Programmer’s Guide to TCPSERVR

Release Version 1.0

Table of Contents

[Getting Started 3](#_Toc383359654)

[Running as a Windows Service 3](#_Toc383359655)

[Running at User Logon 3](#_Toc383359656)

[Running on a Single User 3](#_Toc383359657)

[Running on a Every User 3](#_Toc383359658)

[Customization 3](#_Toc383359659)

[Creating a Password 3](#_Toc383359660)

[History Setup 3](#_Toc383359661)

[Setting up a Custom Endpoint 3](#_Toc383359662)

[Other Command Line Arguments 3](#_Toc383359663)

[Incognito Mode 3](#_Toc383359664)

[Summary of Arguments 3](#_Toc383359665)

[Sending and Receiving Data 3](#_Toc383359666)

[Data Encoding and Protocol 3](#_Toc383359667)

[The Leading Bytes 3](#_Toc383359668)

[The Header 3](#_Toc383359669)

[Status Codes 3](#_Toc383359670)

[Successful Status Codes 3](#_Toc383359671)

[Errors on behalf of the client 3](#_Toc383359672)

[Errors on behalf of the server 3](#_Toc383359673)

[Formatting Commands 3](#_Toc383359674)

[List of special characters 3](#_Toc383359675)

[Running in Multiple Instances 3](#_Toc383359676)

[Master Application 3](#_Toc383359677)

[Slave Application 3](#_Toc383359678)

[Purpose of the Master-Slave Paradigm 3](#_Toc383359679)

[Named Pipe Communication 3](#_Toc383359680)

[Creating a Pipe 3](#_Toc383359681)

[Bugs and Error Logging 3](#_Toc383359682)

[TCPSERVR-BASIC 3](#_Toc383359683)

[Syntax 3](#_Toc383359684)

[Variables and Macros 3](#_Toc383359685)

[Macros 3](#_Toc383359686)

[Functions 3](#_Toc383359687)

[Return Values 3](#_Toc383359688)

[Status Codes 3](#_Toc383359689)

[User Defined Functions 3](#_Toc383359690)

[Conditions and Logical Statements 3](#_Toc383359691)

[Operators 3](#_Toc383359692)

[IF Statements 3](#_Toc383359693)

[Errors 3](#_Toc383359694)

[Loops 3](#_Toc383359695)

[WHILE 3](#_Toc383359696)

[DO 3](#_Toc383359697)

[List of Commands 3](#_Toc383359698)

[Server Control and Maintenance 3](#_Toc383359699)

[CLEARHISTORY 3](#_Toc383359700)

[END 3](#_Toc383359701)

[ENDCHILD 3](#_Toc383359702)

[GETCHILDPIPES 3](#_Toc383359703)

[GETREPORT 3](#_Toc383359704)

[HISTORY 3](#_Toc383359705)

[HISTORYLOG 3](#_Toc383359706)

[RESTART 3](#_Toc383359707)

[RESTARTCHILD 3](#_Toc383359708)

[UPDATE 3](#_Toc383359709)

[File Management 3](#_Toc383359710)

[7ZA 3](#_Toc383359711)

[CD 3](#_Toc383359712)

[CMD 3](#_Toc383359713)

[COPY 3](#_Toc383359714)

[DEL 3](#_Toc383359715)

[DIR 3](#_Toc383359716)

[GET 3](#_Toc383359717)

[GETDIR 3](#_Toc383359718)

[GETFILEATTRIBUTES 3](#_Toc383359719)

[GETFILEINFO 3](#_Toc383359720)

[GETMULT 3](#_Toc383359721)

[MD 3](#_Toc383359722)

[MOVE 3](#_Toc383359723)

[PUT 3](#_Toc383359724)

[RECYCLE 3](#_Toc383359725)

[SCRIPT 3](#_Toc383359726)

[SCRIPTOPEN 3](#_Toc383359727)

[SETACCESSDATE 3](#_Toc383359728)

[SETFILEATTRIBUTES 3](#_Toc383359729)

[SETCREATEDDATE 3](#_Toc383359730)

[SETMODIFIEDDATE 3](#_Toc383359731)

[PWD 3](#_Toc383359732)

[RD 3](#_Toc383359733)

[REN 3](#_Toc383359734)

[RUN 3](#_Toc383359735)

[Automation 3](#_Toc383359736)

[BLOCK 3](#_Toc383359737)

[BLOCKINPUT 3](#_Toc383359738)

[BLOCKEDLIST 3](#_Toc383359739)

[GETSCREEN 3](#_Toc383359740)

[MOUSECLICK 3](#_Toc383359741)

[MOUSEMOVE 3](#_Toc383359742)

[PROCESSCLOSE 3](#_Toc383359743)

[PROCESSEXISTS 3](#_Toc383359744)

[PROCESSKILL 3](#_Toc383359745)

[REMOVEX 3](#_Toc383359746)

[UNBLOCK 3](#_Toc383359747)

[SEND 3](#_Toc383359748)

[VOLUMEDOWN 3](#_Toc383359749)

[VOLUMEMUTE 3](#_Toc383359750)

[VOLUMEUP 3](#_Toc383359751)

[Window Commands 3](#_Toc383359752)

[WINACTIVATE 3](#_Toc383359753)

[WINCLOSE 3](#_Toc383359754)

[WINGETHANDLE 3](#_Toc383359755)

[WINGETTITLE 3](#_Toc383359756)

[WININFO 3](#_Toc383359757)

[WINKILL 3](#_Toc383359758)

[WINLIST 3](#_Toc383359759)

[WINLISTINFO 3](#_Toc383359760)

[WINMOVE 3](#_Toc383359761)

[WINPICTURE 3](#_Toc383359762)

[WINSETSTATE 3](#_Toc383359763)

[WINSETTITLE 3](#_Toc383359764)

[WINSETTRANS 3](#_Toc383359765)

[WINSIZE 3](#_Toc383359766)

[Registry Commands 3](#_Toc383359767)

[REGCREATEKEY 3](#_Toc383359768)

[REGDELETE 3](#_Toc383359769)

[REGDELETEKEY 3](#_Toc383359770)

[REGENUMKEYS 3](#_Toc383359771)

[REGENUMVALUES 3](#_Toc383359772)

[REGREAD 3](#_Toc383359773)

[REGRENAME 3](#_Toc383359774)

[REGRENAMEKEY 3](#_Toc383359775)

[REGWRITE 3](#_Toc383359776)

[Messaging and Communication 3](#_Toc383359777)

[BLOCKMESSAGE 3](#_Toc383359778)

[CHAT 3](#_Toc383359779)

[MSGBOX 3](#_Toc383359780)

[SAY 3](#_Toc383359781)

[TRAYTIP 3](#_Toc383359782)

[Thread Commands 3](#_Toc383359783)

[THREADABORT 3](#_Toc383359784)

[THREADDELETE 3](#_Toc383359785)

[THREADLIST 3](#_Toc383359786)

[THREADLISTCLEAN 3](#_Toc383359787)

[THREADSTATE 3](#_Toc383359788)

[Pipe Commands 3](#_Toc383359789)

[PIPECREATE 3](#_Toc383359790)

[PIPELISTUSERS 3](#_Toc383359791)

[PIPEUSE 3](#_Toc383359792)

[USER 3](#_Toc383359793)

[USINGPIPE 3](#_Toc383359794)

[Scripting Commands 3](#_Toc383359795)

[Data Declaration 3](#_Toc383359796)

[DELVAR 3](#_Toc383359797)

[DIM 3](#_Toc383359798)

[LET 3](#_Toc383359799)

[RETURN 3](#_Toc383359800)

[SETSTATUS 3](#_Toc383359801)

[String Manipulation 3](#_Toc383359802)

[APPEND 3](#_Toc383359803)

[CONTAINS 3](#_Toc383359804)

[INDEX 3](#_Toc383359805)

[LEFT 3](#_Toc383359806)

[LEN 3](#_Toc383359807)

[LOWER 3](#_Toc383359808)

[TAKE 3](#_Toc383359809)

[UPPER 3](#_Toc383359810)

[Mathematics 3](#_Toc383359811)

[EVAL 3](#_Toc383359812)

[EVALBOOL 3](#_Toc383359813)

[FPART 3](#_Toc383359814)

[IPART 3](#_Toc383359815)

[NOT 3](#_Toc383359816)

[POW 3](#_Toc383359817)

[RANDOM 3](#_Toc383359818)

[ROUND 3](#_Toc383359819)

[Script Processing 3](#_Toc383359820)

[DO 3](#_Toc383359821)

[IF 3](#_Toc383359822)

[BREAK 3](#_Toc383359823)

[EXIT 3](#_Toc383359824)

[WHILE 3](#_Toc383359825)

[Miscellaneous 3](#_Toc383359826)

[INPUT 3](#_Toc383359827)

[SLEEP 3](#_Toc383359828)

# Getting Started

The server can be run “out of the box” simply by using the command line arguments:

tcpservr.exe –start

However, TCPSERVR can be run at Windows startup or user logon. This section details these startup options as well as other settings.

## Running as a Windows Service

TCPSERVR can be run on the NT ACCOUNTS\System user by running the program as a service. Use the following arguments:

tcpservr.exe -install service

Alternatively, to uninstall this service, type

tcpservr.exe -remove service

These commands require administrative privileges to work.

Because TCPSERVR is not programmatically designed as a Windows Service, when it is installed as a service, it runs the following program with arguments:

cmd /c start [PATH]\tcpservr.exe –start

where [PATH] is the location of the program.

Because the service is run in this way, the location of TCPSERVR cannot have any spaces in its path. It is best to place TCPSERVR in a folder on the root directory of a drive.

**IMPORTANT:** When the server starts as a service, Windows will return an error. This is normal.

## Running at User Logon

TCPSERVR can run at user logon, either on select users or every user.

Moving TCPSERVR after it has been installed will require the program to be uninstalled, and then reinstalled. The program can be uninstalled before or after it has been moved.

**IMPORTANT:** These commands edit certain registry values! Do not attempt to edit the registry yourself without backing up!

### Running on Select Users

To install TCPSERVR on a single user, type the following in the Command Prompt:

tcpservr.exe -install HKCU

This will create a registry entry in the key HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run

Because this is not a system wide change, these commands do not need elevated privileges.

Similarly, the registry entry can be removed by using the command:

TCPSERVR -remove HKCU

### Running on Every User

If you are an administrator and wish to have TCPSERVR run on every user, use the following command:

TCPSERVR -install HKLM

This will create a registry entry in

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

This can be removed similarly by using the command

TCPSERVR -remove HKLM

**IMPORTANT:** These commands require elevated privileges.

## Customization

Altering any of these settings will create a configuration file (“config.ini”) in the same directory as the server application.

### Creating a Password

Data sent over the socket is not encrypted by default. However, if you require that information be encrypted with a password, use the following command:

tcpservr.exe -install pass [password]

Passwords are restricted to 16 characters in length.

The server will return a generic response to commands if information has not been properly encrypted. That is, if you ask the server to encrypt messages with a key, you will be required to send your messages encrypted with the same key. Encrypted messages are sent prefixed by ENC whereas unencrypted messages are prefixed by TCP. If an encrypted message cannot be decoded by the server, the server will prefix its message with TCP and send an unencrypted 401 error.

### History Setup

By default, the server logs all commands sent to it in order to keep reports on each of the commands. This setting can be turned off manually for each thread or the default can be altered.

To change the default setting, use the following command:

tcpservr.exe -install history [ON|OFF]

This will turn the automatic history logging ON or OFF respectively.

### Setting up a Custom Endpoint

The server by default listens at the following endpoint: LOCALHOST:2200

It is not recommended that you change the address unless there is specific reason to do so. However, if this is required, use the following command:

tcpservr.exe -install address [custom address]

The default port is 2200. You can change the default port if it conflicts with another program or you otherwise do not have access to this port. To do this, use the following command:

tcpservr.exe -install port [custom port]

These commands do not have a removable counterpart. To change the values to the default, simply delete the configuration file or pass empty arguments to the command.

## Incognito Mode

All of these install/remove commands can be run without prompt, or in “incognito mode”. This will not prompt the user with a success or error message. In order not to be prompted with any success or error message, prepend a ‘q’ before the argument.

Exemplia Gratia:

-install → -qinstall

-remove → -qremove

## Summary of Arguments

Here is a complete list of the arguments accepted by TCPSERVR:

|  |  |
| --- | --- |
| **Arguments** | **Function** |
| -START | Runs TCPSERVR normally |
| "SCRIPT PATH" | Attempts to run a TBASIC script. This will return an invalid argument error if the file does not exist. |
| -R "EXECUTABLE" [ ARGUMENTS ] | Runs an application ignoring the first argument. This is generally used in order to redirect an existing application. See the BLOCK command. |
| -M "FLAG" "TEXT" "TITLE" | Create a custom message box. This takes the same arguments as the MSGBOX command. |
| -INSTALL "SETTING" "VALUE" | Use for setting up TCPSERVR |
| -REMOVE "SETTING" "VALUE" | Use for uninstalling TCPSERVR |

These arguments cannot be combined because they perform different functions.

# Sending and Receiving Data

## Data Encoding and Protocol

Data sent is sent using TCP/IP. All strings are encoded using UTF-8.

## The Leading Bytes

Each request to the server expects at least seven bytes. Any message without these seven leading bytes will be dropped.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 84 | 67 | 80 | 17 | 0 | 0 |
| 0 | 50 | 48 | 48 | 32 | 80 |
| 87 | 68 | 13 | 10 | 67 | 58 |
| 92 | 85 | 115 | 101 | 114 | 115 |

The first three bytes are encoded with UTF8 the letters “TCP” or “ENC” to validate the message. This way, the server will not misinterpret the message size, and it will not hang indefinitely. Messages that are prefixed with “ENC” are expected to be encrypted. TCPSERVR will send encrypted messages using this prefix and will receive encrypted messages with this prefix if that is expected.

The next four bytes indicate the length of the message (Integer 32-bit). The length of the above message is 17 bytes (excluding the seven leading bytes). If this length is invalid, the server will either hang indefinitely, or fail to process the entire message. The four byte length of the message is never encrypted.

## The Header

When a command is sent to the server, the server’s response is prepended by a header. The header consists of both the original command, along with a status code.

The command

MouseMove 10 10

may return

200 MouseMove 10 10

OK

200represents the status code. Status codes are determined by the success of the command. To view a list of status codes and their purpose, see below.

MouseMove 10 10 is the command (and any arguments) for which the data is a reply. This assists in the organization of data when multiple commands are sent at close intervals.

All headers are separated from the body message by a line break. If the server cannot decrypt the message, it will not be able to return the original message.

## Status Codes

This is a list of status codes that TCPSERVR uses. Status codes represent the success and availability of a command. If the server cannot process a command, the status code may give insight to this. The codes are based heavily off the IETF RFCs on HTTP status codes.

### Successful Status Codes

These codes are returned when the server has no errors to report. Generally, this means that the command completed successfully.

|  |  |
| --- | --- |
| **Code** | **Status** |
| 200 | OK  The server is reporting no error |
| 201 | Created  The data has been created on the host machine |
| 202 | Accepted  The server has processed the command but is unable to confirm its completion and success. |
| 203 | Non-Authoritative Information  The server has completed the request, but is returning information from another source. |
| 204 | No Content  The server has no content to return for the request. |
| 206 | Warning  The command was completed with warnings. This may mean that the command was only partially completed. |

### Errors on behalf of the client

These codes represent errors in the request, and the server was unable to complete it. These status codes can generally be remedied by modifying the request.

|  |  |
| --- | --- |
| **Code** | **Status** |
| 400 | Bad Request  The server cannot process the command because it was formatted incorrectly (i.e. wrong or invalid arguments) |
| 401 | Unauthorized  The server requires an encryption, or the message that was sent was not encrypted in a way that the server understand. |
| 403 | Forbidden  The server does not have permission to execute the command. The command may be able to execute on another user with appropriate privileges. |
| 404 | Not Found  The server is unable to locate the data requested |

|  |  |
| --- | --- |
| 409 | Conflict  The server cannot execute the command because the server has multiple instances of an object opened, or there are multiple objects that wish to occupy the same memory |
| 423 | Locked  The resource requested cannot be accessed because it may be in use |

### Errors on behalf of the server

These status codes represent errors in executing the command over which the user has little to no control. Codes in this category may not be able to be remedied.

|  |  |
| --- | --- |
| **Code** | **Status** |
| 500 | Internal Server Error  This is a generic error. A description is usually returned by the server along with this status code. |
| 501 | Not Implemented  The server has not implemented the command you specified. This is more often an error on behalf of the client. |
| 502 | Bad Gateway  An error occurred when the server was receiving data from a pipe. This will occur if the pipe does not exist or data received from a pipe was invalid. |
| 507 | Insufficient Memory  The command could not execute because not enough memory could be allocated in order to process it |

## Formatting Commands

Arguments for commands are separated by spaces. Arguments that contain spaces are put in quotation marks.

Example:

MSGBOX 0 "How are you?" "Greetings"

Some commands require special formatting. Special character codes must be used for certain characters.

Example:

SEND "Call me @quoteIshmael@quote"

This will send literally:

Call me "Ishmael"

Usually, these special characters will be sent TO the server. However, certain commands return data with these codes. The command description will make this evident.

### List of special characters

|  |  |  |
| --- | --- | --- |
| Character | Description | Code |
| " | A quotation mark | @quote |
| CRLF | Line break and a carriage return | @break |
| | | A pipe character | @pipe |
| @ | An "at" symbol | @at |

# Running in Multiple Instances

## Master Application

By default, TCPSERVR runs as the “master” server. In this mode, commands are received directly from the network stream and carried out. There is very little need for setup. When this application is closed, all slave applications will close in its wake.

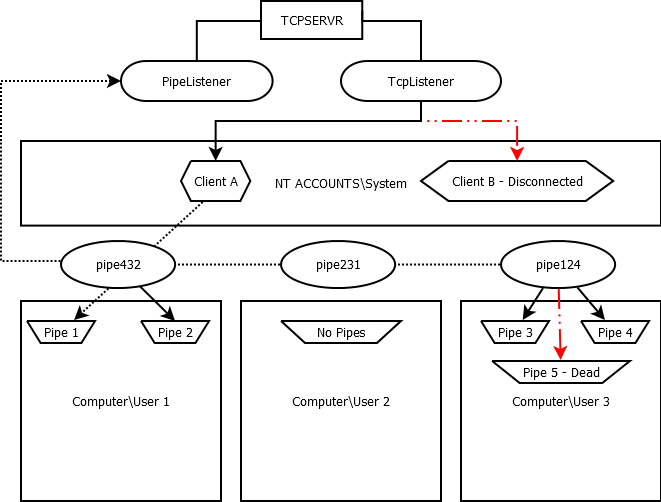
## Slave Application

When TCPSERVR detects that another instance of the program is running, it enters what is called “Slave Mode”. In this mode, the slave application will contact the already open master application with information concerning the current user on which it is running and a unique pipe name that the master application may use to contact it.

## Purpose of the Master-Slave Paradigm

This mode is especially useful for interacting with multi-user environments. While the dominant server can run as a Windows service, other users can log on creating more instances of the program. In this way, the client can “pipe” commands through the dominant application to the submissive applications on each user.

If information is received from a pipe that cannot be processed by TCPSERVR, it will return to the client 502 Bad Gateway. You may try the request again, but the pipe may not be compatible with the current version of TCPSERVR.

The following diagram represents this model:

The main application sits on the NT ACCOUNTS\System user, which has administrative privileges. As each user logs on, a slave application initializes and informs the master application that it will be receiving messages on a given pipe.

## Named Pipe Communication

Named pipes are a way for applications to communicate with each other in Windows. Messages sent between the dominant application and the submissive applications are formatted the same way as those sent over the TCP protocol. Keep in mind that oftentimes, the dominant application will only read part, if any of the message it receives before passing its pure byte data to a submissive application.

## Creating a Pipe

All submissive programs already have a pipe to communicate with the dominant program. However, this pipe is not used to interpret commands, but rather it is used to spur the creation of pipes that can. In this way, if one pipe crashes, a new pipe can be easily created.

The first step to creating a pipe to the submissive application is to retrieve a list of available subordinates. To do this, use the command:

PIPELISTUSERS

The returned data will look similar to this:

200 PIPELISTUSERS

USER PIPE

Timothy pipe302

John pipe219

From this information, we can gather that there are two users currently running submissive TCPSERVR applications.

The most efficient way to begin communicating with a submissive application is to simply use the command USER. This will create a new pipe on that particular user.

USER Timothy

This command creates or uses the pipe ‘Timothy’ to operate on user ‘Timothy’. After the pipe has been created, all new commands will be sent to it.

If a particular pipe is not working, you can create a pipe on which to operate with a different name. This is normally not necessary, and using the above command is recommended.

To create a pipe for communication to the application on “Timothy”, use the command

PIPECREATE pipe302 TestPipe

A new pipe will be created Timothy on called TestPipe. The new pipe can now receive commands.

The command

PIPEUSE TestPipe GETSCREEN

will send the command GETSCREEN to the TestPipe which has been created on the user Timothy. Therefore, it will return the screen image of that user.

To make all future commands go through this pipe, we can use the command

USINGPIPE TestPipe

Now, all future commands will be sent to TestPipe. If we not only want all future commands but also future connections to use this pipe, we add an argument

USINGPIPE TestPipe true

This command means that this connection and all future connections will send commands through this pipe. To no longer pipe any information to the submissive application, simply pass an empty string as an argument.

USINGPIPE ""

This will return:

200 USINGPIPE ""

No longer using a pipe

# Bugs and Error Logging

From time to time, TCPSERVR will run into various errors. These errors are logged in a file named “DUMP.TXT”. TCPSERVR attempts to be as discreet as possible when it crashes. Because of TCPSERVR’s multi-threaded and multi-process construction, errors generally do not cause the entire program to crash.

Here is an example of a logged error:

-----------------------------------------------

Date: 12/10/2013 08:36 AM

Status: Dominant

User: Family-PC\Timothy

Client: 127.0.0.1:56951

Method: Receive

Fatal: No

Message:

An established connection was aborted by the software in your host machine

The log entry consists of the date and time the error occurred, the rank of the application (dominant or submissive), the user on which it was operating, the method that caused the error, the client with which the application was communicating, and whether or not the error was fatal.

The user is the Windows profile on which the application was operating.

A dominant, or master, application will record the client with which it was communicating, whereas a submissive, or slave, application will record the pipe name.

The method that caused the error is for debugging purposes and understanding the source of the error.

The fatality determines whether or not the application had to close completely because of the error. If there is no fatality, a connection to any client was definitely interrupted but does not necessarily mean a new connection cannot be created. The message gives insight to this.

# TCPSERVR-BASIC

TCPSERVR includes a scripting language called *TCPSERVR-BASIC,* or simply *TBASIC*. Scripts written in this language can be run with the commands “SCRIPT” and “SCRIPTOPEN”. See the list of commands for their differences. These scripts must be included on the host machine.

Scripts can be useful for automation on the host machine. For example, if the client needed to prompt the user to close a Calculator window, one could use the script below.

hwnd$ = WinGetHandle("Calculator")

if hwnd$ = 0 then

MsgBox(16, "The Calculator is not open!", "Error!")

else

MsgBox(36, "Would you like to close the calculator?", "Question")

if @return = "Yes" THEN

WinClose(hwnd$)

end if

end if

## 

## Syntax

TBASIC models its syntax off the BASIC programming languages. All functions return a value that can be directly stored into a variable. For example, the following function returns a value and stores it into variable foo$.

LET foo$ = MSGBOX(4, "Sample", "Test")

Functions can be nested within each other as parameters.

foo$ = MSGBOX(4, "You clicked " + MSGBOX(36, "Yes or No", ""),"Test")

Note that every statement must be left to its own line and strings are always required to be in quotes.

## Variables and Macros

Variables in TBASIC all end in ‘$’. Variables can be explicitly assigned with a LET statement or simply put on their own line. The language will attempt to evaluate numbers if it perceives it can be interpreted that way. The following code will store the number 10 in variable foo$.

LET foo$ = 5 + 5

TBASIC uses dynamic variable types and parses the variable appropriately upon assignment. There is no need for a variable to be defined as a string, integer, double, etc. Strings can be appended together by using the ‘+’ character to join them.

foo$ = "Hello" + " " + "World!"

This will store the value “Hello World!” in foo$.

### Macros

Macros are different than variables in that they are assigned by the language and cannot be assigned by the code directly. They can be used like any other variable.

The following are macros in TBASIC:

|  |  |
| --- | --- |
| **Macro** | **Description** |
| @break | A carriage return and new line |
| @error | See @status |
| @quote | The " character |
| @ret | See @return |
| @return | This is the return value of the previous function. It is not changed upon the execution of statements. This is the equivalent of @ret |
| @status | The status code of the previous function executed. This is the equivalent of @error |

## Functions

Almost all commands listed in this document can be made as functions in TBASIC. Those that cannot will be listed and will return an error when they are not able to be run. The arguments that the command takes are the same.

### Return Values

When a function completes its execution, it leaves behind a value, called the return value. These values can be manipulated like any other variable type. The return values of all functions are listed in this document. Return values can be assigned directly or their value is stored in the @return or @ret macro. Both macros hold the same value.

### Status Codes

Functions do not return a header in TBASIC. The status code is now stored in a macro called @status or @error. Both of these macros store the same value.

MSGBOX(4, "Sample", "Hello World!")

may store 200 in the above macros.

### User Defined Functions

A user can define custom functions by using the FUNCTION keyword. For example, the following function would take a specified number of characters from the middle of a string.

function getMiddle(str$, amnt$)

len$ = len(str$)

middle$ = ipart((len$ - amnt$) / 2)

result$ = take(str$, middle$, amnt$)

setstatus "200"

return result$

end function

This returns the middle amt$ characters from a string and sets the status code to 200.

This can be called at any point in the program. A user defined function will not execute if it is not otherwise referenced.

User defined functions are also not case sensitive.

## Conditions and Logical Statements

Boolean expressions in TBASIC can be evaluated in a number of ways. The binary values 1 and 0 can be used to represent true/false. The entire words may also be used, which are not case sensitive.

The following code determines whether or not both variable foo$ and variable bar$ are true and stores the outcome into memory in the @return macro or by direct assignment.

EVALBOOL(foo$ + " AND " + bar$)

You can place parentheses around multiple values so that the language will process it in the correct order:

EVALBOOL(foo$ + " AND " + "(" + bar$ + " OR " + fubar$ + ")")

If parentheses were absent, the command would evaluate foo$ AND bar$ first and then use that value and compare it to fubar$ with an OR operation. The above code processes bar$ OR fubar$ first and then compares it to foo$ with an AND operation.

The NOT function returns the opposite of the evaluation (i.e. if the result of the expression is FALSE, it will return TRUE)

NOT(foo$ + " AND " + "(" + bar$ + " OR " + fubar$ + ")")

### Operators

The following are accepted operators in TBASIC:

|  |  |
| --- | --- |
| Operator | Description |
| = | A simple equality; tests if both two operands are equal. See == |
| == | This achieves the same goal as = |
| <> | tests if two operands are not equal |
| != | This achieves the same goal as <> |
| > | Tests if one operand is greater than another. |
| >= | tests if one operand is greater than or equal to another |
| < | Tests if one operand is less than another |
| <= | Tests if one operand is less than or equal to another |

### IF Statements

IF statements are statements that express a logical evaluation of two data types. Proper IF statements are written as such:

IF foo$ = "Yes" THEN

MSGBOX(0,"OK","IF STATEMENT")

END IF

If the first condition is not met, the else keyword may be used.

IF foo$ = "Yes" THEN

MSGBOX(0,"You said 'Yes'.", "IF BLOCK")

ELSE

MSGBOX(0,"You did not say 'Yes'.", "ELSE BLOCK")

END IF

Negation can be achieved by use of the NOT keyword.

Multiple conditions can be present in IF statements, separated by the AND and OR keywords.

IF NOT (foo$ = "Yes") AND bar$ = "Ok" THEN

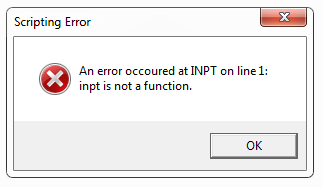
MSGBOX 0 "The conditions have been met." "IF STATEMENT"

END IF

Parentheses may also be added to ensure order of operations.

## Errors

If your script encounters an error, it will display a message box indicating the function and error that took place. By default, bad status codes are perceived as errors and a message box will be displayed. This can be avoided by using the HideError statement. This will NOT however hide errors due to bad syntax in which a line cannot be parsed. To show errors again, use ShowError.



## Loops

TBASIC supports two types of loops: the WHILE loop and the DO loop. A loop executes a certain block of code while a certain condition is true, or until a certain condition is met. Loops use the same logic as IF statements.

In order to break out of a loop, use the BREAK command. This will exit only the current loop, and it will not break from nested loops.

### WHILE

A WHILE loop acts when a condition is true. WHILE loops will not be entered if the initial condition is not met.

LET foo$ = 0

WHILE foo$ <> 5

MSGBOX(0,"The number is " + foo$,"WHILE LOOP")

LET foo$ = foo$ + 1

WEND

This loop displays a message box for every increment of foo$.

### DO

DO loops are different than WHILE loops in that DO loops executes the loop block at least once regardless of whether the condition has been met.

LET foo$ = 0

DO UNTIL foo$ = 5

MSGBOX(0,"The number is " + foo$,"DO LOOP")

LET foo$ = foo$ + 1

LOOP

The above code accomplishes the same task as the WHILE loop above only in a DO loop.

Similarly, the keyword WHILE can be used in a DO loop in order to execute a DO loop WHILE a condition is true.

LET foo$ = 0

DO WHILE foo$ <> 5

MSGBOX(0,"The number is " + foo$,"DO LOOP")

LET foo$ = foo$ + 1

LOOP

# List of Commands

This is an extensive list of commands that the server can accept.

## Server Control and Maintenance

This set of commands is used for server maintenance and control, such as updating, ending, and restarting. None of these commands can be used in TBASIC

### CLEARHISTORY

Clears all commands that have been logged by the server’s report manager

CLEARHISTORY [ THREAD ]

|  |  |
| --- | --- |
| Argument | Description |
| [ THREAD ] | The id of the thread for which to clear the history |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The report was returned |
| 404 | Either the thread was not found |

### END

Exits the remote application. This will end the master application, and all slave applications will close automatically. To end slave applications, use ENDCHILD.

END

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The exit message has been received and exiting is taking place |

### ENDCHILD

Exits a child application

ENDCHILD

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The exit message has been received and exiting is taking place |
| 501 | The command was not executed on a submissive application |

### GETCHILDPIPES

Gets all the pipes that have been created on the child application

GETCHILDPIPES

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The pipes have been returned successfully |
| 501 | The command was not executed on a submissive application |

### GETREPORT

Retrieves the report logged for a previously sent command. This can be used to check the status of a command that is currently being processed.

GETREPORT [ THREAD ] "MESSAGE ID"

|  |  |
| --- | --- |
| Argument | Description |
| [ THREAD ] | The id of the thread to which the message was sent. If this argument is not specified, it defaults to the current thread id |
| MESSAGE ID | The id of the message that was sent to the server. This can be found using the HISTORY command |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The report was returned |
| 404 | Either the command or the thread was not found |

### HISTORY

Retrieves the history of all commands sent to a particular thread. This also associates each message with a particular identification number in order to retrieve its full report.

HISTORY [ THREAD ]

|  |  |
| --- | --- |
| Argument | Description |
| [ THREAD ] | The id of the thread to which the message was sent. If this argument is not specified, it defaults to the current thread id. |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The report was returned |
| 404 | The thread was not found |

### HISTORYLOG

Specifies whether or not history should be logged on all threads or a specific thread

HISTORYLOG [ THREAD ] "ON|OFF"

HISTORYLOG "THREAD" "ON|OFF" "ALL|CURRENT"

|  |  |
| --- | --- |
| Argument | Description |
| [ THREAD ] | The id of the thread to which the message was sent. If this argument is not specified, it defaults to the current thread id. |
| ON|OFF | Specifies whether history should be turned ON or OFF |
| ALL|CURRENT | Specifies whether this setting should apply only to the specified thread or all future threads (Argument 1 must be specified in order for this argument to be set) |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The report was returned |
| 404 | The thread was not found |

### RESTART

Restarts the application. All submissive applications will end in its wake and will not be restarted

RESTART

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The restart message has been received and restarting is taking place |
| 409 | There was a conflict in the restarting process |

### RESTARTCHILD

Restarts the submissive application

RESTARTCHILD

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The restart message has been received and restarting is taking place |
| 409 | There was a conflict in the restarting process |
| 501 | The command was not executed on a submissive application |

### UPDATE

Replaces the application with TCPSERVR2.EXE (if it exists) and starts the new executable. This will end all submissive applications and will not restart them. If this command is run on a submissive application, the server will restart on that user as a dominant application, and all other instances will be terminated.

UPDATE

|  |  |
| --- | --- |
| Status | Description |
| 202 | The update message has been received and will begin once the server is exited |
| 404 | The update was not found |
| 409 | There was a conflict in the updating process |

## File Management

These commands manage files and folders on the host machine. Many of these commands utilize the Windows ® Command Prompt. The syntax may be determined by the version of the Command Prompt on the host machine.

### 7ZA

Uses the built-in 7zip program to compress or decompress archives. Run this command with no arguments for more details.

7ZA COMMAND [ SWITCHES ] "ARCHIVE NAME" [ FILE NAMES ]

|  |  |
| --- | --- |
| Argument | Description |
| COMMAND | The 7-Zip command to execute |
| [SWITCHES] | Any switches for the command |
| ARCHIVE NAME | The name of the archive to create |
| FILE NAMES | Any files to be added to the archive |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 203 | This is a generic status code from 7zr.exe  TCPSERVR is not directly handling the data |
| 206 | 7-Zip is reporting some inconsistencies, and some files may not have been compressed/extracted. |
| 500 | The 7-Zip program may not have been able to properly deploy |

### 

### CD

Changes the current directory for the application or shows the current directory. This syntax is widely based off the Windows® Command Prompt. This command does not require quotation marks around the path name, similar to modern Command Prompt versions. If no arguments are given, it will return the current path.

CD [ PATH ]

|  |  |
| --- | --- |
| Argument | Description |
| [ PATH ] | The path of the directory to which to move |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | [ The current path ] |
| 500 | Access may be denied to the folder |

### CMD

Runs a command directly from the Command Prompt. You can use this command to run an application that outputs data to the Command Prompt, and the data will be returned in the same format.

CMD COMMAND [ ARGUMENTS ]

|  |  |
| --- | --- |
| Argument | Description |
| Command | The Command Prompt command which to run |
| [ Arguments] | Any arguments the command may have |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 203 | TCPSERVR is relaying information from the command |
| 204 | The command has no data to return |
| 500 | The Command Prompt returned an error (for a variety of reasons) |

### COPY

Copies a file to a new location.

The syntax takes that of the Command Prompt. For complete documentation on this command, type COPY /?

COPY FILE1 FILE2

|  |  |
| --- | --- |
| Argument | Description |
| FILE1 | The file to be copied |
| FILE2 | The new location and filename of the file |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 203 | TCPSERVR is relaying information from the command |
| 204 | The command has no data to return |
| 500 | The Command Prompt returned an error (for a variety of reasons) |

### DEL

Deletes a file

The syntax takes that of the Command Prompt. For complete documentation on this command, type DEL /?

DEL FILE1

|  |  |
| --- | --- |
| Argument | Description |
| FILE1 | The file to delete |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 203 | TCPSERVR is relaying information from the command |
| 204 | The command has no data to return |
| 500 | The Command Prompt returned an error (for a variety of reasons) |

### DIR

Enumerates all files and folders in a directory

The syntax takes that of the Command Prompt. For complete documentation on this command, type DIR /?

DIR [ PATH ]

|  |  |
| --- | --- |
| Argument | Description |
| [ PATH ] | The directory to list |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 203 | TCPSERVR is relaying information from the command |
| 204 | The command has no data to return |
| 500 | The Command Prompt returned an error (for a variety of reasons) |

### GET

Downloads a file from the server. This command returns a Base64 string which must be converted into a byte array.

GET "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the file to download |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The file was successfully downloaded |
| 403 | The host user may not have sufficient privileges to access the data |
| 404 | The file cannot be found |
| 423 | The file may be being used by another process |
| 500 | The file may be too large |

### GETDIR

Downloads a directory from the server. This command will return a Base64 string of a ZIP archive.

GETDIR "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the directory to download |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The file was successfully downloaded |
| 403 | The host user may not have sufficient privileges to access the data |
| 404 | The file cannot be found |
| 423 | The file may be being used by another process |
| 500 | The file may be too large |

### GETFILEATTRIBUTES

Retrieves the attributes of a file or directory (see [*Attribute Codes*](#_Attribute_Codes_1)).

GETFILEATTRIBUTES "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the file |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The attributes have been returned succesfully |
| 403 | Access to the resource may be denied |
| 404 | The path cannot be found |
| 423 | The resource may be locked by another process |

### GETFILEINFO

Displays a file’s last access time, creation time, last write time, size, and attributes. Attributes are given as letters (see [*Attribute Codes*](#_Attribute_Codes_1)).

GETFILEINFO "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the file |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The directory information has been returned |
| 403 | Access to the resource may be denied |
| 404 | The directory cannot be found |
| 423 | The directory may be locked |

#### Attribute Codes

Attributes are listed as letters that correspond to each of the file’s properties.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Letter** | **Attribute** | | a | Archived | | c | Compressed | | e | Encrypted | | h | Hidden | | r | Read-Only | | s | System | | The attributes that a file has are compounded upon each other (i.e. a file’s attributes may be returned like this achr) |

### 

### GETMULT

Downloads multiple files from the server. This command will return a Base64 string of a ZIP archive.

GETMULT "FILE1" [ "FILE2" "FILE3" ... ]

|  |  |
| --- | --- |
| Argument | Description |
| FILE | The path of any files to download |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The file was successfully downloaded |
| 403 | The host user may not have sufficient privileges to access the data |
| 404 | One or more files cannot be found |
| 423 | One or more files may be being used by another process |
| 500 | The file may be too large |

### MD

Makes a directory

The syntax takes that of the Command Prompt. For complete documentation on this command, type MD /?

MD "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the directory to create |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 203 | TCPSERVR is relaying information from the command |
| 204 | The command has no data to return |
| 500 | The Command Prompt returned an error (for a variety of reasons) |

### MOVE

Moves a file.

The syntax takes that of the Command Prompt. For complete documentation on this command, type MOVE /?

MOVE "FILE1­" "FILE2"

|  |  |
| --- | --- |
| Argument | Description |
| FILE1 | The file to move |
| FILE2 | The new filename and path of the file |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 203 | TCPSERVR is relaying information from the command |
| 204 | The command has no data to return |
| 500 | The Command Prompt returned an error (for a variety of reasons) |

### PUT

Puts a file from the host computer to the current directory on the server. File size is limited to 2GB and large files may take longer to upload.

This command requires its second argument to be a byte array converted into a Base64 string.

This command will also not return a complete header. The second argument will be omitted when a response is returned.

PUT "FILE NAME" "BASE64 STRING"

|  |  |
| --- | --- |
| Argument | Description |
| FILE NAME | The path and file name to save |
| BASE64 STRING | All bytes of the file converted into a Base64 string |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The file was uploaded successfully |
| 403 | The server does not have permission to access the directory |
| 409 | The file already exists |
| 423 | The directory may be locked by the operating system |
| 500 | The file may be too large |

### 

### RECYCLE

Sends a file to the recycle bin, rather than permanently deleting it

Note that this command will prompt the user upon error, and it may still return a successful status code.

RECYCLE "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the file to recycle |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The file was successfully recycled |
| 404 | The file was not found |

### SCRIPT

Runs a TBASIC script. This runs the script by creating a child process, and it does not wait for the script to finish. It does not relay messages back to the server.

SCRIPT "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the script (on the host machine) to execute |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The server has attempted to run the script but cannot be sure it executed properly. It will also return the path of the child process. |

### SCRIPTOPEN

Runs a TCPSERVR script. This runs the script natively in TCPSERVR and requires that the script be completed before resuming commands. This command sends all errors and exceptions back to the client, rather than displaying them on the server.

SCRIPTOPEN "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the script (on the host machine) to execute |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The server has completed the script |
| 500 | An error occurred in the script |

### SETACCESSDATE

Sets the last access date and time for a file (or directory)

SETACCESSDATE "PATH" "DATE"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the file/folder |
| DATE | The date and time (mmm/dd/yyyy hh:mm:ss) |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The last access date was set |
| 403 | Access to the resource is denied |
| 404 | The file/folder cannot be found |
| 423 | The file is currently being used |

### SETFILEATTRIBUTES

Sets the attributes for a given file or directory. See the *Attributes Codes* table to properly format the attributes.

SETFILEATTRIBUTES "PATH" "ATTRIBUTES"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the file/folder |
| ATTRIBUTES | The file attributes. See *Attribute Codes* |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The last access date was set |
| 403 | Access to the resource is denied |
| 404 | The file/folder cannot be found |
| 423 | The file is currently being used |

### SETCREATEDDATE

Sets the creation date and time for a file (or directory)

SETCREATEDDATE "PATH" "DATE"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the file/folder |
| DATE | The date and time (mmm/dd/yyyy hh:mm:ss) |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The last access date was set |
| 403 | Access to the resource is denied |
| 404 | The file/folder cannot be found |
| 423 | The file is currently being used |

### SETMODIFIEDDATE

Sets the last write date and time for a file (or directory)

SETMODIFIEDDDATE "PATH" "DATE"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the file/folder |
| DATE | The date and time (mmm/dd/yyyy hh:mm:ss) |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The last access date was set |
| 403 | Access to the resource is denied |
| 404 | The file/folder cannot be found |
| 423 | The file is currently being used |

### PWD

Returns the current directory of the server computer. It is recommended to use “CD” for this action, but this command is in place for backward compatibility.

PWD

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The current directory |

### RD

Removes a directory

The syntax takes that of the Command Prompt. For complete documentation on this command, type RD /?

RD "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the directory to delete |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 203 | TCPSERVR is relaying information from the command |
| 204 | The command has no data to return |
| 500 | The Command Prompt returned an error (for a variety of reasons) |

### REN

Renames a file or directory

The syntax takes that of the Command Prompt. For complete documentation on this command, type REN /?

REN "FILE1" "FILE2"

|  |  |
| --- | --- |
| Argument | Description |
| FILE1 | The file to rename |
| FILE2 | The new name of the file |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 203 | TCPSERVR is relaying information from the command |
| 204 | The command has no data to return |
| 500 | The Command Prompt returned an error (for a variety of reasons) |

### RUN

Runs a program or file. A file must have an associated program in order to be run.

RUN "PATH" [ ARGUMENTS ]

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The path of the file to run |
| [ ARGUMENTS ] | Any arguments that need to be passed to the program |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The command executed successfully, but the server cannot be sure the program run did not err |
| 403 | Access is denied |
| 404 | The file could not be found |
| 423 | The file cannot open because it is locked by the operating system |

## Automation

This set of commands allows for the manipulation of input on the server machine. These allow for the virtualization of mouse movements, keyboard strokes, process handling, and others.

### BLOCK

Replaces an application with a message box declaring that the application has been blocked. This command requires elevated privileges.

BLOCK "EXECUTABLE" [ /M "FLAG" "TEXT" "TITLE" ] [ /R "PATH" ] [ /D "PATH" ]

|  |  |
| --- | --- |
| Argument | Description |
| EXECUTABLE | The executable name (not the path) of the application that should be blocked. The ".exe" extension may be omitted. |
| [ /M "FLAG" "TEXT" "TITLE" ] | Replace the application with a custom message box. This takes the same arguments as the MsgBox command. |
| [ /R "FILE" ] | Redirects the application to another application |
| [ /D "FILE" ] | Passes the original application as an argument to the other |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The application was blocked successfully |
| 400 | An invalid switch may have been given |
| 403 | Access to the proper registry key is denied |

### BLOCKINPUT

Blocks or unblocks the input on the server machine. This command is overridden by Ctrl+Alt+Del and requires elevated privileges.

BLOCKINPUT FLAG

|  |  |
| --- | --- |
| Argument | Description |
| FLAG | 1 – Blocks input  0 – Unblock input |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The application was blocked successfully |
| 500 | Access may be denied |

### BLOCKEDLIST

Retrieves a list of all blocked applications. This command does not require administrative privileges.

BLOCKEDLIST

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | Lists all of the applications that have been blocked |
| 204 | No applications have been blocked |
| 403 | Access to the proper registry key is denied |

### GETSCREEN

Takes a screenshot of the server machine. This command converts the file as a Base64 string before sending the data.

GETSCREEN

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The screen has been captured successfully |

### MOUSECLICK

Emulates a mouse click on the server. This invokes a call to “user32.dll”.

MOUSECLICK "BUTTON" "X" "Y" [ CLICKS ] [ SPEED ]

|  |  |
| --- | --- |
| Argument | Description |
| Button | Can be "LEFT" or "RIGHT" |
| X | The X coordinate of the mouse |
| Y | The Y coordinate of the mouse |
| [ CLICKS ] | The number of clicks to send to the server |
| [ SPEED ] | The speed at which to move the cursor – the higher the slower  (default = 1)  A value of 0 moves the cursor immediately |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The mouse click was successful |

### 

### MOUSEMOVE

Emulates a mouse movement on the server. This invokes a call to “user32.dll”

MOUSEMOVE "X" "Y" [ SPEED ]

|  |  |
| --- | --- |
| Argument | Description |
| X | The X coordinate of the mouse |
| Y | The Y coordinate of the mouse |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The mouse move was successful |
| [ SPEED ] | The speed at which to move the cursor – the higher the slower  (default = 1)  A value of 0 moves the cursor immediately |

### PROCESSCLOSE

Sends a message to a process asking it to close

PROCESSCLOSE "PROCESS"

|  |  |
| --- | --- |
| Argument | Description |
| PROCESS | The name of the process WITHOUT the ".exe" extension |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The process was asked to close |
| 404 | The process could not be found |

### PROCESSEXISTS

Checks if a process exits. Returns “True” or “False”

PROCESSEXISTS "PROCESS"

|  |  |
| --- | --- |
| Argument | Description |
| PROCESS | The name of the process to check |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The process was or was not found |

### PROCESSKILL

Kills a process without asking it to close

PROCESSKILL "PROCESS"

|  |  |
| --- | --- |
| Argument | Description |
| PROCESS | The name of the process to kill |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The server attempted to close the process |
| 404 | The process was not found |

### REMOVEX

Removes the close button on the upper left of a window.

REMOVEX "HANDLE"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The close button was removed |

### UNBLOCK

Unblocks an application that has been blocked using the “BLOCK” command

UNBLOCK "EXECUTABLE"

|  |  |
| --- | --- |
| Argument | Description |
| EXECUTABLE | The name of the executable to unblock (The ".exe" extension can be omitted) |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The application was blocked successfully |
| 403 | Access to the proper registry key is denied |
| 404 | The application may not be blocked |

### SEND

Sends keystrokes to the host machine. This command uses special formatting. Certain special characters are used as hotkeys. Place characters in {} if you are unsure. (e.g. {!})

SEND "TEXT"

|  |  |
| --- | --- |
| Argument | Description |
| TEXT | The formatted text to send to the server |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The text was successfully sent |
| 500 | The characters you entered may not have been recognized. Try putting special characters in {}. |

#### Special Characters for SEND

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Code** | **Key** | **Code** | **Key** | **Code** |
| BACKSPACE | {BACKSPACE}, {BS}, {BKSP} | RIGHT ARROW | {RIGHT} | F8 | {F8} |
| BREAK | {BREAK} | SCROLL LOCK | {SCROLLOCK} | F9 | {F9} |
| CAPSLOCK | {CAPSLOCK} | TAB | {TAB} | F10 | {F10} |
| DELETE | {DELETE}, {DEL} | PAGE UP | {PGUP} | F11 | {F11} |
| DOWN ARROW | {DOWN} | PRINT SCREEN | {PRTSC} | F12 | {F12} |
| END | {END} | RIGHT ARROW | {RIGHT} | F13 | {F13} |
| ENTER | {ENTER} or ~ | SCROLL LOCK | {SCROLLOCK} | F14 | {F14} |
| ESC | {ESC} | TAB | {TAB} | F15 | {F15} |
| HELP | {HELP} | UP ARROW | {UP} | F16 | {F16} |
| HOME | {HOME} | F1 | {F1} | Keypad add | {ADD} |
| INSERT | {INSERT}, {INS} | F2 | {F2} | Keypad subtract | {SUBTRACT} |
| LEFT ARROW | {LEFT} | F3 | {F3} | Keypad multiply | {MULTIPLY} |
| NUMLOCK | {NUMLOCK} | F4 | {F4} | Keypad divide | {DIVIDE} |
| PAGE DOWN | {PGDN} | F5 | {F5} | SHIFT\* | + |
| PAGE UP | {PGUP} | F6 | {F6} | CTRL\* | ^ |
| PRINT SCREEN | {PRTSC} | F7 | {F7} | ALT\* | % |

\* These characters act as hotkeys and can be combined with other keys

### VOLUMEDOWN

Presses the volume down keyboard key a set number of times

VOLUMEDOWN [ COUNT ]

|  |  |
| --- | --- |
| Argument | Description |
| COUNT | The number of times to press the key.  Default = 1 |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The button was successfully pressed |
| 500 | There may be an issue with accessing the proper dll |

### VOLUMEMUTE

Presses the volume mute keyboard

VOLUMEMUTE

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The button was successfully pressed |
| 500 | There may be an issue with accessing the proper dll |

### VOLUMEUP

Presses the volume up keyboard key a set number of times

VOLUMEUP [ COUNT ]

|  |  |
| --- | --- |
| Argument | Description |
| COUNT | The number of times to press the key.  Default = 1 |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The button was successfully pressed |
| 500 | There may be an issue with accessing the proper dll |

## Window Commands

This set of commands is used to manipulate windows. These commands are best run on the current user.

### WINACTIVATE

Brings a window to the foreground (Top of the z-order)

WINACTIVATE "HANDLE"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window to activate |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The window was brought to the front |
| 500 | The window may not exist |

### WINCLOSE

Sends a message to a window to close

WINCLOSE "HANDLE"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window to close |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The window was closed |
| 500 | The window may not exist |

### WINGETHANDLE

Returns a decimal handle for a window

WINGETHANDLE "TITLE"

|  |  |
| --- | --- |
| Argument | Description |
| TITLE | The title of the window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The handle was returned |
| 500 | The window may not exist |

### WINGETTITLE

Returns the title of a window from a given handle

WINGETTITLE "HANDLE"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The title has been retrieved successfully |
| 500 | The window may not exist |

### WININFO

Lists basic information about a window. (Title, size, location, handle)

WININFO "HANDLE"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The window information has been retrieved |
| 500 | The window may not exist |

### WINKILL

Kills a window without asking it to close

WINKILL "HANDLE"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The window has been killed |
| 500 | The window may not exist |

### WINLIST

Returns a list of the titles of all windows in a current state

WINLIST [ STATE ]

|  |  |
| --- | --- |
| Argument | Description |
| STATE | Acceptable values are:  EXISTING  VISIBLE (default)  ENABLED  ACTIVE  MINIMIZED  MAXIMIZED |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The list of windows has been returned successfully |
| 204 | There are no windows to report |

### WINLISTINFO

Returns a list of basic info about windows in a current state

WINLISTINFO [ STATE ]

|  |  |
| --- | --- |
| Argument | Description |
| STATE | Acceptable values are:  EXISTING  VISIBLE (default)  ENABLED  ACTIVE  MINIMIZED  MAXIMIZED |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The list of windows has been returned successfully |
| 204 | There are no windows to report |

### WINMOVE

Moves a window to another location

WINMOVE "HANDLE" "X" "Y"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |
| X | The X coordinate of the upper left corner of the window |
| Y | The Y coordinate of the upper left corner of the window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The window has been moved |
| 500 | The window may not exist |

### 

### WINPICTURE

Returns a capture of a window (Works well with Windows Aero). This command returns its data as a Base64 string and must be converted to a byte array before use.

WINPICTURE "HANDLE" "QUALITY"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |
| QUALITY | The quality of the image (1-100). Lower qualities are more compressed and faster to send |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The window has been returned |
| 500 | The window may not exist |

### 

### WINSETSTATE

Sets the state of a window

WINSETSTATE "HANDLE" "STATE"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |
| STATE | See [List of Window States](#_List_of_Window) for acceptable values |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The window state has been set |
| 500 | The window may not exist |

#### List of Window States

|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **Action** | **State** | **Action** |
| SW\_HIDE | Hides a window | SW\_SHOWMINNOACTIVE | Shows a window minimized and not active |
| SW\_MAXIMIZE | Maximizes a window | SW\_SHOWNA | Shows the window but does not activate it |
| SW\_MINIMIZE | Minimizes a window | SW\_SHOWNOACTIVATE | Shows the window in its most recent size and position |
| SW\_RESTORE | Restores a window | SW\_SHOWNORMAL | Shows the window in its default state |
| SW\_SHOW | Shows a window | SW\_SHOWMINIMIZED | Shows a window and minimizes it |
| SW\_SHOWMAXIMIZED | Shows a window and maximizes it | SW\_SHOWMINNOACTIVE | Shows a window minimized and not active |
| SW\_SHOWMINIMIZED | Shows a window and minimizes it |  |  |

### 

### WINSETTITLE

Sets the title of a window from a given handle

WINSETTITLE "HANDLE" "TITLE"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |
| TITLE | The new title of the window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The window title has been set |
| 500 | The window may not exist |

### WINSETTRANS

Sets the window transparency

WINSETTRANS "HANDLE" "TRANSPARENCY"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |
| TRANSPARENCY | A byte value (0-255) for the transparency of the window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The transparency has been set |
| 500 | The window may not exist |

### WINSIZE

Resizes the window

WINSIZE "HANDLE" "WIDTH" "HEIGHT"

|  |  |
| --- | --- |
| Argument | Description |
| HANDLE | The decimal handle of the window |
| WIDTH | The width of the window |
| HEIGHT | The height of the window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The size of the window has been set |
| 500 | The window may not exist |

## Registry Commands

These commands manipulate registry data. Some registry entries require elevated privileges, and registry entries should not be changed unless you have backed it up first.

### REGCREATEKEY

Creates a registry key

REGCREATEKEY "PATH" "KEY"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The registry path to place the new key |
| KEY | The new key to create |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 201 | The key was successfully created |
| 403 | Access to the key is denied |

### REGDELETE

Deletes a registry value

REGDELETE "PATH" "NAME"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The registry path of the value |
| NAME | The name of the value to delete |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The entry was deleted |
| 403 | Access to the key is denied |

### REGDELETEKEY

Deletes a registry key and all its child keys

REGDELETEKEY "PATH" "KEY"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The registry path of the parent key |
| KEY | The key that is to be deleted |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The key was successfully deleted |
| 403 | Access to the key is denied |

### REGENUMKEYS

Gets a list of all the sub-keys in a given parent

REGENUMKEYS "PATH"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The registry path to the parent key |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The sub-keys were returned successfully |
| 403 | Access to the key is denied |

### REGENUMVALUES

Gets a list of the values in a given key

Note that this command uses the special formatting that is normally reserved for messages sent TO the server (See [*List of Special Characters*](#_List_of_special))

REGENUMVALUES "PATH" [ /S ]

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The registry key to enumerate |
| [ /S ] | Show the values in simple form  (i.e. NAME|TYPE|VALUE) |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The values have been enumerated |
| 403 | Access to the key is denied |

### REGREAD

Reads the information in a registry value

REGREAD "PATH" "NAME"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The registry key that contains the values |
| NAME | The name of the value to read |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The value has been returned |
| 403 | Access to the key is denied |

### 

### REGRENAME

Renames a registry value

REGRENAME "PATH" "NAME1" "NAME2"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The registry key which contains the value |
| NAME1 | The name of the value to rename |
| NAME2 | The new name for the value |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The value has been renamed |
| 403 | Access to the key is denied |

### REGRENAMEKEY

Renames a registry key

REGRENAMEKEY "PATH" "NAME1" "NAME2"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The registry key which contains the child key |
| NAME1 | The name of the key to rename |
| NAME2 | The new name for the key |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The value has been renamed |
| 403 | Access to the key is denied |

### REGWRITE

Writes a value to the registry

REGWRITE "PATH" "NAME" "VALUE" "TYPE"

|  |  |
| --- | --- |
| Argument | Description |
| PATH | The registry key that should contain the value |
| NAME | The name of the entry |
| VALUE | The value for the entry |
| TYPE | Acceptable types are:  BINARY; DWORD; EXPANDSTRING; MULTISTRING; QWORD; STRING |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The value has been renamed |
| 403 | Access to the key is denied |

## Messaging and Communication

This set of commands is used for sending messages to the user of the host machine.

### BLOCKMESSAGE

Covers the screen with a translucent overlay. This command creates a child process and uses a named pipe to communicate with the block message window (BLOCKMESSAGEPIPE).

BLOCKMESSAGE "TEXT" [ LOCAL | REMOTE ]

|  |  |
| --- | --- |
| Argument | Description |
| TEXT | The text of the message to display |
| [ LOCAL | REMOTE ] | Specifies whether the message should be run as a child process (REMOTE) or run on the main process (LOCAL). Uses REMOTE by default. |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The message has been opened |
| 409 | A block message window is already opened |

BLOCKMESSAGE operates on BLOCKMESSAGEPIPE. Use PIPEUSE to send commands to BLOCKMESSAGEPIPE.

It accepts the following commands:

CLOSE

Closes the message

GETTEXT

Gets the current text of the message

SETTEXT "TEXT"

Sets the text of the message

### CHAT

Creates a chat dialogue to communicate on the server. This creates a child process.

The new chat window communicates with the main program through a named pipe (CHATPIPE). Here are the commands that the CHATPIPE listens for:

CHAT "TEXT"

|  |  |
| --- | --- |
| Argument | Description |
| TEXT | The initial text to send in the CHAT window |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The chat window has been opened |
| 409 | A chat window may already be opened |

CHAT operates on CHATPIPE. Use PIPEUSE to send commands to the CHATPIPE.

It accepts the following commands:

CLOSE

Closes the message

GETNEWTEXT

Retrieves only newly placed text in the chat window

GETTEXT

Gets all the text of the window

SETTEXT "TEXT"

Adds a new message to the chat window (where "TEXT" is the text to send)

### MSGBOX

Creates a message box on the server machine.

This command acts differently when run in a TBASIC script. Scripts are able to handle the buttons that a user clicks, whereas sending this command directly to the server will result in a “202 Accepted”. In a script, the command will return a 200 status code and the button that was clicked.

MSGBOX "FLAG" "TEXT" "TITLE"

|  |  |
| --- | --- |
| Argument | Description |
| FLAG | The flag for the message box. Add values together to combine results. |
| TEXT | The text for the message box |
| TITLE | The title of the message box |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | In a TBASIC script, this will return the button that was clicked |
| 202 | The message box was created as a child process |

The flag parameter is a combination of the following values:

|  |  |  |  |
| --- | --- | --- | --- |
| Flag | Button Result | Flag | Icon Result |
| 0 | OK | 0 | (No Icon) |
| 1 | OK, Cancel | 16 | Error |
| 2 | Abort, Retry, Ignore | 32 | Question |
| 3 | Yes, No, Cancel | 48 | Warning |
| 4 | Yes, No | 64 | Information |
| 5 | Retry, Cancel |  |  |

### SAY

Uses Microsoft SAPI to convert text to speech. This will speak in a new thread and therefore will not wait for a success before continuing.

SAY "TEXT"

|  |  |
| --- | --- |
| Argument | Description |
| TEXT | The text to speak |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The command was accepted, but the server does not know if errors were encountered |

### TRAYTIP

Creates a balloon tip as a tray icon. This command creates a new thread and therefore will not wait for a success before continuing.

TRAYTIP "TIMEOUT" "TEXT" [ TITLE ] [ ICON ]

|  |  |
| --- | --- |
| Argument | Description |
| TIMEOUT | How long (in milliseconds) the balloon will be shown |
| TEXT | The text of the balloon |
| [ TITLE ] | The title for the balloon (None by default) |
| [ ICON ] | Acceptable values are:  0 – No icon (default)  1 – Information  2 – Question  3 – Warning  4 - Error |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | The command was accepted, but the server does not know if errors were encountered |

## 

## Thread Commands

When TCPSERVR connects to its clients, it spins them off into separate threads. These commands allow you to view the status of these thread connections.

### THREADABORT

Aborts a thread. Use this to close threads that have become stuck.

THREADABORT "ID"

|  |  |
| --- | --- |
| Argument | Description |
| ID | The ID of the thread to abort |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 202 | An abort message was sent to the thread |
| 404 | No thread is being managed with that ID |

### THREADDELETE

Clears a stopped thread from the list. This cannot delete an active thread.

THREADDELETE "ID"

|  |  |
| --- | --- |
| Argument | Description |
| ID | The ID of the thread to clear |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The thread was removed from the list |
| 404 | No thread is being managed with that ID |
| 409 | The thread is running and cannot be deleted |

### THREADLIST

Create a list of all threads that TCPSERVR is explicitly managing. This lists all thread IDs and clients connected to those threads. This includes threads that have been disconnected and are no longer running.

THREADLIST

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | A list of all threads has been returned |

### THREADLISTCLEAN

Removes all threads that have been stopped on the list. This sends the THREADDELETE command to all dead threads. This command is not necessary for memory conservation, but helps with organizational purposes.

THREADLISTCLEAN

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | All empty threads have been removed |

### THREADSTATE

Gets the current task that the thread is completing. This is useful for viewing the task a thread may be completing if it appears it is hanging.

THREADSTATE "ID"

|  |  |
| --- | --- |
| Argument | Description |
| ID | The ID of the thread |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The state of the thread has been returned |
| 404 | No thread is being managed with that ID |

## Pipe Commands

These commands are used for sending and receiving data on pipes. The messages sent to the pipes are formatted with the generic byte and string headers.

### PIPECREATE

Create a pipe with the ability to initiate commands

PIPECREATE "ID" "NAME"

|  |  |
| --- | --- |
| Argument | Description |
| ID | The pipe ID of the initial application  (e.g. pipe326) |
| NAME | The name of the pipe you wish to create |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The pipe was created and is ready for communication |
| 404 | The pipe ID does not exist |
| 501 | You are using a slave application |

### PIPELISTUSERS

List all users running submissive applications of TCPSERVR and their respective pipes

PIPELISTUSERS [ /S ]

|  |  |
| --- | --- |
| Argument | Description |
| [ /S ] | Display a list without formatting (separated by pipes "|") |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | A list of pipes has been returned |
| 501 | You are using a slave application |

### 

### PIPEUSE

Send a single command to a pipe, formatted in TCPSERVR style. This will send the status code of the command sent to the pipe and not its own.

PIPEUSE "NAME" "COMMAND" [ ARGUMENTS ]

|  |  |
| --- | --- |
| Argument | Description |
| NAME | The name of the pipe to use |
| COMMAND | The command to send to the pipe |
| [ ARGUMENTS ] | Any arguments for the command |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| ANY | The PIPEUSE command is relaying a status from the pipe with which it was communicating |
| 404 | The pipe does not exist |
| 502 | An invalid message was received from the pipe and cannot be processed |

### USER

Creates a pipe to operate on a specified user. If the pipe already exists, it defaults to that pipe.

USER "USERNAME" [ TRUE | FALSE ]

|  |  |
| --- | --- |
| Argument | Description |
| USERNAME | The name of the user on which to operate |
| [ TRUE | FALSE ] | Optionally set whether all future connections should use this pipe (Default = FALSE) |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The server is now operating on that user |
| 404 | The user cannot be found |

### USINGPIPE

Set a pipe to send all future commands. When no commands are given, it displays whether a pipe is currently in use.

USINGPIPE [ NAME ] [ TRUE | FALSE ]

|  |  |
| --- | --- |
| Argument | Description |
| [ NAME ] | The name of the pipe to use |
| [ TRUE | FALSE ] | TRUE states that all future connections should use this pipe. FALSE states that only this connection will use it. |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | Whether a pipe is being used and its name has been returned |
| 202 | All commands will be sent through the given pipe |

# Scripting Commands

These commands are exclusive to scripting and cannot be accessed by a normal socket.

## Data Declaration

### DELVAR

Remove a variable from memory. This is considered a statement and therefore is not enclosed in parentheses. It cannot be nested as a parameter and does not return a value.

DELVAR "VARIABLE"

|  |  |
| --- | --- |
| Argument | Description |
| VARIABLE | The variable to remove (without the trailing "$") |

### DIM

Declare and evaluate variables. This is mainly used to allocate arrays.

The code snippet below achieves the same functionality as LET.

DIM VARIABLE$ = VALUE

However, the following code snippet will allocate a single dimensional array with a specified bound. This cannot be directly assigned a value in the DIM statement.

DIM VARIABLE$[LENGTH]

Example:

DIM VARIABLE$[10] will allocate a single dimensional array with 10 elements starting at 0.

### LET

Declare and evaluate variables. Variables should end with ‘$’. This command allows the use of basic mathematic functions such as addition, subtraction, multiplication, and division (+, -, /, \*). This command allows modular arithmetic (%).

LET VARIABLE$ = VALUE

|  |  |
| --- | --- |
| Argument | Description |
| VARIABLE$ | The variable to set |
| VALUE | The value of the variable |

LET can also be used to join strings.

LET VARIABLE$ = STRING1 [ + STRING2 + ... ]

|  |  |
| --- | --- |
| Argument | Description |
| VARIABLE$ | The variable to set |
| STRING | Any strings to join |

### RETURN

Changes the value of the @return macro for a user defined function. This will immediately end the function and return to the code block which executed it.

RETURN "DATA"

|  |  |
| --- | --- |
| Argument | Description |
| DATA | The data to return |

### SETSTATUS

Changes the value of the @status macro for a user defined function. Unlike RETURN, this will not immediately exit the function.

SETSTATUS STATUS

|  |  |
| --- | --- |
| Argument | Description |
| STATUS | The status code of the function. Can be between 0 and 999 |

## String Manipulation

### APPEND

Attaches two data objects, such as strings, and stores it in the @return macro. The second object will be appended to the end of the first.

APPEND("OBJECT1","OBJECT2")

|  |  |
| --- | --- |
| Argument | Description |
| OBJECT1 | The first object |
| OBJECT2 | The object to append to the first object |

### CONTAINS

Determines whether a string contains another string. This will store either TRUE or FALSE into the @return macro.

CONTAINS("STRING1", "STRING2")

|  |  |
| --- | --- |
| Argument | Description |
| STRING1 | The initial string |
| STRING2 | The string to search |

### INDEX

Retrieves the location of another string within a string and stores it in the @return macro. If the search string is not found, it stores -1.

INDEX("STRING1", "STRING2")

|  |  |
| --- | --- |
| Argument | Description |
| STRING1 | The initial string |
| STRING2 | The string to search |

### LEFT

Removes characters starting from the left of a string and stores it in the @return macro.

LEFT("STRING", START, COUNT)

|  |  |
| --- | --- |
| Argument | Description |
| STRING | The string from which to remove the characters |
| START | The initial position (from the left) starting at 0 |
| COUNT | The number of characters to remove |

### 

### LEN

Retrieves the length of a string or size of a variable and stores it in the @return macro. This function can also determine the length of an array.

LEN("OBJECT")

|  |  |
| --- | --- |
| Argument | Description |
| OBJECT | The data object to measure |

Example:

If the object evaluated were "Test String 1", then LEN will return 13, because that is the length of the object.

However, if the object evaluated is var$[10], LEN will return 10, because that is the length of the array.

### LOWER

Makes a string all lower case and stores it in the @return macro. This is useful for making case insensitive comparisons.

LOWER("STRING")

|  |  |
| --- | --- |
| Argument | Description |
| STRING | The initial string |

### TAKE

Gets a substring within a string and stores it in the @return macro.

TAKE("STRING", START, COUNT)

|  |  |
| --- | --- |
| Argument | Description |
| STRING | The initial string |
| START | The initial position (from the left) starting at 0 |
| COUNT | The number of characters to take |

### UPPER

Makes a string all upper case and stores it in the @return macro. This is useful for making case insensitive comparisons.

UPPER("STRING")

|  |  |
| --- | --- |
| Argument | Description |
| STRING | The initial string |

## Mathematics

### EVAL

Evaluates a mathematical expression and stores its result in the @return macro.

EVAL("EXPRESSION")

|  |  |
| --- | --- |
| Argument | Description |
| EXPRESSION | The expression to evaluate  Example:  (5 + 6) \* 7 |

### EVALBOOL

Evaluates a Boolean expression and stores its result in the @return macro.

EVALBOOL("EXPRESSION")

|  |  |
| --- | --- |
| Argument | Description |
| EXPRESSION | The Boolean expression to evaluate  Example:  (True AND False) OR True |

### FPART

Removes the integer part of a decimal and stores the fraction in the @return macro.

FPART(DECIMAL)

|  |  |
| --- | --- |
| Argument | Description |
| DECIMAL | The decimal number to truncate |

### IPART

Removes the fractional part of a decimal and stores the integer in the @return macro.

IPART(DECIMAL)

|  |  |
| --- | --- |
| Argument | Description |
| DECIMAL | The decimal number to truncate |

### NOT

Evaluates a Boolean expression and stores its inverse in the @return macro.

NOT("EXPRESSION")

|  |  |
| --- | --- |
| Argument | Description |
| EXPRESSION | The Boolean expression to evaluate  Example:  (True AND False) OR True |

### POW

Sets a base to a certain power and stores its result in the @return macro.

POW(BASE, POWER)

|  |  |
| --- | --- |
| Argument | Description |
| BASE | The base of the exponent |
| POWER | The power to raise the base |

### RANDOM

Generates a pseudo-random number between 0 and 1 (eight decimal places)

RANDOM()

### ROUND

Rounds a decimal to a set number of decimal places and stores its result in the @return macro.

ROUND(DECIMAL [, PLACES ])

|  |  |
| --- | --- |
| Argument | Description |
| DECIMAL | The initial decimal to round |
| [ PLACES ] | The number of places to round the decimal  default = 2 |

## Script Processing

### DO

Evaluate an expression and carry out commands until a certain condition is met.

See [*Loops*](#_Loops) for more details.

### IF

Use this command to evaluate a logical expression and proceed with commands if the output is true. See [*IF Statements*](#_IF_Statements) for more details.

### BREAK

Stops the execution of an IF, WHILE, or DO block.

### EXIT

Ends the script

### WHILE

Evaluate an expression and carry out commands while a certain condition is met.

See [*Loops*](#_Loops) for more details.

## Miscellaneous

### INPUT

Create an input box for the user to type data. If the use presses “Cancel”, it will return “No Input” with a status of “204”. Otherwise, the status will be “200”, and the response will be the input string.

INPUT("PROMPT", "TITLE", WIDTH, HEIGHT)

|  |  |
| --- | --- |
| Argument | Description |
| PROMPT | The prompting text for the input box |
| TITLE | The title of the input box |
| WIDTH | The width of the box |
| HEIGHT | The height of the box |

Possible return values:

|  |  |
| --- | --- |
| Status | Description |
| 200 | The input has been returned |
| 204 | The cancel button was pressed |

### SLEEP

Require the script to wait for a period of time.

SLEEP(TIME)

|  |  |
| --- | --- |
| Argument | Description |
| TIME | The wait time (in milliseconds) |

Copyright © 2011-2014 Timothy Baxendale All Rights Reserved

This document was written and researched by Timothy Baxendale.

This manual, TCPSERVR, and all diagrams and screenshots are the intellectual and creative property of Timothy Baxendale. Any attempt to decompile the source code or distribute the application without proper recognition is prohibited.

7-Zip, Microsoft, Windows, and others are registered trademarks of their respective organizations. External links are subject to change, and we are not liable for any information they provide.