommSetRxBuffer

**gtCommSetRxBuffer** "port,buffersize"

Change the receive buffer size on the open communications 'port'

© 2011 ... G Turner

Port must already be open.
Port must be in the range of 1 to 16.
'buffersize' must be an integer in the range 0 to 32768

This command expects two comma separated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Set the receive buffer size of open comm port 1 to 8192 bytes*
```
gtCommSetRxBuffer "1,8192"
```

**Example #2**
*.Set the receive buffer size of open comm port 1 to 8192 bytes*
```
SetVar "[cport]" "1"
SetVar "[crxbuff]" "8192"
gtCommSetRxBuffer "[cport],[crxbuff]"
```

## 2.3 Transmit Data

### 2.3.1 gtCommSend

```
gtCommSend "port,data"
```

Send 'data' to the open communications 'port'
Port must already be open.
Port must be in the range of 1 to 16.

'data' can be any string value.

The data is sent as-is (no CR/LF is added)

This command expects two comma separated values.

Returns 0 or an error code in gtCommError for success or failure.

**Example #1**
*.Send (Transmit) the string 'Hello World' out open comm port 1*
```
gtCommSend "1,Hello World"
```

**Example #2**
*.Send (Transmit) the string 'Hello World' out open comm port 1*
```
SetVar "[cport]" "1"
SetVar "[mystring]" "Hello World"
gtCommSend "[cport],[mystring]"
```

### 2.3.2 gtCommSendASCII

```
gtCommSendASCII "port,ascii code,additional ascii codes"
```

Send one ore more 'ascii codes' to the open communications 'port'
Port must already be open.
Port must be in the range of 1 to 16.

© 2011 ... G Turner

'ascii code' can be any number between 1 and 255.

The data is sent as-is.

This command expects two or more comma separated values.

Returns 0 or an error code in gtCommError for success or failure.

**Example #1**
*.Send (Transmit) the Esc (Escape) character out open comm port 1*
```
gtCommSend "1,27"
```

**Example #2**
*.Send (Transmit) the Esc (Escape) character out open comm port 1*
```
SetVar "[cport]" "1"
SetVar "[mychar]" "27"
gtCommSend "[cport],[mychar]"
```

**Example #3**
*.Send (Transmit) the Esc (Escape) character followed by a CR (carriage return) and a LF (line feed) out open comm port 1*
```
SetVar "[cport]" "1"
gtCommSend "[cport],27,13,10"
```

**Example #4**
*.Send (Transmit) a lower case letter 'a' out open comm port 1*
```
SetVar "[cport]" "1"
gtCommSend "[cport],65"
```

### 2.3.3 gtCommSendCRLF

```
gtCommSendCRLF "port,data"
```

Send 'data' to the open communications 'port'
Port must already be open.
Port must be in the range of 1 to 16.
'data' can be any string value.
The data is sent with a CR/LF added to the end.

This command expects two comma separated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Send (Transmit) the string 'Hello World' followed by a CR and LF out open comm port 1*
```
gtCommSendCRLF "1,Hello World"
```

**Example #2**
*.Send (Transmit) the string 'Hello World' followed by a CR and LF out open comm port 1*
```
SetVar "[cport]" "1"
SetVar "[mystring]" "Hello World"
gtCommSendCRLF "[cport],[mystring]"
```

© 2011 ... G Turner

## 2.4 Receive Data

### 2.4.1 gtCommRxByte

**gtCommRxByte** "port,received byte variable,bytes left variable"

Receive 1 byte of data from serial port. If no data exists in the receive buffer then no data will be returned.

Port must already be open.
Port must be in the range of 1 to 16.

'received byte variable' is the variable name in Neobook to store the received byte into. NOTE: do NOT use [MyVar]  just use MyVar.

'bytes left variable' is the variable name in Neobook to store the number of bytes left in the rx buffer.

**Example 1:**
**gtCommRxByte** "1,rxdata,rxbuffsize"

**Example 2:**
**SetVar** "**[cport]**" "1"
**gtCommRxByte** "**[cport]**,rxdata,rxbuffsize"

**BAD EXAMPLE (Wont Work!)**
**gtCommRxByte** "1,**[rxdata]**,**[rxbuffsize]**"

### 2.4.2 gtCommRxS

**gtCommRxS** "port,returnvarname,no of bytes to receive"

Receive a string of bytes from serial port.

Port must already be open.
Port must be in the range of 1 to 16.

'returnvarname' is the variable name in Neobook to store the

'no of bytes to receive' is the variable name in Neobook to store the number of bytes left in the rx buffer.

**Example #1**
**gtCommRxS** "1,rxdata,5"
*.Get 5 bytes for data (if available) from the rx buffer and return them in a Neobook variable called [rxdata]*

**Example #2**
**SetVar** "**[cport]**" "1"
**SetVar** "**[rxsize]**" "5"
**gtCommRxS** "**[cport]**,rxdata,**[rxsize]**"
*.Get 5 bytes for data (if available) from the rx buffer and return them in a Neobook variable called [rxdata]*

© 2011 ... G Turner

**BAD EXAMPLE (Wont Work!)**
**gtCommRxS** "**[cport]**,**[rxdata]**,**[rxsize]**"
*.This example wont work because the return variable [rxdata] has square braces. Remove the braces for the plugin to work.*

### 2.4.3 gtCommRx

**gtCommRx** "port,filename var,received bytes count var"

Receive all bytes in the receive buffer of an open communications port and place them directly into a disk file.

**Warning!!**
If the file exists, it will be overwritten.

Port must already be open.
Port must be in the range of 1 to 16.

'filename var' is the variable name in Neobook to store the received byte into. NOTE: do NOT use [MyVar] just use MyVar.

'bytes count var' is the variable name in Neobook to store the number of bytes actually saved to the file.

**Example #1**
**gtCommRx** "1,myfile.txt,rxsize"

**Example #2**
**SetVar** "**[cport]**" "1"
**SetVar** "**[filename]**" "myfile.txt"
**gtCommRx** "**[cport]**,**[filename]**,rxsize"

**BAD EXAMPLE (Wont Work!)**
**gtCommRx** "**[cport]**,**[filename]**,**[rxsize]**"

### 2.4.4 gtCommRxF

**gtCommRxF** "port,filename var,timeout value"

Receive all bytes in the receive buffer of an open communications port and place them directly into a file until no more bytes arrive for the timeout value.

If the file exists, received data will be appended to it.

Port must already be open.
Port must be in the range of 1 to 16.

'filename var' is the variable name in neobook to store the

'timeout value' is number of milliseconds (1/1000 second) to wait for the timeout.

**Notes**
This plugin function runs in a loop until the receive buffer has been written to the nominated file and the receive buffer remains empty for the nominated timeout value.

© 2011 ... G Turner

Use this to write a large stream of incoming data to a file. When the incoming stream stops for the timeout value, the function will end.

**WARNING**
This function will not return until it times out!
During this time, your Neobook program will seem to have locked up.

**Example #1**
```
gtCommRxF "1,myfile.txt,1000"
```

**Example #2**
```
SetVar "[cport]" "1"
SetVar "[filename]" "myfile.txt"
gtCommRxF "[cport],[filename],1000"
```

**BAD EXAMPLE (Wont Work!)**
```
gtCommRxF "[cport],[filename],[rxsize],1000"
```

### 2.4.5 gtCommRxLine

```
gtCommRxLine "port,returnedlinevar,timeoutvar"
```

Receive a complete line of bytes from the serial port (CR/LF terminated) or until a timeout occurs (which ever occurs first).

Port must already be open.
Port must be in the range of 1 to 16.

'returnedlinevar' is the variable name in neobook to store the
received byte into. NOTE: do NOT use [MyVar]  just use MyVar.

'timeoutvar' is a value in milliseconds * 10 for the timeout. This is how long the plugin will wait for a CR/LF before returning with the received data. 1second = 1000milliseconds, so for a 2.5second timeout use a value of 250. The allowable range for the timeout is 1 to 1000 (10 milli-Seconds to 10 Seconds)

**Returns**
gtCommEof = 1 if the plugin exits because of a timeout.
gtCommErr = 0 if no errors occurred, or other number if error.

**Example #1**
*Receive a CR terminated line of data from serial port 1. Return any received data in the neobook variable [myvar]. If no data OR no CR has been found in less than 1 second (100 * 10mS) then return anyway.*
```
gtCommRxLine "1,myvar,100"
```

**Example #2**
*Receive a CR terminated line of data from serial port 1. Return any received data in the neobook variable [myvar]. If no data OR no CR has been found in less than 1 second (100 * 10mS) then return anyway.*
```
SetVar "[cport]" "1"
SetVar "[timeout]" "100"
```

© 2011 ... G Turner

```
gtCommRxLine "[cport],myvar,[timeout]"
```

**BAD EXAMPLE (Wont Work!)**
*This example wont work because the second option [myvar] should have NO square brackets.*

```
gtCommRxLine "[cport],[myvar],[timeout]"
```

### 2.4.6 gtCommRxFlush

```
gtCommRxFlush "port"
```

Flush any bytes waiting in the receive buffer.

Port must already be open.
Port must be in the range of 1 to 16.

Returns
gtCommErr = 0 if no errors occurred, or other number if error.

**Example #1**

```
gtCommRxFlush "1"
```

**Example #2**

```
SetVar "[cport]" "1"
gtCommRxFlush "[cport]"
```

## 2.5 Information

### 2.5.1 gtCommRxCount

```
gtCommRxCount "port,count"
```

Find out how many bytes are in the receive buffer of an open communications port.
Port must already be open.
Port must be in the range of 1 to 16.

This command expects two comma seperated values.

**Returns**
gtCommErr = 0 if no errors occurred, or other number if error.

**Example #1**
*.Get the rx buffer count and store into variable 'rxbuffercount'*

```
gtCommRxCount "1,rxbuffercount"
```

**Example #2**
*.Get the rx buffer count and store into variable 'rxbuffercount' then display this in an Alert Box.*

```
SetVar "[cport]" "1"
gtCommRxCount "[cport],rxbuffercount"
AlertBox "Information" "There are [rxbuffercount] bytes in the RX buffer"
```

© 2011 ... G Turner

### 2.5.2 gtCommGetBaud

**gtCommGetBaud** "port,baud var"

Find out what the baud rate is of an open communications port.
Port must already be open.
Port must be in the range of 1 to 16.

This command expects two comma separated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Get the current baud rate setting for comm 1 and store this into variable 'mybaud'*
**gtCommGetBaud** "1,mybaud"

**Example #2**
*.Get the current baud rate setting for comm 1 and store this into variable 'mybaud'*

```
SetVar "[cport]" "1"
gtCommGetBaud "[cport],mybaud"
AlertBox "Information" "Comm[cport] is current open at [mybaud] baud"
```

## 2.6 Handshaking

### 2.6.1 gtCommXonXoff

**gtCommXonXoff** "port,setting"

setting 1 = on, 0 = off(default)

Set (Enable) or Clear (Disable) Xon/XOff (Software) flow control for an open com port.

Port must already be open.
Port must be in the range of 1 to 16.

This command expects two comma seperated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Set comm1 to enable software handshaking.*
**gtCommXoXoff** "1,1"

**Example #2**
*.Set comm1 to enable software handshaking.*

```
SetVar "[cport]" "1"
SetVar "[cflow]" "1"
gtCommGetBaud "[cport],[cflow]"
```

© 2011 ... G Turner

### 2.6.2 gtCommCTSflow

**gtCommSetCTSflow** "port,setting"

setting 1 = enable, 0 = disable(default)

Enable CTS output flow control (Input signal). When CTSFLOW is enabled, it causes the DTE (computer) to stop sending data whenever the CTS signal is set to logic low by the DCE (modem). Transmission continues when the DCE (modem) sets the CTS signal back to logic high. The CTS signal is usually used in response to an RTS signal.

Port must already be open.
Port must be in the range of 1 to 16.

This command expects two comma separated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Set comm1 to enable CTS hardware flowcontrol*
```
gtCommSetCTSflow "1,1"
```

**Example #2**
*.Set comm1 to enable CTS hardware flowcontrol*
```
SetVar "[cport]" "1"
SetVar "[ctshwflow]" "1"
gtCommSetCTSflow "[cport],[ctshwflow]"
```

### 2.6.3 gtCommDSRflow

**gtCommSetDSRflow** "port,setting"

setting 1 = enable, 0 = disable(default)

Enable DSR output flow control (Input signal). When DSRFLOW is enabled, it causes the DTE (computer) to stop sending data whenever the DSR signal is set to logic low by the DCE (modem). Transmission is enabled when the DSR signal returns to logic high. The DSR signal is often used in conjunction with CTS in response to a RTS signal.

Port must already be open.
Port must be in the range of 1 to 16.

This command expects two comma separated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Set comm1 to enable DSR flow control*
```
gtCommSetDSRflow "1,1"
```

**Example #2**
*.Set comm1 to enable DSR flow control*
```
SetVar "[cport]" "1"
```

© 2011 ... G Turner

```
SetVar "[dsrhwflow]" "1"
gtCommSetDSRflow "[cport],[dsrhwflow]"
```

### 2.6.4 gtCommDTRflow

**gtCommSetDTRflow** "port,setting"

setting 1 = enable, 0 = disable(default)

Enable DTR handshaking flow control (Output signal). When DTRFLOW is enabled, it signals that the DCE (modem) should prepare to connect to the communications channel. DTR is usually used for modem on-hook/off-hook control, but can also be used in conjunction with DSR for handshaking.

Port must already be open.
Port must be in the range of 1 to 16.

This command expects two comma separated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
*.Set comm1 to enable DTR flow control*

```
gtCommSetDSRflow "1,1"
```

**Example #2**
*.Set comm1 to enable DTR flow control*

```
SetVar "[cport]" "1"
SetVar "[dtrhwflow]" "1"
gtCommSetDTRflow "[cport],[dtrhwflow]"
```

### 2.6.5 gtCommRTSflow

**gtCommSetRTSflow** "port,setting"

Ready To Send (Output signal). 0 = Disable, 1 = Enable, 2 = Handshake, 3 = Toggle. Toggle is used for half-duplex (2-wire) operations to "reverse" the line. While the DTE (computer) is busy sending data, it raises the RTS signal and the DCE (modem) blocks its data receive channel. When RTS signal reverts to logic low, the DCE (modem) reverts to transmit mode and the DTE (computer) switches to receive mode.

Handshake mode causes the DTE (computer) to check the receive buffer (RXQUE) after each character is placed into the buffer. When the buffer is 5/6th full, the RTS signal is dropped. When the receive buffer drops to below 1/6th full, RTS is raised again.

Port must already be open.
Port must be in the range of 1 to 16.

This command expects two comma separated values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**

*.Set comm1 to enable RTS flow control*
```
gtCommSetDSRflow "1,1"
```

**Example #2**
*.Set comm1 to enable RTS flow control*
```
SetVar "[cport]" "1"
SetVar "[rtshwflow]" "1"
gtCommSetRTSflow "[cport],[rtshwflow]"
```

### 2.6.6 gtCommFlowReset

```
gtCommFlowReset "port"
```

Switches off all flow control to the serial port as specified by the port number.

Port must already be open.
Port must be in the range of 1 to 16.

This command expects one single values.

Returns 0 or an error code in gtCommErr for success or failure.

**Example #1**
```
gtCommFlowReset "1"
```

**Example #2**
```
SetVar "[cport]" "1"
gtCommFlowReset "[cport]"
```

# 3 TCP Commands

## 3.1 gtTCPopen

```
gtTCPopen "host,port,timeout"
```

**Important**
Unlike the RS232 port, a TCP connection requires a timeout value that is assigned when the port is opened. All plugin functions (actions) that work with a TCP connection will NOT return until this timeout expires. So: Neobook will not continue until the operation completes, or the timeout expires.

**Limitations**
You can only open one port at a time.

**Description**
Opens a TCP connection to the host at the specified port with a timeout value in milliseconds (1000 = 1 second)

Host is standard IP address notation such as 192.168.0.1 and must NOT be padded with leading zeros (192.168.000.001 will NOT work).

Port is a valid TCP port in the range 1 to 65535

This command expects three comma separated values.

Returns 0 or an error code in gtTcpErr for success or failure.

**Notes**
There are several devices available to connect serial devices to an ethernet network. You can use the TCP commands to communicate easily with one of these serial-to-ethernet converters. I personally have used and recommend the Lantronix Xport devices ( www.lantronix.com ) with great success in several commercial products.

**Example #1**
*.Attempt connection to address 192.168.0.1 at port 10001 with a 5 second timeout.*
```
gtTcpOpen "192.168.0.1,10001,5000"
```

**Example #2**
*.Attempt connection to address 192.168.0.1 at port 10001 with a 5 second timeout.*
```
SetVar "[ipaddr]" "192.168.0.1"
SetVar "[ipport]" "10001"
SetVar "[timeout]" "5000"
gtTcpOpen "[ipaddr],[ipport],[timeout]"
```

## 3.2 gtTCPclose

```
gtTCPclose ""
```

Closes the open TCP connection.

This command expects nothing.

Returns 0 or an error code in gtTcpErr for success or failure.

**Example #1**
*.Attempt connection to address 192.168.0.1 at port 10001 with a 5 second timeout.*
```
gtTcpClose ""
```

## 3.3 gtTCPsend

```
gtTCPsend "data"
```

Neobook will not continue until the operation completes, or the timeout expires.

Send the 'data' string to the open TCP connection.

This command expects one value.

Returns 0 or an error code in gtTcpErr for success or failure.

**Example #1**
*.Send Hello There out the open TCP connection*
```
gtTcpSend "Hello There"
```

© 2011 ... G Turner

**Example #2**
*.Send Hello There out the open TCP connection*

```
SetVar "[ipdata]" "Hello There"
gtTcpSend "[ipdata]"
```

## 3.4 gtTCPrxbyte

gtTCPrxbyte "var"

Receive 1 byte of data from the open TCP connection.

Port must already be open.

'var' is the variable name in neobook to store the received byte into.
NOTE: do NOT use [MyVar] just use MyVar.

The plugin returns information in the following variables;
gtTcpErr
gtTcpEof

The gtTcpEof variable is used in a loop when reading one byte at a time and the loop should be terminated when gtTcpEof is Non-zero.

**Example #1**

```
.Get one byte from the open TCP connection and place it in the NeoBook variable
rxdata
gtTCPrxbyte "rxdata"
```

**Example #2**

```
.Get one byte from the open TCP connection and place it in the NeoBook variable
rxdatabyte
.Then append it to the end of the string called rxdata. Terminate when the plugin
returns
. and end of file error condition.

While "[gtTcpEof]" "=" "0"
 gtTCPrxbyte "rxdatabyte"
 StrLen "[rxdata]" "[rxdatalen]"
 StrIns "[rxdatabyte]" "[rxdata]" "[rxdatalen]+1" "[rxdata]"
EndWhile
```

## 3.5 gtTCPrxline

gtTCPrxline "var"

Receive 1 complete line of data (CR/LF terminated) from the open TCP connection.

Port must already be open.

'var' is the variable name in neobook to store the received byte into.
NOTE: do NOT use [MyVar] just use MyVar. (ie dont use the square braces)

The plugin returns information in the following variables;
gtTcpErr
gtTcpEof

The gtTcpEof variable is used in a loop when reading one byte at a time and the loop should be terminated when gtTcpEof is Non-zero.

**Example #1**

```
.Get one line from the open TCP connection and place it in the NeoBook variable rxdata
gtTCPrxbyte "rxdata"
```

## 3.6 gtTCPrxF

gtTCPrxF "var"

Receive 1 complete line of data (CR/LF terminated) from the open TCP connection.

Port must already be open.

'var' is the variable name in neobook to store the received byte into.
NOTE: do NOT use [MyVar]  just use MyVar. (ie dont use the square braces)

The plugin returns information in the following variables;
gtTcpErr
gtTcpEof

The gtTcpEof variable is used in a loop when reading one byte at a time and the loop should be terminated when gtTcpEof is Non-zero.

**Example #1**

```
.Get one line from the open TCP connection and place it in the NeoBook variable rxdata
gtTCPrxbyte "rxdata"
```

## 3.7 gtTCPrxFlush

Requires no parameters

Flushes any waiting bytes in the TCP receive buffer.

# 4 Notes

© 2011 ... G Turner