The “WXSock” WIL Extender for Windows Sockets

Dial-Up Networking Commands

The Dial-Up Networking (DUN) commands let your modem dial an Internet Service Provider (ISP) without having to respond to the “Connect To” dialog, which you normally see when you run an Internet program such as a web browser.

HTTP Commands

HTTP (short for “Hyper-Text Transfer Protocol”) is the protocol used to request files from a web server.

When a web browser needs to get a file from a web server, the client app takes the URL that the user specified and extracts the server name and file’s pathname. Then it connects to the server and requests the file. The server then sends back the file, along with several lines of header information which the browser uses to determine how to display the data (as an HTML page, a GIF graphic, etc).

You can act like a web browser with the HTTPRecvText function.

When a user fills out a form on a web page, it gets sent to the server in one of three ways, depending on the METHOD clause in the form’s <form> tag: GET, POST, and MAILTO.

In the GET method, the browser collects the data from the various fields into one long string (in a format called “urlencoded”) and pastes it onto the end of a URL specifying a “CGI” script located on the server computer. The browser then sends a “GET” request to the web server, as if it was just requesting another file to display. But in this case the server passes the query string on to the CGI program, which decodes the query string, processes the request, creates an HTML file on the fly, and sends the results back to the browser. (This is the page you see after you hit the “Submit” button that says something like “This form successfully completed…” or else shows you the search results you asked for.)

The GET method is used only for short forms, such as most search engines. Most servers truncate the query string to 255 characters.

The POST method is somewhat similar to a GET. The main thing is that there is no limit at all to the length of the query string. More and more CGI programmers are creating their forms with the POST method and ignoring GET altogether.

Either way, you can mimic a user submitting a web form by calling HTTPRecvQuery.

If you want a caller to your website to be able to fill out a form but your website can’t handle CGI scripting, you can create a “METHOD=MAILTO” form on your web page that simply creates an email message from the urlencoded query string & mails it to your mailbox. These messages will have a line in the header that says “Content-Type: application/x-www-form-urlencoded”. The sample ScanMail.wbt script shows how you can use the POP3 & URLDecode functions to check your mailbox and process these messages automatically.

POP3 Commands

Checking your mailbox for incoming mail is accomplished on the Internet via “Post-Office Protocol v3”, or POP3.

To check your mailbox you must first start a POP3 session with your mail server by calling P3Open. The mail server name is usually the same as you would use to send mail via the SMTP commands – i.e. “mail.nerds-r-us.com”.

With the POP3 session started, you can query the number of messages in your mailbox with P3Count; Look at the first n lines with P3Peek, download the whole message with P3RecvText, then delete the messages from the server with P3Delete. When you’re finished, call P3Close to close the POP3 session.

SMTP Commands

Simple Mail Transport Protocol (SMTP) is the protocol used to send outgoing mail through the Internet.

To send mail, you must specify a mail server. If you use an Internet Service Provider, the mail server name is usually the same as your ISP’s domain name with a prefix of “mail.”. (For example, nerds-r-us.com probably uses “mail.nerds-r-us.com”.) They probably told you this information when you first signed up with the ISP, since all mail programs require it.

Checking your mailbox for incoming mail is more complicated, and requires a different protocol called “POP3”. Hence our P3xxx commands.

Sockets Commands

Windows Sockets is the underlying system that lets you communicate over the Internet (or over any network that supports TCP/IP).

WXSock Command Reference

DUNConnect

Dials up an Internet host, without making you respond to the “Connect to” dialog.

Syntax:

```
nConn = DUNConnect (sHost)
```

Parameters:

(s) dialupname	The Internet host to dial up, such as returned from DUNItemize. (This is the descriptive name the user would see in the “Connect To” dialog, not the actual domain name.)

Returns:

(i)	A handle that identifies the connection, or @FALSE if error. If you’re already connected to this host, then this handle identifies the existing connection.

SGetLastErr():

@SOK	Dialed up the specified host successfully.
@SAlready	The dialup is already connected. The return value identifies the existing connection.
@SErrBusy	Line was busy.
@SErrNoAnswer	No answer.
@SErrVoice	A human answered.
@SErrNotFound	Unknown host name specified.
600..750	Remote Access Service error code (see Appx. B).

See Also:

DUNItemize, DUNDisconnect

DUNDisconnect

Hangs up a specified dial-up Internet connection, or all existing connections.

Syntax:

```
nRet = DUNDisconnect (nConn)
```

Parameters:

(i) connection	The dial-up connection handle to hang up on (as returned from DUNConnect), or 0 to hang up on all open connections.

Returns:

(i)	@TRUE if OK, else @FALSE if there was an error.

SGetLastErr():

@SOK	Hung up the host(s) successfully.
@SErrParam	Unknown connection handle specified.
600..750	Remote Access Service error code (see Appx. B).

See Also:
DUNConnect

DUNGetPWNT3

Gets the current dial-up networking password under Windows NT v3.x.

Syntax:
```
sPW = DUNGetPWNT3 (sHost)
```

Returns:

(s)	The dial-up host whose password to get.

This password is only used by WXSock when running under Windows NT v3.x, and is saved in encrypted form in the Registry. When running under Windows 95, WXSock uses the password that was entered in the Windows Control Panel.

SGetLastErr():

@SOK	Found the password successfully.

See Also:
DUNSetPWNT3

DUNItemize

Creates a list of all the valid Dial-Up Networking hosts. These are the ones the user normally chooses from when they connect to the Internet via the "Connect To" dialog.

Syntax:
```
sHosts = DUNItemize ()
```

Returns:

(s)	A tab-delimited list of dial-up hosts.

SGetLastErr():

@SOK	Created the list successfully.
600..750	Remote Access Service error code (see Appx. B).

See Also:
DUNConnect

DUNSetPWNT3

Changes the dial-up networking password under Windows NT v3.x.

Syntax:

```
sPW = DUNSetPWNT3 (sHost, sNewPW)
```

Returns:

(s)	The dial-up host whose password to change.
(s)	The new password.

This password is only used by WXSock when running under Windows NT v3.x, and is saved in encrypted form in the Registry. When running under Windows 95, WXSock uses the password that was entered in the Windows Control Panel.

SGetLastErr():

@SOK	Changed the password successfully.

See Also:
DUNGetPWNT3

HTTPGetAnchor

Extracts the anchor name from an HTTP URL.

Syntax:
sAnchor = HTTPGetAnchor (sURL, sDefault)

Parameters:

(s) URL	The URL to parse. This can be a fully-qualified URL or relative.
(s) default	The default anchor name to use.

Returns:

(s)	The full anchor name if found, else the default.

When you click on a link in a web page that looks like “http://www.server.com/anotherpg#theanchor”, it will take you directly to the middle of another page. That position is called an “anchor.” This corresponds to an <ISINDEX> tag somewhere in the target HTML page.

This function would extract “theanchor” from the URL above.

See Also:
HTTPGetDir, HTTPGetFile, HTTPGetPath, HTTPGetServer, HTTPRecvQuery, HTTPRecvText, URLGetScheme

HTTPGetDir

Extracts the full directory path from an HTTP URL.

Syntax:
sDir = HTTPGetDir (sURL, sDefault)

Parameters:

(s) URL	The URL to parse. This can be a fully-qualified URL or relative.

(s) default	The default directory path to use.

Returns:

(s)	The full directory path if found, else the default.

See Also:

HTTPGetAnchor, HTTPGetFile, HTTPGetPath, HTTPGetServer, HTTPRecvQuery, HTTPRecvText, URLGetScheme

HTTPGetFile

Extracts the filename from an HTTP URL.

Syntax:

sFile = HTTPGetFile (sURL, sDefault)

Parameters:

(s) URL	The URL to parse. This can be a fully-qualified URL or relative.
(s) default	The default filename to use.

Returns:

(s)	The filename if found, else the default.

See Also:

HTTPGetAnchor, HTTPGetDir, HTTPGetPath, HTTPGetServer, HTTPRecvQuery, HTTPRecvText, URLGetScheme

HTTPGetPath

Extracts the file's full pathname from an HTTP URL.

Syntax:

sPath = HTTPGetPath (sURL, sDefault)

Parameters:

(s) URL	The URL to parse. This can be a fully-qualified URL or relative.
(s) default	The default directory path for the file.

Returns:

(s)	The full pathname of the file.

See Also:

HTTPGetAnchor, HTTPGetDir, HTTPGetFile, HTTPGetServer, HTTPRecvQuery, HTTPRecvText, URLGetScheme

HTTPGetQuery

Extracts a CGI query string from an HTTP URL.

Syntax:

sQuery = HTTPGetQuery (sURL, sDefault)

Parameters:

(s) URL	The URL to parse. This can be a fully-qualified URL or relative.
(s) default	The default query string to use.

Returns:

(s)	The full query string if found, else the default.

When you fill out a form in a web browser, it builds a long URL string consisting of a CGI script filename on the web server, followed by a "?" and all the data you filled out on the form in "urlencoded format". This function extracts just that data from the URL.

See Also:
HTTPGetAnchor, HTTPGetDir, HTTPGetFile, HTTPGetPath, HTTPGetServer, HTTPRecvQuery, HTTPRecvText, URLDecode, URLEncode, URLGetScheme

HTTPGetServer

Extracts the server's domain name from an HTTP URL.

Syntax:
sServer = HTTPGetServer (sURL, sDefault)

Parameters:

(s) URL	The URL to parse. This can be a fully-qualified URL or relative.
(s) default	The default domain name to use.

Returns:

(s)	The domain name if found, else the default.

See Also:
HTTPGetAnchor, HTTPGetDir, HTTPGetFile, HTTPGetPath, HTTPRecvQuery, HTTPRecvText, URLGetScheme

HTTPRecvQuery

Sends a GET or POST request to a CGI program on a web server and gets the response.

Parameters:

(s) server	The web server where the CGI script is located.
(s) path	The pathname of the CGI script on the server's computer.
(s) query	The query string for the CGI script.
(i) maxsize	Max chars of data to receive.
(i) flags	Specify @HMethodGet or @HMethodPost (the default) depending on which type of web form method to simulate. "Or" this together with @HHeader if you want to receive the HTTP header lines as well as the data, else @HNoHeader (the default).

Returns:

(s)	The CGI program’s output.

SGetLastErr():

@SOK	Recieved the response OK.
@SErrNoConn	Server closed the connection before sending anything back.

Other errors from SOpen, SConnect, SSendLine, or SRecvLine.

Each HTTP reply starts with several lines of header information and a blank line before the requested data.

A standard CGI query string consists of one or more “name=value” pairs, each separated by “&”. If you’re simulating a user entering data in a web form, each name=value pair corresponds to a field on the form. The “name” part is the same as the NAME= field in the HTML code that defines the field, and the “value” part is the data the user entered.

NOTE: Before building up your query string, call URLEncode on each “value” string, just in case there are spaces or punctuation marks in it. The CGI script will get very confused if it’s not urlencoded first.

Before using this function to simulate a user filling out a web form, check the actual form you want to mimic & notice the METHOD= clause in its <form> tag. It should read either “METHOD=GET”, “METHOD=POST”, or “METHOD=MAILTO”. GET & POST correspond to a flag value of @HMethodGet & @HMethodPost. If it’s a MAILTO form, use SMTPSendText to email the query as the body of the message.

See Also:

HTTPGetPath, HTTPGetServer, HTTPRecvText, URLEncode

HTTPRecvText

Downloads a text file from a web server.

Parameters:

(s) server	The web server to get the file from.
(s) path	The pathname of the file on the server's computer.
(i) maxsize	Max size to receive.
(i) hdr too?	@TRUE if you want to receive the HTTP header lines as well as the data, else @FALSE.

Returns:

(s)	The file you requested, optionally with the HTTP header lines at the beginning.

SGetLastErr():

@SOK	Recieved the file OK.
@SErrNoConn	Server closed the connection before sending anything back.

Other errors from SOpen, SConnect, SSendLine, or SRecvLine.

Each HTTP reply starts with several lines of header information and a blank line before the requested data.

A file that's received from a web server is usually an HTML web page, but it could be any file you get when you click on a link in a web browser.

See Also:
HTTPGetAnchor, HTTPGetDir, HTTPGetFile, HTTPGetPath, HTTPGetServer, HTTPRecvQuery, URLDecode, URLGetScheme

P3Close

Closes a POP3 session.

Syntax:
bOK = P3Close (hPOP)

Parameters:

(i) handle	The POP3 session handle you got from P3Open.

Returns:

(i)	@TRUE if it was closed OK, else @FALSE if an error.

SGetLastErr():

@SOK	Disconnected OK.

P3Count

Counts the number of messages in your mailbox.

Syntax:
nMsgs = P3Count (hPOP)

Parameters:

(i) handle	The POP3 session handle you got from P3Open.

Returns:

(i)	How many mail messages there are waiting for you, or 0 if there was an error. (You should check SGetLastErr() to make sure the return value is valid.)

SGetLastErr():

@SOK	Got the information OK.
@P3ErrReply	The mail server sent back an error message instead of the message count.

Other error codes from SSendLine, SRecvLine.

P3Delete

Deletes a message from your mailbox.

Syntax:
bOK = P3Delete (hPOP, nMsg)

Parameters:

(i) handle	The POP3 session handle you got from P3Open.

(i) message	Which message in your mailbox to delete. The message numbers start at 1.

Returns:

(number)	@TRUE if it deleted the message OK, else @FALSE if an error occurred.

SGetLastErr():

@SOK	Deleted it OK.

Other error codes from SSendLine, SRecvLine.

P3GetReply

Gets the description part of the last POP3 reply code.

Syntax:

sReplyText = P3GetReply ()

Returns:

(s)	The POP3 reply received after the last command we sent it.

A POP3 server responds to a request with a line starting with either +OK or -ERR, and usually including a description after it. This function just returns the description part. To find out if a P3xxx function call had an -ERR reply, check SLastErr() for @P3ErrReply.

See Also:

SGetLastErr

P3Open

Opens a session with a POP3 mail server.

Syntax:

bOK = P3Open (sMailServer, sUser, sPW)

Parameters:

(s) server	Your mail server. This is usually in the form of "mail.yourisp.com".
(s) user	Your user name.
(s) PW	Your password, if any. (This is the one you had to use when you logged on to the network.)

Returns:

(number)	@TRUE if it started the session OK, else @FALSE if an error or the password is wrong.

SGetLastErr():

@SOK	Connected OK.
@SErrParam	No POP3 server specified.
@P3ErrReply	POP3 server sent back an error reply to one of our login commands.

Other error codes from SOpen, SConnect, SSendLine, SRecvLine.

P3Peek

Reads the header & first few lines of a message in your mailbox.

Syntax:

sMsg = P3Peek (hPOP, nMsg)

Parameters:

(i) handle	The POP3 session handle you got from P3Open.
(i) message	Which message in your mailbox to peek at. The message numbers start at 1.
(i) lines	How many lines to retrieve.

Returns:

(s)	The truncated message with all the header lines plus as many lines of the body as you specified.

SGetLastErr():

@SOK	Connected OK.
@SErrParam	No mail server, from, or to address specified.

Other error codes from SSendLine, SRecvLine.

P3RecvText

Downloads a message from your mailbox.

Syntax:

sMsg = P3RecvText (hPOP, nMsg)

Parameters:

(i) handle	The POP3 session handle you got from P3Open.
(i) message	Which message in your mailbox to download.The message numbers start at 1.
(i) maxsize	Max size to receive. (This value must be less than 65535, which is the total allowed for all strings in your WIL script. A good value would be 32767.)

Returns:

(s)	The text of the message if OK, else NULL if error.

SGetLastErr():

@SOK	Received it OK.

Other error codes from SRecvLine.

SByteOrder16
SByteOrder32

Converts a 16- or 32-bit binary number from network byte order to the PC's byte order & vice versa.

Syntax:
```
   nData = SByteOrder16 (nData, nDirection)
   nData = SByteOrder32 (nData, nDirection)
```

Parameters:

(i) number	The binary number to translate.
(i) direction	@SNet2PC - translates a number received from the net to the PC's byte order. @SPC2Net - translates a number from the PC's byte order to the network's.

Returns:

(i)	The translated number.

You normally don't need to call these functions if you are just sending or receiving a value thru SSendNum8, SSendNum16, SSendNum32, SRecvNum8, SRecvNum16, or SRecvNum32. They all do the translation for you.

SClose

Closes a socket.

Syntax:
```
   nRet = SClose (hSocket)
```

Parameters:

(i) socket	The socket to close.

Returns:

(i)	@TRUE if the socket was closed successfully, else @FALSE.

SGetLastErr():

@SOK	Closed the socket OK.
10000..11004	Winsock error code (see Appendix A).

See Also:
SOpen

SConnect

Connects a socket to an Internet host & network service (i.e. "ftp").

Syntax:
```
   nRet = SConnect (hSocket, szHost, szService)
```

Parameters:

(i) socket	Which socket to connect.
(s) hostaddr	Which host to connect to. This can be either a symbolic host name like "myserv.com", or a dotted-decimal IP address like "192.123.456.1".
(s) service	The name of the service to connect this socket to, or the service's port number (ex: "time", or "37"). NOTE: To connect the socket to an http port, you must specify its port # ("80") directly.

Returns:

(i)	@TRUE if everything's OK, else @FALSE.

SGetLastErr():

@SOK	Connected OK.
@SCancel	User hit Cancel in the "Connect To" dialog, or unknown host name.
@SErrParam	Unknown socket # specified.
@SErrService	Unknown service name specified.
@SErrIPAddr	Invalid IP dotted-decimal address specified.
@SErrHostName	Unknown host name specified.
@SErrNoConn	Host computer refused to connect, possibly because it doesn't support the requested service.
@SErrBusy	Host computer too busy to connect.
10000..11003	Winsock error code (see Appendix A).

If you're not already hooked up to an Internet Service Provider, this will bring up the "Connect To" dialog. If you want it to dial up without human intervention, you must call DUNConnect first.

If the server doesn't connect within the global timeout # of seconds (default is 20), then SConnect will return @SErrNoConn. You can change this global timeout with SSetParam.

See Also:
SClose, DUNConnect, SOpen, SSetParam

SGetLastErr

Gets the last error generated by a WXSock extender function.

Syntax:

```
nErr = SGetLastErr ()
```

Returns:

(i)	The result of the last Sxxx function. (See the specific function description).

See Also:
P3GetReply

SGetParam

Gets a global WXSock parameter.

Syntax:

```
nValue = SGetParam (nParamID)
```

Parameters:

(i) paramID	Which parameter to get. The only valid value is: @SParTimeout - How many seconds to wait for a SConnect to connect to a server. The default is 20.

Returns:

(i)	The value of the specified parameter.

See Also:
SSetParam

SSetParam

Sets a global WXSock parameter.

Syntax:
```
nOldValue = SSetParam (nParamID, nNewValue)
```

Parameters:

(i) paramID	Which parameter to set. The only valid value is: @SParTimeout - How many seconds to wait for a SConnect to connect to a server. The default is 20.
(i) newvalue	The new value of the parameter.

Returns:

(i)	The previous value of the specified parameter.

See Also:
SGetParam

SMTPSendText

Sends an email message. This message must be text-only.

Syntax:
```
SMTPSendText (sMailSvr, sFrom, sTo, sSubject, sText)
```

Parameters:

(s) server	Your mail server; i.e. "mail.nerds-r-us.com".
(s) from	Your mail address; i.e. “myname@nerds-r-us.com”.
(s) to	The recipient's mail address.
(s) subject	A subject line. This can be "".
(s) text	The message body.

Returns:

(number)	@TRUE if the message was sent OK, else @FALSE if an error or the user hit Ctrl+Break.

SGetLastErr():

@SOK	Sent the message OK.

Other errors from SOpen, SConnect, SSendLine, or SRecvLine.

See Also:
P3xxx commands

SOK2Recv

If we received data from this socket now, would we get it immediately?

Syntax:

```
bOK = SOK2Recv (hSocket, nSize)
```

Parameters:

(i) socket	A handle specifying the socket to check.
(i) size	How many bytes we want to receive.

Returns:

(i)	@TRUE if an SRecvXXX command could be carried out without waiting, else @FALSE if the WIL script would have to wait first.

SGetLastErr():

@SOK	Determined the status OK.
10000..11004	Winsock error code from its recv function (see Appendix A).

The SRecvXXX functions all wait until this computer actually receives all the data the function is requesting. This function tells us if there is currently enough data in the receive queue to return immediately.

See Also:
SOpen, SConnect, SRecvXXX

SOK2Send

If we sent data thru this socket now, would it be sent immediately?

Syntax:

```
bOK = SOK2Send (hSocket)
```

Parameters:

(i) socket	A handle specifying the socket to check.

Returns:

(i)	@TRUE if SSendXXX command could be carried out without waiting, else @FALSE if the WIL script would have to wait.

SGetLastErr():

@SOK	Determined the status OK.

The SSendXXX functions all wait until there is enough space available in this computer's send buffer to put the data we're sending. This function tells us if there is currently enough space in the send queue to do it immediately.

See Also:
SOpen, SConnect, SSendXXX

SOpen

Creates a new stream (TCP) socket.

Syntax:

```
hSocket = SOpen ()
```

Returns:

(i)	A handle specifying the socket that was created, or @FALSE if there was an error.

SGetLastErr():

@SOK	The socket was created successfully.
@SErrWinsock	Couldn't initialize Winsock. WIL needs Winsock v1.1 or higher.
@SErrSocket	Error creating the socket.

See Also:

SConnect, SClose

When sending & receiving data to/from a remote host, your script can't control how fast the process runs. The server could be slow in responding to your requests for data, or you may be pushing so much data out faster than the network can process it that your Winsock send buffer may fill up. In either case the socket has to wait.

Standard Winsock programming recognizes two kinds of sockets: Blocking and non-blocking. If you create a socket as "blocking", any send or receive calls you make will wait until all the data can be sent/received. This makes for by far the simplest code. However, if the server stops responding or the network seems hung up, the user can't hit Ctrl+Break to "wake up" the application.

On the other hand, when a "non-blocking" socket can't send or receive all the data, it returns immediately with an error code telling the program to wait a bit. The program is expected to enter a loop and retry the call until the send or receive is successful, or else set up an event notification complete with callback function or dummy hidden window to receive Windows messages telling of the change in status.

WIL sockets are "psuedo-blocking", and these give you the best of both worlds. As far as your script is concerned, any socket you create is a blocking socket—you can just call SRecvXXX or SSendXXX, and when they return they've either processed the data or there was some other error. But the user can still hit Ctrl+Break if the network seems hung up. For example:

```
<Open a socket & connect it to something>

nRet = SSendLine (hSocket, sCmd)
if (nRet==@FALSE)
     Message (sTitle, "Error sending data!")
     goto Cancel
endif

sData = SRecvLine (hSocket)
nErr = SGetLastErr()
if (nRet<>@SOK)
     Message (sTitle, "Error receiving data!")
     goto Cancel
endif

<Do something with sData>
```

```
; We're thru with the socket, there was an error, or user
; hit Ctrl+Break during the operation. Just close it...
:Cancel
SClose (hSocket)
exit
```

SRecvBinary

Gets binary data from a socket into a binary data type. This must have been created with BinaryAlloc.

Syntax:

```
bOK = SRecvBinary (hSocket, hData, nMaxChars)
```

Parameters:

(i) socket	A handle specifying the socket to get data from.
(i) blob	A handle specifying the binary object that receives the data.
(i) size	The number of characters to receive.

Returns:

(i)	@TRUE if the data was received OK, else @FALSE if there was an error or not enough data ever came in.

SGetLastErr():

@SOK	Received the data OK.
@SErrNoConn	The server has already closed the connection.
10000..11004	Winsock error code (see Appendix A).

See Also:

SOpen, SConnect, SSendBinary, SRecvLine, SRecvNum8/16/32

SRecvLine

Gets a line of text from a socket, up to the first CR/LF.

Syntax:

```
sData = SRecvLine (hSocket, nMaxChars)
```

Parameters:

(i) socket	A handle specifying the socket to get data from.
(i) maxsize	The maximum characters to receive in one line. (This value must be less than 65535, which is the total allowed for all strings in your WIL script. A good value would be 32767.)

Returns:

(s)	The next line of text in the receive buffer, with its trailing CR/LF stripped off.

SGetLastErr():

@SOK	Received the data OK.
@SErrParam	Max chars requested > 65535.
@SErrNoConn	The server has already closed the connection.
10000..11004	Winsock error code (see Appendix A).

See Also:

SOpen, SConnect, SSendLine, SSendString

SRecvNum8
SRecvNum16
SRecvNum32

Gets numeric data from a socket. The result is then converted from network byte order to PC byte order for you.

Syntax:

```
nData = SRecvNum8 (hSocket)
nData = SRecvNum16 (hSocket)
nData = SRecvNum32(hSocket)
```

Parameters:

(i) socket	The socket to get data from.

Returns:

(i)	The next 1, 2, or 4 bytes of data received, expressed as a signed 8-, 16-, or 32-bit number. (These are converted, if necessary, from network byte order to PC byte order.)

SGetLastErr():

@SOK	Received the data OK.
@SErrNoConn	The server has already closed the connection.
10000..11004	Winsock error code (see Appendix A).

See Also:

SOpen, SConnect, SSendNum8, SSendNum16, SSendNum32

SSendBinary

Sends binary data to a socket.

Syntax:

```
nRet = SSendBinary (hSocket, sData, nLen)
```

Parameters:

(i) socket	A handle specifying the socket to send data to.
(i) blob	The binary object to send.
(i) size	How many bytes to send.

Returns:

(i)	@TRUE if all the data was sent OK, else @FALSE if there was an error.

SGetLastErr():

@SOK	Sent the data OK.
10000..11004	Winsock error code (see Appendix A).

See Also:
SOpen, SConnect, SSendString, SSendNum8, SSendNum16,
SSendNum32

SSendNum8
SSendNum16
SSendNum32

Sends an integer to a socket.

Syntax:
```
nRet = SSendNum8   (hSocket, nData)
nRet = SSendNum16 (hSocket, nData)
nRet = SSendNum32 (hSocket, nData)
```

Parameters:

(i) socket	A handle specifying the socket to send data to.
(i) number	The 1-, 2-, or 4-byte number to send. (This is converted, if necessary, from PC byte order to network byte order.)

Returns:

(i)	@TRUE if all the data was sent OK, else @FALSE if there was an error.

SGetLastErr():

@SOK	Sent the data OK.
10000..11004	Winsock error code (see Appendix A).

See Also:
SOpen, SConnect, SSendBin, SSendString

SSendString
SSendLine

Sends a string to a socket. SSendLine() will first append a CR/LF on the end of the string if it isn't already there.

Syntax:
```
nRet = SSendString (hSocket, sData)
nRet = SSendLine    (hSocket, sData)
```

Parameters:

(i) socket	The socket to send data to.
(s) string	The data to send.

Returns:

(i)	@TRUE if all the data was sent OK, else @FALSE if there was an error.

SGetLastErr():

@SOK	Sent the data OK.
10000..11004	Winsock error code (see Appendix A).

See Also:
SOpen, SConnect, SSendBin, SSendNum8, SSendNum16, SSendNum32

URLDecode

Takes a string in urlencoded format and converts it to plain text.

Syntax:
sPlain = URLDecode (sQuery)

Parameters:

(s) data	The string to decode.

Returns:

(s)	The string in plain text.

See Also:
URLEncode

URLEncode

Takes a plain text string and converts it to urlencoded format.

Syntax:
sData = URLEncode (sPlainText)

Parameters:

(s) data	The string to encode.

Returns:

(s)	The string in urlencoded form.

When a web client (a browser for instance) sends a request to a CGI script running on a web server, the data is sent to the CGI program appended onto the end of a URL. In this format, there can't be any spaces or control characters in the data that's sent.

Because of this, you have to "urlencode" the data values you send. In urlencoding, spaces are turned into "+"s, and punctuation marks or control characters are "escaped" into a "%" and two hex digits.

If you want to simulate a user filling out a web form, you must call URLEncode for each data value you're including in the query string.

Example:
```
sField1 = "My name"
sField2 = "My address, Apt. #c"
sField1 = URLEncode (sField1)
sField2 = URLEncode (sField2)
sQuery = "name=%sField1%&addr1=%sField2%"
```

At this point, sQuery would be `"name=My+name&addr1=My+address%2C+Apt.+%23c"`

```
sPage = HTTPRecvQuery ("www.someserver.com",
         "\cgi-bin\formproc", sQuery, 32767, @FALSE)
```

See Also:
URLDecode

URLGetScheme

Extracts the scheme from a URL. A fully-qualified URL comes in this syntax:

```
          <scheme>:<details>
ex.:   http://www.windowware.com/index.html
       ftp://ftp.windowware.com/WinBatch/wbt32i.dll
       mailto:morriew@windowware.com
```

Each scheme has its own syntax for the details part.

Syntax:
sScheme = HTTPGetScheme (sURL, sDefault)

Parameters:

(s) URL	The URL to parse. This can be a fully-qualified URL or relative.
(s) default	The default scheme to use.

Returns:

(s)	The URL’s scheme if found, else the default.

See Also:
HTTPGetAnchor, HTTPGetDir, HTTPGetFile, HTTPGetPath, HTTPGetQuery, HTTPGetServer, HTTPRecvQuery, HTTPRecvText

APPENDIX A: Winsock error codes

10004	WSAEINTR
10009	WSAEBADF
10013	WSAEACCES
10014	WSAEFAULT
10022	WSAEINVAL
10024	WSAEMFILE
10035	WSAEWOULDBLOCK (SSend/SRecv - Gets translated to @SErrMustWait)
10036	WSAEINPROGRESS
10037	WSAEALREADY
10038	WSAENOTSOCK (SConnect - gets translated to @SErrSocket)
10039	WSAEDESTADDRREQ
10040	WSAEMSGSIZE
10041	WSAEPROTOTYPE
10042	WSAENOPROTOOPT
10043	WSAEPROTONOSUPPORT
10044	WSAESOCKTNOSUPPORT
10045	WSAEOPNOTSUPP
10046	WSAEPFNOSUPPORT
10047	WSAEAFNOSUPPORT
10048	WSAEADDRINUSE
10049	WSAEADDRNOTAVAIL
10050	WSAENETDOWN
10051	WSAENETUNREACH
10052	WSAENETRESET (SConnect - gets translated to @SErrNoConn)
10053	WSAECONNABORTED (SConnect - gets translated to @SErrNoConn)
10054	WSAECONNRESET
10055	WSAENOBUFS
10056	WSAEISCONN
10057	WSAENOTCONN
10058	WSAESHUTDOWN
10059	WSAETOOMANYREFS
10060	WSAETIMEDOUT (SConnect - gets translated to @SErrNoConn)
10061	WSAECONNREFUSED (SConnect - gets translated to @SErrBusy)
10062	WSAELOOP
10063	WSAENAMETOOLONG
10064	WSAEHOSTDOWN
10065	WSAEHOSTUNREACH
10066	WSAENOTEMPTY
10067	WSAEPROCLIM
10068	WSAEUSERS
10069	WSAEDQUOT
10070	WSAESTALE
10071	WSAEREMOTE
10091	WSASYSNOTREADY
10092	WSAVERNOTSUPPORTED

10093	WSANOTINITIALISED
10101	WSAEDISCON
11001	WSAHOST_NOT_FOUND
11002	WSATRY_AGAIN
11003	WSANO_RECOVERY
11004	WSANO_DATA (SConnect - gets translated to @SCancel)

APPENDIX. B: Dial-Up Networking return codes

600	PENDING
601	ERROR_INVALID_PORT_HANDLE
602	ERROR_PORT_ALREADY_OPEN (DUNConnect - becomes @SAlready)
603	ERROR_BUFFER_TOO_SMALL (User has > 16 Dialups defined)
604	ERROR_WRONG_INFO_SPECIFIED
605	ERROR_CANNOT_SET_PORT_INFO
606	ERROR_PORT_NOT_CONNECTED
607	ERROR_EVENT_INVALID
608	ERROR_DEVICE_DOES_NOT_EXIST
609	ERROR_DEVICETYPE_DOES_NOT_EXIST
610	ERROR_BUFFER_INVALID
611	ERROR_ROUTE_NOT_AVAILABLE
612	ERROR_ROUTE_NOT_ALLOCATED
613	ERROR_INVALID_COMPRESSION_SPECIFIED
614	ERROR_OUT_OF_BUFFERS
615	ERROR_PORT_NOT_FOUND
616	ERROR_ASYNC_REQUEST_PENDING
617	ERROR_ALREADY_DISCONNECTING
618	ERROR_PORT_NOT_OPEN
619	ERROR_PORT_DISCONNECTED
620	ERROR_NO_ENDPOINTS
621	ERROR_CANNOT_OPEN_PHONEBOOK
622	ERROR_CANNOT_LOAD_PHONEBOOK
623	ERROR_CANNOT_FIND_PHONEBOOK_ENTRY (DUNConnect - becomes @SErrNotFound)
624	ERROR_CANNOT_WRITE_PHONEBOOK
625	ERROR_CORRUPT_PHONEBOOK
626	ERROR_CANNOT_LOAD_STRING
627	ERROR_KEY_NOT_FOUND
628	ERROR_DISCONNECTION
629	ERROR_REMOTE_DISCONNECTION
630	ERROR_HARDWARE_FAILURE
631	ERROR_USER_DISCONNECTION
632	ERROR_INVALID_SIZE
633	ERROR_PORT_NOT_AVAILABLE
634	ERROR_CANNOT_PROJECT_CLIENT
635	ERROR_UNKNOWN
636	ERROR_WRONG_DEVICE_ATTACHED
637	ERROR_BAD_STRING
638	ERROR_REQUEST_TIMEOUT
639	ERROR_CANNOT_GET_LANA
640	ERROR_NETBIOS_ERROR
641	ERROR_SERVER_OUT_OF_RESOURCES
642	ERROR_NAME_EXISTS_ON_NET
643	ERROR_SERVER_GENERAL_NET_FAILURE
644	WARNING_MSG_ALIAS_NOT_ADDED

645	ERROR_AUTH_INTERNAL
646	ERROR_RESTRICTED_LOGON_HOURS
647	ERROR_ACCT_DISABLED
648	ERROR_PASSWD_EXPIRED
649	ERROR_NO_DIALIN_PERMISSION
650	ERROR_SERVER_NOT_RESPONDING
651	ERROR_FROM_DEVICE
652	ERROR_UNRECOGNIZED_RESPONSE
653	ERROR_MACRO_NOT_FOUND
654	ERROR_MACRO_NOT_DEFINED
655	ERROR_MESSAGE_MACRO_NOT_FOUND
656	ERROR_DEFAULTOFF_MACRO_NOT_FOUND
657	ERROR_FILE_COULD_NOT_BE_OPENED
658	ERROR_DEVICENAME_TOO_LONG
659	ERROR_DEVICENAME_NOT_FOUND
660	ERROR_NO_RESPONSES
661	ERROR_NO_COMMAND_FOUND
662	ERROR_WRONG_KEY_SPECIFIED
663	ERROR_UNKNOWN_DEVICE_TYPE
664	ERROR_ALLOCATING_MEMORY
665	ERROR_PORT_NOT_CONFIGURED
666	ERROR_DEVICE_NOT_READY
667	ERROR_READING_INI_FILE
668	ERROR_NO_CONNECTION
669	ERROR_BAD_USAGE_IN_INI_FILE
670	ERROR_READING_SECTIONNAME
671	ERROR_READING_DEVICETYPE
672	ERROR_READING_DEVICENAME
673	ERROR_READING_USAGE
674	ERROR_READING_MAXCONNECTBPS
675	ERROR_READING_MAXCARRIERBPS
676	ERROR_LINE_BUSY
677	ERROR_VOICE_ANSWER
678	ERROR_NO_ANSWER
679	ERROR_NO_CARRIER
680	ERROR_NO_DIALTONE
681	ERROR_IN_COMMAND
682	ERROR_WRITING_SECTIONNAME
683	ERROR_WRITING_DEVICETYPE
684	ERROR_WRITING_DEVICENAME
685	ERROR_WRITING_MAXCONNECTBPS
686	ERROR_WRITING_MAXCARRIERBPS
687	ERROR_WRITING_USAGE
688	ERROR_WRITING_DEFAULTOFF
689	ERROR_READING_DEFAULTOFF
690	ERROR_EMPTY_INI_FILE
691	ERROR_AUTHENTICATION_FAILURE
692	ERROR_PORT_OR_DEVICE
693	ERROR_NOT_BINARY_MACRO
694	ERROR_DCB_NOT_FOUND
695	ERROR_STATE_MACHINES_NOT_STARTED
696	ERROR_STATE_MACHINES_ALREADY_STARTED
697	ERROR_PARTIAL_RESPONSE_LOOPING
698	ERROR_UNKNOWN_RESPONSE_KEY

699	ERROR_RECV_BUF_FULL
700	ERROR_CMD_TOO_LONG
701	ERROR_UNSUPPORTED_BPS
702	ERROR_UNEXPECTED_RESPONSE
703	ERROR_INTERACTIVE_MODE
704	ERROR_BAD_CALLBACK_NUMBER
705	ERROR_INVALID_AUTH_STATE
706	ERROR_WRITING_INITBPS
707	ERROR_X25_DIAGNOSTIC
708	ERROR_ACCT_EXPIRED
709	ERROR_CHANGING_PASSWORD
710	ERROR_OVERRUN
711	ERROR_RASMAN_CANNOT_INITIALIZE
712	ERROR_BIPLEX_PORT_NOT_AVAILABLE
713	ERROR_NO_ACTIVE_ISDN_LINES
714	ERROR_NO_ISDN_CHANNELS_AVAILABLE
715	ERROR_TOO_MANY_LINE_ERRORS
716	ERROR_IP_CONFIGURATION
717	ERROR_NO_IP_ADDRESSES
718	ERROR_PPP_TIMEOUT
719	ERROR_PPP_REMOTE_TERMINATED
720	ERROR_PPP_NO_PROTOCOLS_CONFIGURED
721	ERROR_PPP_NO_RESPONSE
722	ERROR_PPP_INVALID_PACKET
723	ERROR_PHONE_NUMBER_TOO_LONG
724	ERROR_IPXCP_NO_DIALOUT_CONFIGURED
725	ERROR_IPXCP_NO_DIALIN_CONFIGURED
726	ERROR_IPXCP_DIALOUT_ALREADY_ACTIVE
727	ERROR_ACCESSING_TCPCFGDLL
728	ERROR_NO_IP_RAS_ADAPTER
729	ERROR_SLIP_REQUIRES_IP
730	ERROR_PROJECTION_NOT_COMPLETE
731	ERROR_PROTOCOL_NOT_CONFIGURED
732	ERROR_PPP_NOT_CONVERGING
733	ERROR_PPP_CP_REJECTED
734	ERROR_PPP_LCP_TERMINATED
735	ERROR_PPP_REQUIRED_ADDRESS_REJECTED
736	ERROR_PPP_NCP_TERMINATED
737	ERROR_PPP_LOOPBACK_DETECTED
738	ERROR_PPP_NO_ADDRESS_ASSIGNED
739	ERROR_CANNOT_USE_LOGON_CREDENTIALS
740	ERROR_TAPI_CONFIGURATION
741	ERROR_NO_LOCAL_ENCRYPTION
742	ERROR_NO_REMOTE_ENCRYPTION
743	ERROR_REMOTE_REQUIRES_ENCRYPTION
744	ERROR_IPXCP_NET_NUMBER_CONFLICT
745	ERROR_INVALID_SMM
746	ERROR_SMM_UNINITIALIZED
747	ERROR_NO_MAC_FOR_PORT
748	ERROR_SMM_TIMEOUT
749	ERROR_BAD_PHONE_NUMBER
750	ERROR_WRONG_MODULE